Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп’ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп’ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістра
(назва освітнього ступеня)

на тему: Метод побітового сортування даних в комп’ютерних системах

Виконав(ла): студент(ка) 6 курсу, групи СІм-61
спеціальності 123 «Комп’ютерна інженерія»
(шифр і назва спеціальності)

(підпис) Семеген В. В. (прізвище та ініціали)

Керівник (підпис) Луцик Н. С. (прізвище та ініціали)

Нормоконтроль (підпис) Тиш Є. В. (прізвище та ініціали)

Завідувач кафедри (підпис) Осухівська Г.М. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2021

Міністерство освіти і науки України

Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп’ютерних систем та мереж
(повна назва кафедри)

ЗАТВЕРДЖУЮ

Завідувач кафедри

________ Осухівська Г.М.
(підпис) (прізвище та ініціали)

« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра
(назва освітнього ступеня)

за спеціальністю 123 «Комп’ютерна інженерія»
(шифр і назва спеціальності)

студенту Семегену Віталію Васильовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Метод побітового сортування даних в комп’ютерних системах

Керівник роботи Луцик Надія Степанівна, доцент кафедри, кандидат технічних наук
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*28*» *жовтня* 2021 року № *4-7/976*

2. Термін подання студентом завершеної роботи 15.12.2021 р.

3. Вихідні дані до роботи Алгоритми сортування та структури даних, мова програмування Асемблер

4. Зміст роботи (перелік питань, які потрібно розробити)

1. Аналітична частина
2. Математичні методи алгоритму побітового сортування даних
3. Програмна реалізація методу побітового сортування даних
4. Охорона праці та безпека в надзвичайних ситуаціях

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)

1. Актуальність роботи, мета і завдання дослідження
2. Об’єкт, предмет, методи дослідження, наукова новизна і практичне значення
3. Блок-схема алгоритму побітового сортування даних
4. Структура даних для сортування списку елементів
5. Інтерфейс програми сортування даних та інтерфейс генератора
6. Вхідний файл із даними та файл із утвореною структурою даних
7. Відсортований перелік елементів
8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Г.М., зав. каф. КС		
Безпека в надзвичайних ситуаціях	Стадник І.Я., проф. каф. ОХ		

7. Дата видачі завдання 28 жовтня 2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	28.10.2021 р.	Виконано
2	Аналіз завдання	29.10.2021 р.	Виконано
3	Виконання розділу 1	09.11.2021 р.	Виконано
4	Виконання розділу 2	14.11.2021 р.	Виконано
5	Виконання розділу 3	19.11.2021 р.	Виконано
6	Виконання розділу 4	26.11.2021 р.	Виконано
7	Оформлення пояснювальної записки	01.12.2021 р.	Виконано
8	Оформлення графічного та презентаційного матеріалу	09.12.2021 р.	Виконано
9	Перевірка роботи на антиплагіат	14.12.2021 р.	Виконано
10	Попередній захист	15.12.2021 р.	Виконано
11	Захист		

Студент ______________ Семеген В. В.
(підпис) (прізвище та ініціали)

Керівник роботи ______________ Луцик Н. С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Метод побітового сортування даних в комп'ютерних системах // Кваліфікаційна робота магістра// Семеген Віталій Васильович // ТНТУ, комп’ютерна інженерія, група СІм-61 // Тернопіль, 2021 // С. – 98, рис. – 42, табл.. – 2, аркушів А1 – 8, додат. – 2, бібліогр. – 24.

Ключові слова: СОРТУВАННЯ ДАНИХ, ОБЧИСЛЮВАЛЬНА СКЛАДНІСТЬ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ОПРАЦЮВАННЯ ДАНИХ, СТРУКТУРИ.

У кваліфікаційній роботі розроблено алгоритм сортування даних та його практична реалізація. У алгоритмі оптимізовано кількість всіх порівнянь, при сортуванні, і відсутні перестановки елементів через виділення додаткової пам'яті у якій формується оптимальна адресна структура даних і з неї вкінці опрацювання зчитується відсортований масив даних.

Мовою програмування Assembler розроблено програмне забезпечення з графічним інтерфейсом, яке сортує текстові дані у файлі.

Представлено результати теоретичної та практичної швидкодії алгоритму побітового сортування даних.

ANNOTATION

Bitwise data sorting method in computer systems // Qualification work of the master // TNTU, Computer Engineering // Semehen Vitalii // group SIm-61

// Ternopil, 2021 // p. – 98, fig. – 42, tab. – 2, Sheets A1 – 8, Add – 2, Ref. – 24.

Keywords: DATA SORTING, COMPUTATIONAL COMPLEXITY, SOFTWARE, DATA PROCESSING, STRUCTURES.

The algorithm of data sorting and its partical realization was developed in the qualification work. The algorithm optimizes the number of all comparisons during sorting, and there are no permutations of elements due to the allocation of additional memory in which the optimal address data structure is formed and from it at the end of processing reads the sorted data array.

GUI software that sorts text data into a file was developed using the Assembler programing language.

The results of theoretical and practical performance of the bit data sorting algorithm are presented.

СПИСОК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

СД – сортування даних

МПСД – метод побітового СД;

ПЗ – програмне забезпечення;

ЧС – часова складність.

API (Application Programming Interface) – набір підпрограм та протоколів взаємодії, що використовуються для розробки програм та їх модулів.

ЗМІСТ

ВСТУП... 9
РОЗДІЛ 1. АНАЛІТИЧНА ЧАСТИНА 12
1.1 Типи алгоритмів і їх характеристики 12
1.2 Аналіз існуючих алгоритмів сортування даних 14
1.3 Алгоритмічна складність 20
1.4 Застосування алгоритмів сортування даних 22
1.5 Висновки до розділу 1 24
РОЗДІЛ 2. МАТЕМАТИЧНІ МЕТОДИ АЛГОРИТМУ ПОБІТОВОГО СОРТУВАННЯ ДАНИХ.. 25
2.1 Опис покращення алгоритмів сортування даних 25
2.2 Розробка алгоритму побітового сортування даних 33
2.3 Висновки до розділу 2 42
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДУ ПОБІТОВОГО СОРТУВАННЯ ДАНИХ... 43
3.1 Опис коду програмного забезпечення 43
3.2 Результати тестування 48
3.3 Висновки до розділу 3 61
РОЗДІЛ 4. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ 62
4.1. Охорона праці 62
4.2. Організація та забезпечення заходів щодо розосередження робітників та службовців суб’єктів господарювання, що продовжують свою роботу в особливий період, і евакуація населення. 64
4.3. Висновки до розділу 4 66
ВИСНОВКИ 67
СПИСОК ЛІТЕРАТУРИ 68
Додаток А 71

Додаток Б.. 76

ВСТУП

Актуальність роботи. На даний час кількість інформації зростає експоненційно з кожним роком. Стає важливим розробка все нових і більш оптимальних алгоритмів роботи з даними для їх швидкого опрацювання і отримання результатів.

Виробники комп’ютерів 1960-х підрахували що більш ніж 25% часу роботи їх комп'ютерів витрачалося на сортування даних [5].

На сьогоднішній день відсоток ймовірно змінився, але все ж залишився значним. Сучасні комп'ютерні системи для своєї роботи використовують велику кількість програмного забезпечення, яка включає у свою роботу системи управління базами даних, пошукові системи, інтерфейси взаємодії із користувачем. Для пошуку, опрацювання і представлення результатів використовують алгоритми сортування даних. Для таких систем важлива швидкодія, тому для оптимальної роботи вибираються кращі серед існуючих алгоритмів сортування даних.

На даний час розроблено чимало алгоритмів сортування даних серед яких велика кількість знаходить своє практичне застосування у різних засобах опрацювання інформації [4].

Недоліками відомих алгоритмів сортування є складність, або мала швидкодія. Тому розробка простого і швидкого алгоритму сортування даних для комп’ютерних систем є актуальною та важливою задачею. Тому розробка простого і швидкого алгоритму сортування даних для комп’ютерних систем є актуальною та важливою задачею.

Мета і завдання дослідження. Розробка методу сортування даних та програмного забезпечення для комп’ютерної системи.

Для досягнення цієї мети потрібно виконати :

1. Провести аналіз відомих методів сортування даних для обґрунтування напряму наукового дослідження.

2. Побудувати математичну модель сигналу у комп'ютерних мережах для задачі його оптимального детектування на фоні завад.

3. Побудувати математичну модель із покращеними характеристиками алгоритму сортування даних.

4. Розробити алгоритм побітового сортування даних.

5. Розробити програмне забезпечення для комп'ютерного засобу оптимального сортування даних.

6. Провести тестування щодо швидкості роботи програмного забезпечення і представлення результатів у порівняні із іншими алгоритмами.

Об'єкт дослідження. Процес адресного сортування даних в оперативній пам'яті.

Предмет дослідження. Алгоритми швидкого сортування даних.

Методи дослідження. Аналіз літератури та існуючих алгоритмів сортування даних із виділенням особливостей їх роботи та оптимізацією методів, які використовувалися у них для швидкого впорядкування даних.

Наукова новизна одержаних результатів.

Вперше розроблено алгоритм сортування даних у якому оптимізовано деякі етапи процесу сортування даних та написано програмне забезпечення із реалізацією даного методу побітового сортування даних.

Практичне значення одержаних результатів. Розроблено швидкий алгоритм і прикладне програмне забезпечення із низьким навантаження на комп'ютерні системи, яке дає змогу на малопотужних пристроях швидко сортувати інформацію, для задач класифікації та впорядкування даних.

Публікації. Результати дослідження апробовано на IX Науково-технічній конференції «Інформаційні моделі, системи та технології» (8–9 грудня 2021 року) Тернопільського національного технічного університету імені Івана Пулюя у вигляді тез конференцій [1,2].

Структура роботи. До складу кваліфікаційної роботи магістра входить розрахунково-пояснювальна записка та графічний матеріал. Розрахунково-

пояснювальна записка містить вступ, 4 розділи, загальні висновки, список використаної літератури і додатки. Обсяг роботи: розрахунково-пояснювальна записка – 98 арк. формату А4, графічна частина – 8 аркушів формату А1.

РОЗДІЛ 1
АНАЛІТИЧНА ЧАСТИНА

1.1 Типи алгоритмів і їх характеристики

Основні властивості алгоритмів сортування.

Стійкість – стійке сортування для випадку елементів з однаковими ключами не змінює їхнього взаємного розташування. Стабільні алгоритми є найбільш корисними для застосування, оскільки на практиці до самих елементів прикріплена додаткова інформація і зміна порядку елементів може виявитися критичною у певних випадках.

Природність поведінки – ефективність методу при обробці вже впорядкованих або частково впорядкованих даних. Алгоритм поводиться природно, якщо враховує цю характеристику вхідної послідовності та працює краще[3].

Використання операцій порівняння. Алгоритми, які використовують для сортування порівняння елементів між собою, називаються на основі порівняння. Мінімальна складність у найгіршому випадку для цих алгоритмів становить $O(n * \log(n))$, але вони відрізняються гнучкістю застосування. Для спеціальних випадків (типів даних) існують ефективніші алгоритми[4].

Алгоритми сортування за час $O(n^2)$.

Сортування вибором – шукається найбільший або найменший елемент і переміщується у впорядкованому списку в початок або його кінець.

Сортування вставкою – визначається місце де повинен знаходитися поточний елемент в упорядкованому списку і туди виконується його вставка переміщенням на одну позицію усіх елементів, які знаходяться між поточною і новою позицією елемента який опрацьовується у даний момент.

Сортування обміном, або ж сортування бульбашкою – для кожної пари індексів проводиться обмін, якщо елементи розташовані не по порядку.

Сортування методом бінарної вставки – послідовно переглядається масив елементів і кожний новий елмент вставляється до вже впорядкованої сукупності елементів яка й розширюється по мірі сортування до моменту коли буде опрацьовано останній елемент. Пошук місця вставки елементу відбувається бінарним пошуком.

Алгоритми сортування за час $O(n * \log^2(n))$.

- Сортування Шелла.
- Сортування злиттям модифіковане.

Алгоритми сортування за час $O(n * \log(n))$

- Швидке сортування.
- Пірамідальне сортування.
- Плавне сортування.
- Сортування злиттям.
- Timsort.

Алгоритми сортування за час $O(n)$ з використанням додаткової інформації про елементи:

- Сортування за розрядами.
- Сортування підрахунком.
- Сортування комірками.

Виділяють такі непрактичні алгоритми сортування.

Випадкове сортування яке має асимптотичну складність $O(n * n!)$ в середньому. Суть його полягає у тому, що потрібно довільно перемішати масив, перевірити порядок.

Сортувати за перестановкою має $O(n * n!)$ найгірший час. Для кожної пари здійснюється перевірка правильного порядку та генеруються всілякі перестановки вихідного масиву.

Гравітаційне сортування. Його складність $O(n)$, але потрібне спеціалізоване апаратне забезпечення.

Млинне сортування $O(n)$, для нього як і для попереднього алгоритму потрібне спеціалізоване апаратне забезпечення.

Алгоритми, не засновані на порівняннях.

Блокове сортування – потрібно $O(k)$ додаткової пам'яті і знання про природу даних, які сортуються, що виходить за рамки функцій “переставити” і “порівняти”. Складність алгоритму: $O(n)$.

Порозрядне сортування – складність алгоритму $O(n * k)$; потрібно $O(k)$ додаткової пам'яті.

Сортування підрахунком. Складність алгоритму за часом $O(n + k)$. Потрібно $O(n + k)$ додаткової пам'яті.

1.2 Аналіз існуючих алгоритмів сортування даних

Важливою складовою сучасних комп’ютерних систем є програмне забезпечення яке написане на основі розроблених сучасних швидких алгоритмів опрацювання даних. Однією із таких комп’ютерних систем є сервери у яких відбувається отримання, опрацювання даних і видача по ним результатів. Для опрацювання і зберігання даних часто використовуються системи управління базами даних (СУБД), у яких зберігаються різного типу дані такі як таблиці, списки, файли. Для можливості швидкого доступу до них, а також для зручного представлення використовують сучасні швидкі алгоритми сортування даних.

Швидке сортування.

Алгоритм швидкого сортування даних є одним із найпростіших, тому що використовує прості цикли, через це працює швидше ніж інші алгоритми.

Розробив цей алгоритм Тоні Гоар. Алгоритм швидкого сортування даних не потребує додаткової пам’яті. В середньому виконує O(n*log(n)) операцій і максимально O(n^2) порівнянь. Якщо порівнювати цей алгоритм з алгоритмом сортування злиттям, то він у двічі швидший.

Алгоритм розділяє елементи масиву на дві частини і виконує перестановку елементів масиву таким чином щоб з однієї сторони будь-який елемент не був більший за кожний елемент другої частини. Дане сортування виконується рекурсивно доки не будуть відсортовані усі частини на які поділено масив елементів.

Швидке сортування є нестабільним і є алгоритмом на основі порівнянь.

Збалансованість є важливою для даного алгоритму, оскільки впливає на його час роботи.

Швидкодія алгоритму залежить від розбиття, що й визначає збалансованість, яка у свою чергу залежить від опорного елемента відносно якого відбувається розбиття.

За мови збалансованості розбиття асимптотична складність алгоритму є такою ж як і в алгоритму сортування злиттям, а в найгіршому випадку швидкодія алгоритму є так само не оптимальною, що й в алгоритму сортування включенням.

Сортування двійковим деревом.

Сортування за допомогою бінарного дерева виконує побудову двійкового дерева пошуку, на основі ключів масиву, і в подальшому виконує обхід дерева та створення результуючого масиву з впорядкованих елементів.

Загальна швидкодія алгоритму.

Додавання елементу в збалансоване дерево пошуку в середньому має алгоритмічну складність O(log(n)), відповідно для всіх n елементів даного масиву складність буде становити O(n*log(n)).

Складність додавання елементу в незбалансоване дерево пошуку може сягати O(n), тому загальна швидкодія в таких випадках буде складати O(n^2).

Переваги даного алгоритму:

- однією з переваг сортування двійковим деревом є те що у ньому можна легко виконувати зміни, подібно як у зв’язаному списку;

- сортування за допомогою двійкового дерева пошуку має таку ж швидкодію, як в алгоритму швидкого сортування.

Недоліки:

- найгіршим випадком сортування є той коли вже відсортовано всі елементи масиву;

- в гіршому випадку швидкодія алгоритму сортування становить $O(n^2)$.

Сортування Шелла.

Алгоритм сортування який є узагальненням сортування включенням.

Алгоритм заснований на двох ідеях:

- сортування включенням щодо майже впорядкованих масивів є ефективним;

- сортування включенням є неефективним, оскільки кожен елемент переміщує за один раз лише на одну позицію.

Опираючись на дані ідеї у сортуванні Шелла виконуються декілька сортувань включенням, кожного разу порівнюючи і переставляючи елементи, які розташовані один від одного на різній відстані.

В результаті сортування Шелла є нестабільним.

Пірамідальне сортування.

Алгоритм сортування, який працює точно за $O(n * \log(n))$ операцій при сортуванні n елементів. Кількість використаної додаткової пам’яті від розміру масиву не залежить і складає $O(1)$.

Переваги алгоритму:

- потребує $O(1)$ додаткової пам'яті;

- час роботи в найгіршому випадку $O(n * \log(n))$.

Недоліки алгоритму:

- на практично відсортованих даних так само довго працює, як на хаотично розташованих даних;

- складний в реалізації;

- алгоритм є нестійким і потребує розширення ключа для забезпечення стійкості;

- потрібен прямий доступ, він не працює на зв'язаних списках та інших структурах пам'яті послідовного доступу;

- погано поєднується з кешуванням і файлом довантаження віртуальної пам’яті.

Пірамідальне сортування виконується на основі організації елементів у масиві подібно до двійкового дерева. Двійковим деревом називають структуру даних, яка є ієрархічною і у ній кожен елемент не має більше двох нащадків (рис. 1.1).

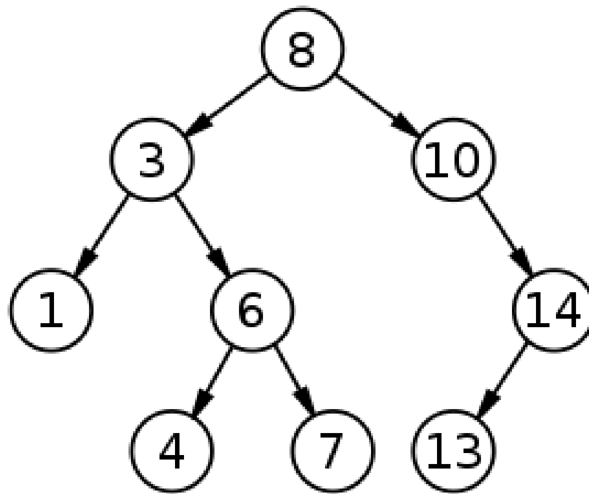

Рис. 1.1. Приклад двійкового дерева

Особливим видом бінарного дерева є піраміда, в якій значення кожного елемента є більшим ніж у його нащадків. Нащадки кожного з вузлів є не впорядковані, тому деколи правий нащадок може бути менше лівого, або ж – ні. Піраміда являє собою повне дерево, у якому новий рівень заповнюється тільки після того, як поточний рівень буде заповнений повністю, а всі вузли що знаходяться на одному рівні заповнюються послідовно (рис. 1.2).

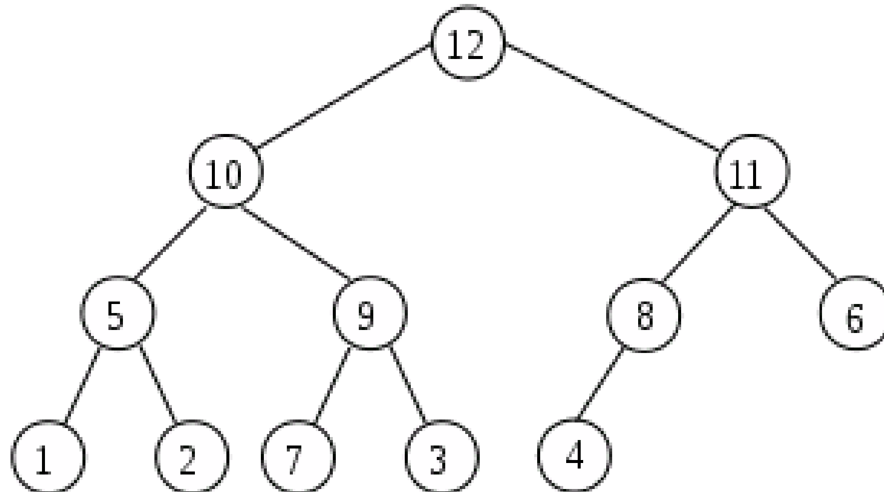

Рис. 1.2. Приклад піраміди

Сортування починається з того, що з усіх елементів послідовності будується піраміда. Максимальний елемент буде знаходитися у вершині дерева, оскільки всі нащадки обов'язково повинні бути меншими. Далі вершина піраміди переміщується в кінець послідовності, а піраміда без максимального елемента переформовується. Як результат, на її вершині виявляється максимальний серед елементів, що залишилися і він записується у відповідне місце. Процедура виконується доти, поки у піраміді не буде опрацьовано останній елемент і в результаті вона залишиться пустою.

Основне навантаження алгоритму несуть дві процедури: побудова піраміди на початку і її переформування на кожному кроці. Для побудови піраміди можна використати її властивість, що в кожного внутрішнього вузла повинно бути лише два безпосередніх нащадка, за винятком одного вузла на передостанньому рівні. Тому, починаючи з вершини можна пронумерувати вузли за правилом: вузол має номер “i”, а його нащадки отримують номери 2*i та 2*i+1. Таким чином, всі вузли піраміди отримають номери в діапазоні від 1 до N, що дає можливість зберігати піраміду у вигляді послідовності (рис. 1.3).

12	10	11	5	9	8	6	1	2	7	3	4
1	2	3	4	5	6	7	8	9	10	11	12

Рис. 1.3. Представлення піраміди у вигляді списку

Після видалення із вершини піраміди найбільшого її елемента кореневий вузол залишається вільним. Перебудовуючи піраміду в головний вузол із двох безпосередніх нащадків потрібно розмістити більший з них, а на його місце потрібно аналогічно визначити інший елемент із тих що залишилися. При тім структура піраміди має залишатися як можна ближчою до повного дерева. Для цього переформування піраміди починається із правого кінцевого елемента на нижньому рівні і він переноситься на його вершину. Далі виконується так звана процедура «просіювання вниз» над тією вершиною за такими кроками:

а) якщо в елемента відсутні нащадки, то кінець;

б) коли ж ні, то переставляються місцями значення даного елемента і максимального з-посеред його безпосередніх нащадків;

в) далі йде перехід до нащадка;

г) далі виконання процедури переходить до нащадка, який змінився і застосовується до нього ця ж процедура починаючи із кроку а).

Цей алгоритм дій можна використовувати не тільки до перебудови, а також для побудови початкового вигляду піраміди, оскільки для певного вільного вузла можна вважати нащадками будь-які два елементи. Із другою половиною не потрібно нічого робити тому, що всі елементи є “листами”. Розпочинаючи із переліку листових елементів, їх можна парами об’єднувати до тих пір доки не сформується вся піраміда. Індекс першого кореня, що буде доданий до послідовності з N елементів визначатиметься виразом N/2-1. В підсумку побудова початкової піраміди відбуватиметься за алгоритмом «просіювання вниз», який буде застосовано до кожного елементу від 0 до N/2-1.

1.3 Алгоритмічна складність

Асимптотична складність обчислювальних алгоритмів – це поняття із теорії складності обчислень, яке визначає оцінку ресурсів необхідних для виконання алгоритму.

Алгоритми мають два типи складності часову і просторову[24].

Складність алгоритмів поділяють на два типи:

- просторову складність – це пам’ять яку потрібно виділити для роботи алгоритму;
- часову складність або по іншому швидкодію – це час роботи алгоритму відносно кількості вхідних даних.

Важливою складовою алгоритмів є ЧС алгоритму, або по іншому швидкодія.

Розроблена велика кількість критеріїв для оцінки алгоритмів. В основному береться до уваги швидкість росту часу виконання і розміру пам’яті в залежності від обсягу даних, які потрібні для розв’язання задачі.

Як відомо класи алгоритмів суттєво відрізняються: великий розмір даних для задач які часто зустрічаються на практиці, сучасні комп’ютери можуть виконувати поліноміальні алгоритми, а вже експоненціальні алгоритми як правило не можуть бути виконані за розумний час навіть для невеликого розміру вхідних даних в прикладних задачах. Вважається що для розв’язку задач в яких при розв’язанні необхідне застосування експоненціальних алгоритмів потрібний повний перебір всіх можливих варіантів, а це є неможливим на практиці, і, відповідно, їх розв’язання відбувається іншими шляхами.

Для прикладу, коли для знаходження оптимального розв’язку певної задачі існує експоненціальний алгоритм, то в дійсності використовуються інші, які якомога ефективніші поліноміальні алгоритми, щоб знайти наближений розв’язок до найоптимальнішого. Але для цього потрібно повністю розуміти поставлене

завдання і мати достатню підготовку, щоб побудувати розв'язок, в якому буде застосовуватися вирішення задачі поліноміальним алгоритмом.

На рис. 1.1 показано графіки основних асимптотичних складностей алгоритмів.

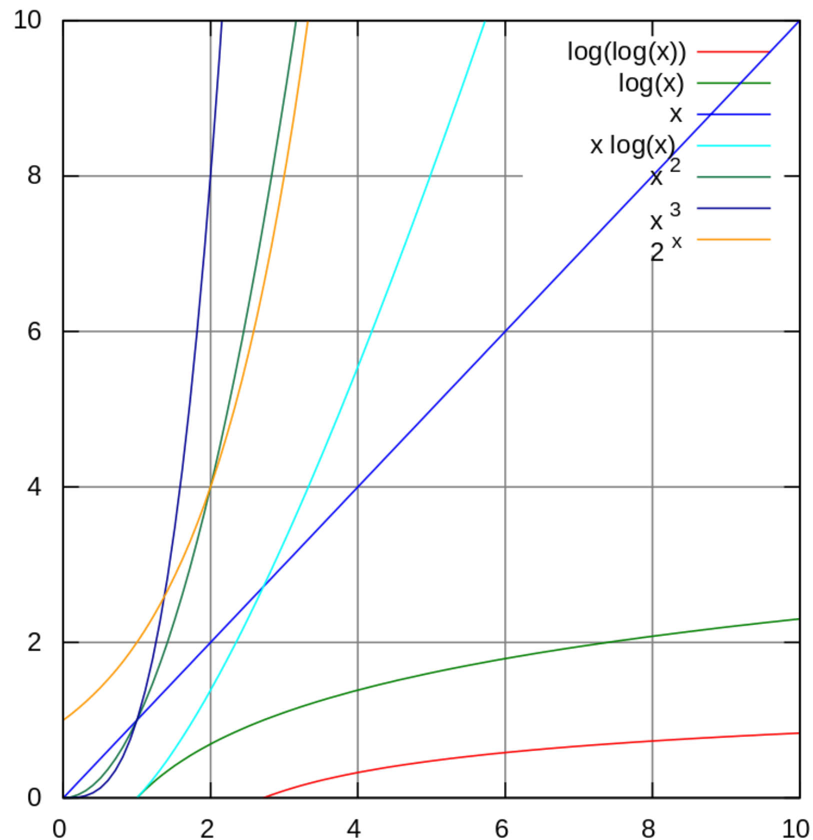

Рис. 1.1. Графіки функцій найпоширеніших асимптотичних складностей

Оцінку складності алгоритму записують нотацією великого "O", або ж інша назва нотація Ландау.

Перелік основних типів асимптотичних складностей алгоритмів і основні приклади алгоритмів із даною складністю:

- $O(1)$ – Стали час роботи. Алгоритм працює не залежно від розміру вхідних даних. Час пошуку у хеш-таблиці.

- $O(\log(\log(n)))$ – Сублогарифмічне зростання. Час роботи алгоритму зростає надзвичайно повільно. Інтерполяційний пошук серед елементів масиву.

- $O(\log(n))$ – Логарифмічне зростання. Зростання кількості вхідних даних у два рази збільшує час виконання алгоритму на певну сталу величину. Обчислення процесором піднесення числа до певного степеня і двійковий пошук елементів у масиві має таку складність.
- $O(n)$ – Лінійне зростання. Час зростає пропорційно розміру задачі. Додавання і віднімання чисел процесором в залежності від кількості цифр; лінійний пошук елементів в масиві.
- $O(n * \log(n))$ – Лінеаритмічне зростання. Збільшення розміру задачі у два рази збільшить час виконання дещо більше ніж у два рази. Двійкове сортування елементів, швидке сортування і деякі інші алгоритми сортування мають таку середню швидкодію.
- $O(n^2)$ – квадратичне зростання. Подвоєння розміру задачі збільшує потрібний час в чотири рази. Прості алгоритми сортування даних.
- $O(n^3)$ – кубічне зростання. Подвоєння розміру вхідних даних алгоритму збільшує у вісім разів потрібний час для його роботи. Множення матриць простим алгоритмом.
- $O(c^n)$ – експоненціальне зростання. При зростанні розміру задачі на одиницю необхідний час зростає у кількість разів визначених константою “c”. Як правило алгоритми із такою складністю є не практичними і задачі які вирішуються такими алгоритмами відносять до класу складності NP.
- $O(n!)$ – факторіальне зростання. Збільшення розміру задачі на одиницю збільшить необхідний час у n+1 разів. Цей клас має найвищу складність серед даного переліку. Вирішення задачі комівояжера виконуючи повний перебір усіх можливих варіантів розв’язку.

1.4 Застосування алгоритмів сортування даних

Алгоритми сортування даних є важливою складовою як інших алгоритмів так і в самостійному застосуванні, оскільки для швидкого пошуку інформації у

списках важливо щоб він був відсортований, або ж була побудована певна структура даних для швидкого пошуку, наприклад двійкове дерево пошуку.

Випадків застосування алгоритмів сортування даних можна чимало перелічити, ось деякі з них:

- Інтернет – технологія, яка дозволила людям із усього світу мати доступ до великих обсягів інформації різного роду. Із використанням правильно обраних алгоритмів інтернет-сайти можуть опрацьовувати і керувати величезними обсягами даних. Однією із важливих задач є швидкий пошук для знаходження сторінок на яких розміщена потрібна інформація і видача результатів у зручному представленні[6].
- Інтерфейси комп’ютерних програм і додатків смартфонів скрізь використовують пошук і видачу результатів.
- Програмні алгоритми використовують у своїй роботі для ефективного керування пам’яттю і опрацювання великих обсягів даних.
- Офісні програм які є дуже важливим для документообігу використовують сортування даних у свої роботі для правильної подачі даних.
- Черги з пріоритетами, які сортуються за певним критерієм.
- Перетин множин ефективно здійснюється, коли вони відсортовані.

Системи управління базами даних (СУБД), які у свою чергу використовуються в серверах, для зберігання і аналізу великих об’ємів даних. Для таких систем важлива швидкодія і невелике покращення роботи таких систем може бути значним на великих масивах даних.

Велике значення має масштабованість алгоритмів сортування, можливість їх застосування для обробки великих даних, придатність для впорядкування складних інформаційних структур.

Збільшення впливу сфери комп’ютерних систем у світі характеризується розвитком галузей, в яких використовується сортування даних у паралельному режимі в реальному часі за допомогою відповідних програмно-апаратних засобів.

Отже сортування є невід'ємною частиною роботи із різними видами інформації, в якій наявна класифікація або аналіз. Можна сказати, що ефективність при обробці даних комп'ютерної системи значно залежить від ЧС алгоритмів і також обсягу пам'яті потрібного для сортування. Важливим також буде звертати увагу на масштабованість алгоритмів сортування і придатність до обробки великих даних, можливість зробити впорядкованими складні структури даних. Варто згадати для прикладу те що завданням інформаційно-пошукових систем є не стільки знайти відповідний документ з посеред доступних сторінок, як ефективно і оптимально відсортувати ці веб-сторінки, щоб знайти серед них відповідні запитам і потребам певного користувача.

Оскільки зростає кількість комп'ютерних систем і сфер їх застосування, то розвивається галузь де використовуються способи сортування даних апаратними засобами паралельно в режимі реального часу.

Паралельне сортування використовують у завданнях аналізу великої кількості експериментальних даних, графічної інформації та розв'язку систем лінійних рівнянь[4].

1.5 Висновки до розділу 1

У розділі було розглянуто найпоширеніші алгоритми сортування даних.

Виконано аналіз принципів роботи даних алгоритмів.

Проаналізовано типи асимптотичної складності, які існують для оцінки швидкодії роботи алгоритмів на комп'ютерних системах.

Описано області застосування даного алгоритму, які відповідно визначають актуальність даної теми.

РОЗДІЛ 2
МАТЕМАТИЧНІ МЕТОДИ АЛГОРИТМУ ПОБІТОВОГО СОРТУВАННЯ ДАНИХ

2.1 Опис покращення алгоритмів сортування даних

Серед великої кількості алгоритмів сортування які були розроблені до кінця XX століття можна побачити подібність алгоритмів один до одного. Серед них велика кількість є неефективною на сьогоднішній день[5].

Можна виділити три принципи роботи алгоритмів сортування:

- алгоритми що працюють на порівняннях і перестановках;

- для своєї роботи створюють спеціальну структуру даних, наприклад бінарне дерево пошуку, або подіну до цієї структуру;

- алгоритми не основані на порівняннях.

Алгоритми що основані тільки на порівняннях і перестановках можуть мати максимальну теоретичну швидкодію $O(n * \log(n))$. Як правило такі алгоритми мають випадки гіршої швидкодії яка становить $\Omega(n^2)$, а деякі алгоритми мають таку середню швидкодію [7].

Щодо алгоритмів із структурою, то у них також окрім середньої швидкодії присутня і гірша швидкодія у певних випадках.

Алгоритми не основані на порівняннях або потребують спеціального обладнання, або ж швидкодія в них лінійна у певних межах, які визначені коефіцієнтом k, що є обов’язково більшим за log(N), тобто логарифму від кількості елементів.

Основні алгоритми, до яких подібний алгоритм що пропонується даній роботі, є такі алгоритми:

- сортування за розрядами;
- сортування двійковим деревом;
- пірамідальне сортування.

Сортування за розрядами.

Це алгоритм розробив Герман Голлеріт ще 1887 році для роботи табулятора із інформацією записаною на перфокартах.

Ефективний алгоритм цього сортування, для роботи на цифрових комп'ютерах, був розроблений у Массачусетському технологічному інституті Гарольдом Сьюардом в 1954 році.

Даний алгоритм спочатку був націлений на сортування елементів, які записуються у вигляді цілих чисел.

Оскільки комп'ютери записують будь-яку інформацію у вигляді чисел, алгоритм можна використовувати для сортування різного роду інформації, яку можна розділити на розряди, які можуть бути порівняні за значеннями. Таким чином можна легко сортувати не тільки переліки чисел, але й рядки, які є насправді представлені у пам'яті набором байт, і загалом будь-яку інформацію, яка записана у даному вигляді.

Алгоритм сортування виконується порозрядно. Розпочинає опрацювання чисел із найменшої значущої цифри. Відбувається обхід по елементам і вони сортуються за даної цифрою. Далі таким чином сортуються значення за наступними цифрами, зберігаючи порядок який був встановлений при сортуванні попередніх розрядів.

Приклад роботи алгоритму сортування за розрядами можна побачити на рис.2.1.

Рис. 2.1. Сортування за розрядами деякого набору чисел

Кожен крок вимагає одного проходу по даним, тому що можливе поміщення кожного елементу у свою комірку не порівнюючи із іншими елементами.

Різні реалізації цього сортування виділяють простір для комірок підраховуванням кількості ключів, які належать коміркам перед тим як перемістити ключові значення у певні комірки. Кількість повторень певної цифри зберігається у масиві.

Завжди можна визначити наперед межі комірок використовуючи підрахунок, але деякі реалізації оптимізовані під використання динамічного виділення пам’яті.

Складність цього сортування визначається кількістю ключів і довжиною ключа. Оскільки кількість всіх проходів буде така ж як число розрядів у значенні ключа, а в комп’ютерних системах кількість розрядів є визначеною вже довгий час, звідси кількість проходів завжди буде однаковою, тільки лінійно зростатиме час разом із кількістю елементів, що й визначає складність даного алгоритму [12].

Алгоритм двійкового сортування при опрацюванні елементів створює двійкове дерево і використовує його для оптимальної роботи.

Дерево двійкового пошуку використовує центрований обхід (рис.2.2).

Центрований обхід має наступний алгоритм виконання:

- перевіряється чи поточний вузол не є пустим;

- відбувається обхід лівого піддерева використовуючи рекурсію, тобто функція центрованого обходу застосовується до піддерева;

- відбувається відображення даних поточного вузла;

- обходиться правого піддерева відбувається за допомогою рекурсії подібно до лівої частини.

Центрований обхід обійде дерево у наступному порядку: A, B, C, D, E, F, G, H, I.

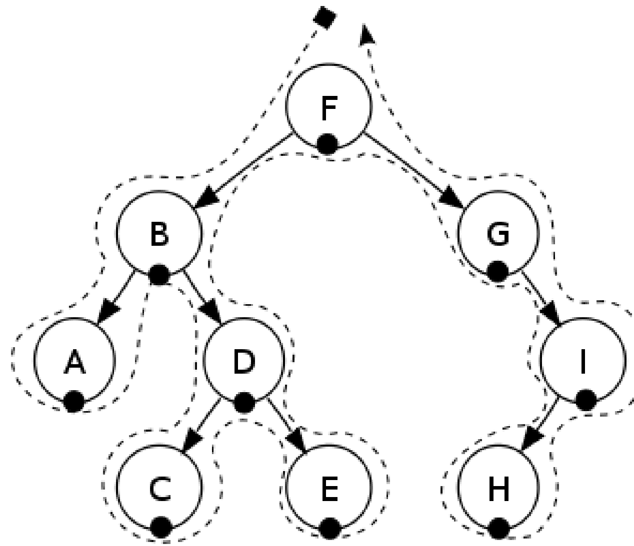

Рис. 2.2. Центрований обхід дерева

Пірамідальне сортування використовує купи у своєму алгоритмі [14-15].

Двійковою купою називається спеціальна структура даних, яка є масивом і її можна розглядати як повне двійкове дерево[8].

Для куп є три прості операції, які використовуються для обходу по вузлам, вони представлення на рис.2.3 трьома процедурами у вигляді псевдокоду.

PARENT(i)
    return $\lfloor i/2 \rfloor$

LEFT(i)
    return $2i$

RIGHT(i)
    return $2i + 1$

Рис. 2.3. Псевдокод процедур обходу купи

Пірамідальний алгоритм сортування даних працює створюючи двійкове дерево, але із тією різницею, що воно має властивості піраміди, звідки і назва даного алгоритму. Самі дерево структурно представлено на рис.2.4 із лівої сторони, його реальне представлення в пам’яті у вигляді масиву – справа.

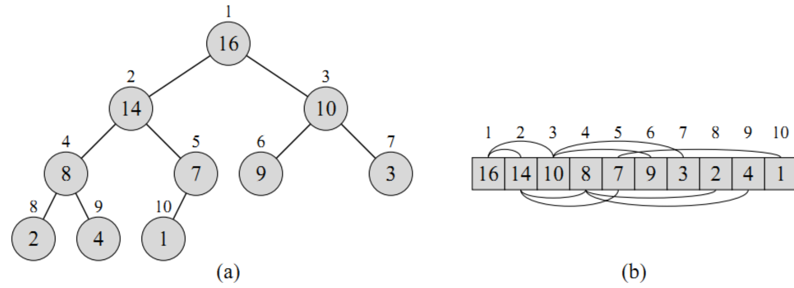

Рис. 2.4. Піраміда у вигляді дерева – зліва і у вигляді масиву – справа

На рис.2.5 можна побачити, зліва зверху, як масив чисел представлений у пам’яті. Нижче показано візуально яким чином покроково відбувається створення цієї піраміди.

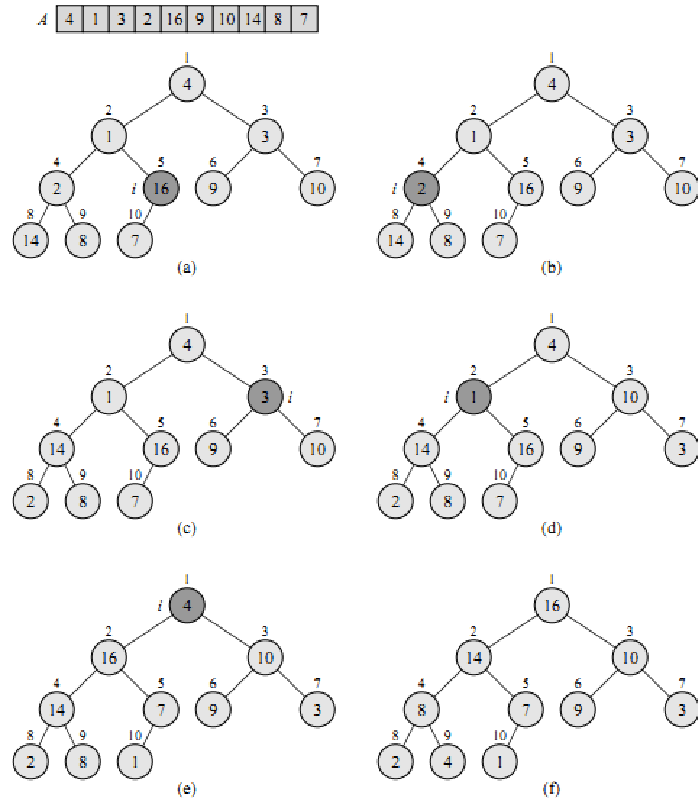

Рис. 2.5 Створення піраміди

Власне сортування пірамідою починається із кореневого елемента, який видаляється із структури і розміщується в кінець масиву.

Далі відбувається перестановка і наступний елемент більший елемент із дочірніх стає кореневим.

Для нього виконується така ж процедура переміщення в кінець відновлення піраміди, як і для першого вузла. Виконання продовжується доки у піраміді не залишиться жодного елемента. Всі елементи відсортовано і отримано їх список, як показано на рис.2.6

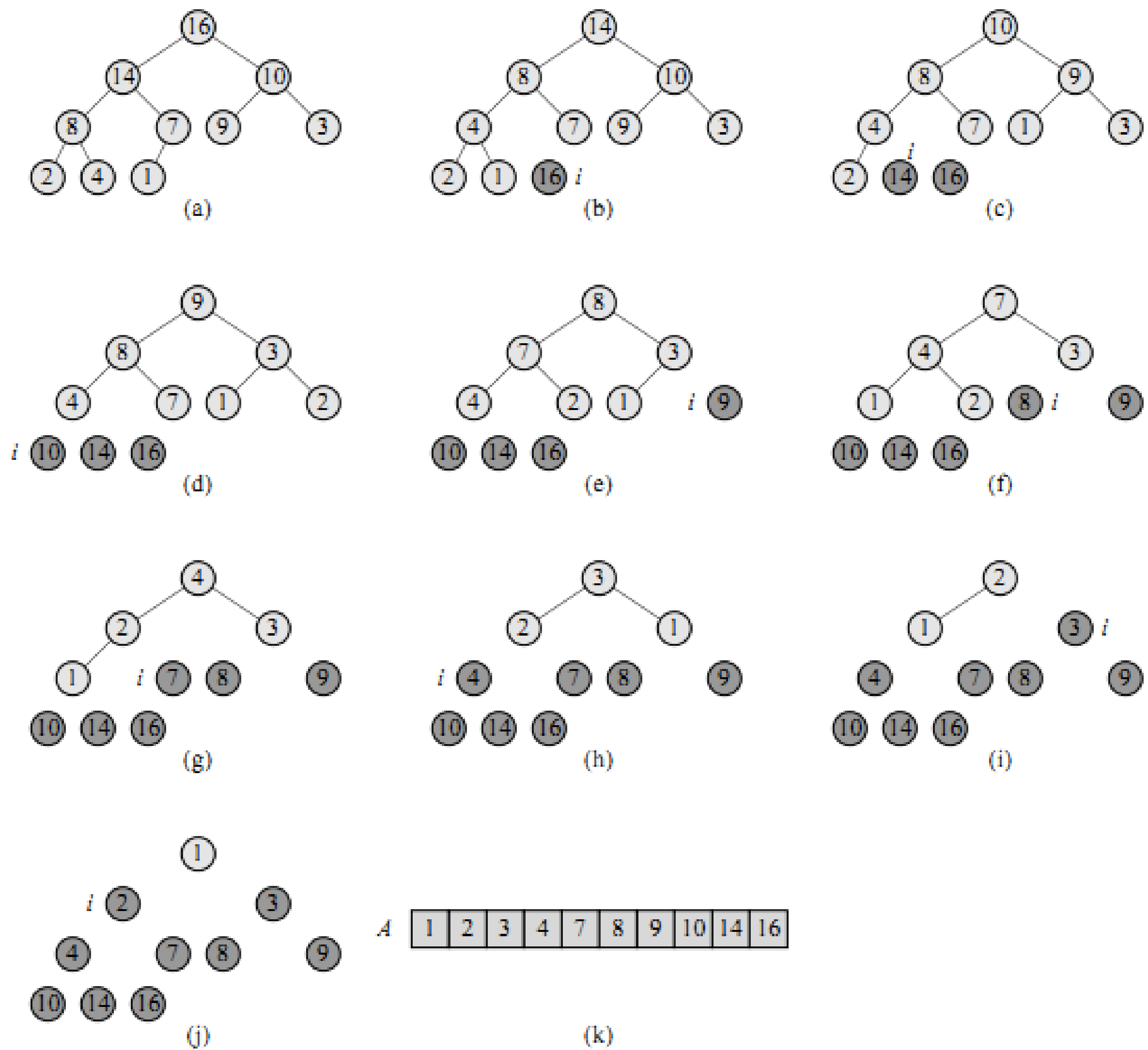

Рис. 2.6. Покрокове пірамідальне сортування

Для підтримки властивостей піраміди при видаленні вузла, виконується спеціальна процедура. Відбувається перехід по елементам і їх перестановки місцями так, щоб відновити властивості піраміди (рис.2.7).

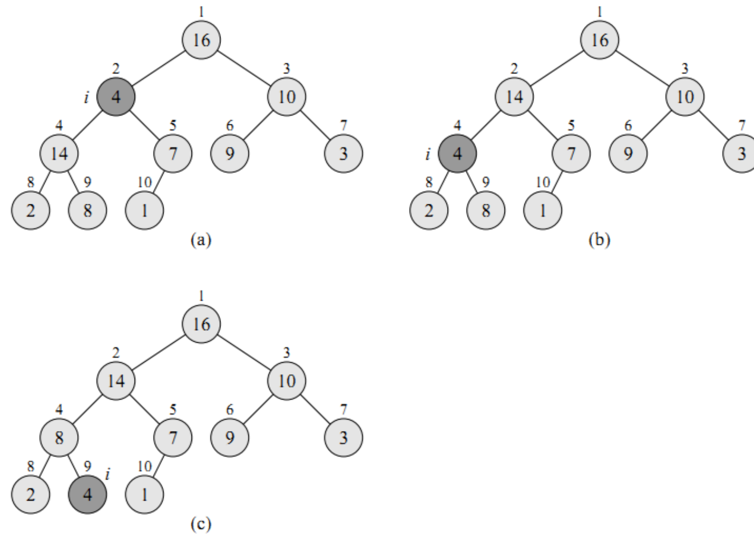

Рис. 2.7. Підтримка властивостей піраміди

Варто зазначати що найбільш подібним алгоритмом до алгоритму, який розробляється у цій роботі, є алгоритм сортування на основі двійкового дерева пошуку, але він має свої відмінності такі як те що кількість всіх порівнянь не більше N, в той час як у бінарного дерева для пошуку по ньому здійснюється в середньому $\log(N)$ порівнянь і ще через його балансування кількість порівнянь елементів теж збільшується і там виконуються постійні перестановки елементів по дереву чого немає у даного алгоритму[23].

Пірамідальне сортування використовує подібне двійкове дерево до двійкового дерева пошуку і відрізняється від дерева яке створюється побітовим методом сортування.

Існує сортування із подібною назвою “Сортування за розрядами” і має подібну ідею сортування, але насправді механізм сортування відрізняється. На відміну від даного алгоритму в побітового сортування виконується сортування починаючи із найстаршого розряду і створюється адресна структура, яка розширюються вглиб не одразу на всю кількість розрядів, а по мірі її заповнення. Порозрядне сортування насправді не є самостійним алгоритмом сортування, оскільки використовує допоміжний алгоритм сортування для кожного розряду, а

наведений алгоритм сортування у даній роботі не використовує допоміжних алгоритмів сортування.

Швидкодія даного алгоритму не залежить від того як пересортовані дані і є стабільною $O(n * \log(n))$. Сортування даних відбувається у два проходи. Перший прохід виконує зчитування елементу і запис його у дерево, а другий прохід зчитує із даного дерева масив даних у відсортованому порядку. Другий прохід по дереву є не обов’язковим якщо потрібен доступ до елементів, оскільки дане дерево забезпечує простий і швидкий механізм доступу до кожного елемента, якщо за ним закріплені якісь дані. Бінарний пошук по відсортованому масиву використовує log(n) переходів і порівнянь, у той час як пошук елемента по даному дереву використовує тільки переходи кількість яких може бути менша за log(n) на сталий коефіцієнт який залежить від кількості біт які зараз опрацьовуються на кожному переході.

Можна виділити ще таку характеристику алгоритмів як використання додаткової пам’яті, вони бувають з додатковою пам’яттю і без неї, тобто просторова складність $O(1)$.

Алгоритми які не використовують додаткової пам'яті базуються на порівняннях і перестановках елементів, оскільки іншим чином вони не можуть бути правильно опрацьовані.

Інший тип алгоритмів сортування використовує додаткову пам’ять для створення певної структури яка допомагає відсортувати дані, у деяких випадках навіть без їхнього порівняння.

Алгоритми які без додаткової пам’яті мусять переставляти елементи повністю не залежно від їхньої довжини, тому часто використовується для таких алгоритмів адресне сортування де є список адрес із посиланнями на елементи і перестановка відбувається саме цих адрес, а вони зазвичай коротші за самі текстові дані. Тому навіть такі алгоритми часто використовують $O(n)$ додаткової пам’яті, якою і є той адресний список. Із урахуванням цього можна зробити висновок, що використання $O(n)$ додаткової пам’яті є необхідним і її ефективне

використання є оптимальним, з точки зору покращення швидкості роботи алгоритму. Тому було вирішено створювати побітову адресну структуру, яка має складність по пам’яті $O(n)$, використання якої зменшує кількість порівнянь елементів між собою і відсутні перестановки елементів місцями. Звідси можна побачити що характеристики алгоритму були значно спрощенні. Щодо самої структури то вона задає переходи по парам бітів, де дві пари бітів формують чотири адреси переходів на наступні адреси для наступної пари бітів. Сама структура розширюється як в глиб, так і в шир по мірі її заповнення елементами. Це дозволяє оптимізувати розмір як структури, так і кількість виконаних кроків, оскільки не потрібні ланки для створення відсортованого масиву не будуть створені, а створені будуть лише до того місця у елементі де він відрізняється від усіх інших елементів. Кількість порівнянь елементів для високоентропійних даних буде менша в середньому у 4 рази за рахунок того що сама структура робить розмежування елементів і самі переходи заміняють порівняння елементів при чому кількість тих переходів складає логарифм від усієї кількості елементів. Коли використовується адресне сортування у алгоритмах на осонові порівняння, то у них кількість переходів по адресам повинна бути така ж як і кількість порівнянь елементів, а кількість порівнянь елементів у таких алгоритмах теоретично не може бути менша ніж $O(n * \log(n))$. Такі алгоритми використовують порівняння, пам’ять, переходи і перестановки, де основну роль у процедурі сортування відіграють порівняння. Переходи у пам’яті можна використати певним чином так щоб зменшити кількість порівнянь і не використовувати перестановки елементів. У такому випадку перестановки стають зайвою процедурою, а кількість порівнянь елементів зменшується.

2.2 Розробка алгоритму побітового сортування даних

Основні цілі, які потрібно досягнути при розробці алгоритму роботи сортування даних: простота алгоритму і висока швидкодія. Простота алгоритму є

не менш важливою адже алгоритм із тією ж асимптотичною складністю коли буде мати складну реалізацію, із точки зору виконання операцій для опрацювання одного елемента, то може виявитися не практичним відносно іншого алгоритму, який швидко опрацьовує елементи, через свою простоту, за час менший на один або два порядки. Це означатиме що для сортування тієї ж кількості елементів потрібно менші ресурси комп'ютера, а при тих самих ресурсах можна отримати виграш по кількості опрацьованих елементів.

На рисунку 2.8 зображена блок-схема методу побітового сортування даних.

Алгоритм виконує сортування даних в оперативній пам'яті. Виділяється пам'ять під список вхідних елементів і записується туди той перелік елементів. Далі алгоритм виділяє іншу пам'ять під створення структури для його роботи.

У алгоритмі є поточна адреса для опрацювання структури. Спочатку вона вказує на початок пам'яті де буде створюватися структура.

Алгоритм починає опрацювання із першого елементу і зчитує його перші два біти. Виконується обчислення нового значення поточної адреси. До неї додається довжина адреси помножена на зчитані біти. Наразі в сучасних комп'ютерах використовується 32-, або 64-розрядна архітектура, тому в залежності від вибраної архітектури довжина адреси складатиме чотири, або вісім байт відповідно. Множник зчитаних бітів буде від 0-3, оскільки два зчитаних біта можуть мати таких 4 значення у десятковій системі. Якщо вибрана 32 розрядна архітектура, а перші два біти будуть наприклад 10 у двійкові системі, то до поточної адреси опрацювання структури буде додано значення 4*2, тобто на 8 байт зміститься даний вказівник. Звідти зчитується число, яке є адресою крім випадку коли воно рівне нулю. У цьому випадку дана комірка пам'яті вважається вільна, і туди записується адреса поточного елементу, який опрацьовується із встановленим в одиничку старшим бітом цієї адреси. Даний біт позначає чи вказана адреса посилається на дані чи на наступну адресу, яка у свою чергу буде відповідати за наступні два біти даного елемента.

Якщо умова не виконалась і у тій комірці пам'яті на яку вказує поточна адреса не нульове значення то це значення є адресою. Якщо старший біт буде нуль, то дана адреса вказує на наступний блок адрес. Відбувається зчитування із поточного елементу наступних чотирьох біт і за новою сформованою адресою із цього значення та адреси наступного блоку зчитується значення яке опрацьовується описаним вище чином.

Коли ж старший біт зчитаної адреси буде одиничкою, то дана адреса вказує на один із попередніх опрацьованих елементів, якщо наразі вже опрацьовано декілька елементів.

Тоді як на блок-схемі показано виконується процедура розширення адресної структури. Це відбувається наступним чином: поточний елемент який опрацьовується порівнюється із елементом вказаним за зчитаною адресою і у вільній частині виділеної пам'яті формується новий блок із чотирьох адрес вказівка на який записується замість вказівки на елемент із яким проходить поточне порівняння. Якщо у цих двох елементів значення читаних бітів різне, то в новому блоці адрес записуються у відповідні комірки вказівки на ці два елементи із зміненими в одиничку старшими бітами, що позначає вказівку саме на елемент, а не на блок адрес. Коли ж біти двох елементів є однаковими то виділяється у вільній частині пам'яті місце під новий блок із чотирьох адрес. Його адреса записується у поточному блоці в комірку яка відповідає номеру тих однакових двох бітів елементів. Далі продовжується опрацювання описаним вище чином до тих пір поки не буде знайдено різні значення зчитаних пар бітів цих елементів, тоді запишуться їх адреси у вільні комірки і опрацювання перейде до наступного елементу, а поточна адреса опрацювання даних у структурі змінить значення на вказівку початку виділеної пам'яті під дану структури.

Коли всі елементи будуть опрацьовані за даним алгоритмом, то відбудеться прохід по утвореному дереву із адрес де елементи розташовані вкінці гілок, тобто на листках. Всі елементи будуть зчитані із нього вже у відсортованому порядку.

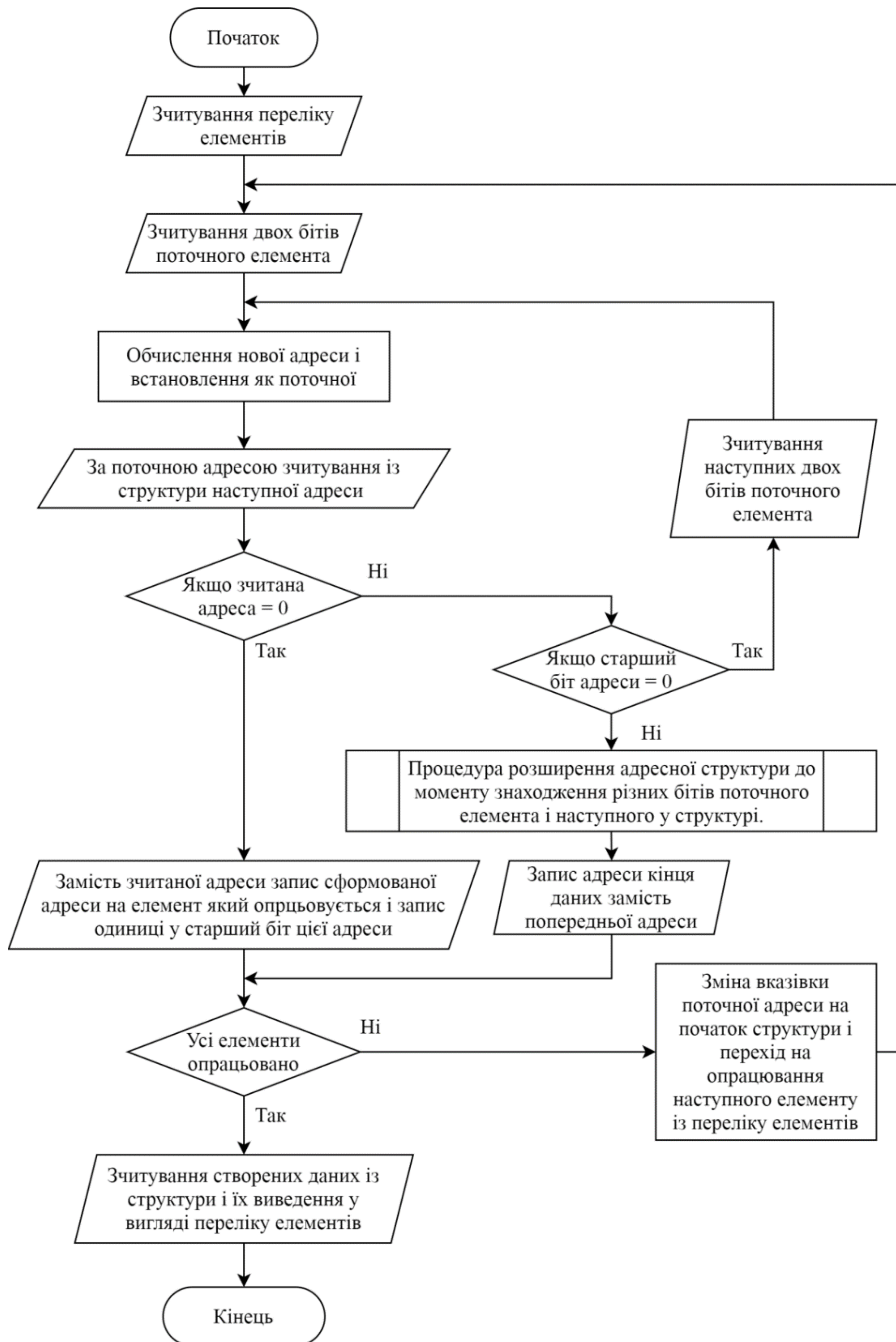

Рис. 2.8. Блок-схема алгоритму побітового сортування даних

Даний алгоритм виконує один прохід по елементам із яких створюється оптимальна структура даних, а другий прохід виконує зчитування із структури.

Таким чином, алгоритм не використовує повторних циклів опрацювання вхідного не відсортованого масиву даних. Другий цикл алгоритму можна не

використовувати, замість того залишити структуру, і по ній звертатися, до елементів.

Алгоритм має квазілінійну часову складність – це пов’язано із структурою дерева яке створюється, його обхід має таку складність по часу.

Переваги алгоритму:

- виконує сортування внутрішньо, але придатний і до зовнішнього сортування, звісно у такому випадку його робота буде дещо повільніша;
- є стабільним, оскільки порядок елементів не змінюється через відсутність будь-яких перестановок місцями елементів що сортуються;
- використовує мінімум операцій порівняння;
- не використовує операцій перестановки;
- має високу теоретичну і фактичну швидкодію, тому є практичним для використання.

Недоліки алгоритму:

- вимагає додаткової пам’яті;
- майже відсортовані дані сортує так само як хаотичні.

Часто алгоритми, які опрацьовують велику кількість елементів, у комп’ютерних системах реалізовують низькорівневою мовою програмування, адже потрібна висока швидкодія із найменшими затратами обчислювальних ресурсів. Тому для реалізації даного алгоритму пропонується використовувати мову програмування Assembler. Окрім окреслених попередніх умов, для даного алгоритму це особливо актуально у зв’язку із його принципами роботи.

Структура дерева, яке будується для роботи цього алгоритму сортування даних.

Спочатку дано для прикладу певний перелік елементів, який наведений у табл.2.1.

Таблиця 2.1

Перелік елементів

№ з/п	Елементи у двійковому представленні
1.	101111
2.	111010
3.	001000
4.	011000
5.	011101
6.	010010
7.	010110
8.	011001
9.	011010
10.	011011

Біти у елементах нехай нумеруються зліва на право від 0 до 5.

По першим двом бітам “10” розміщується перший елемент у дереві, яке будується.

Другий, третій і четвертий елементи подібно розміщуються у перших гілках дерева, оскільки вони свобідні (рис.2.9).

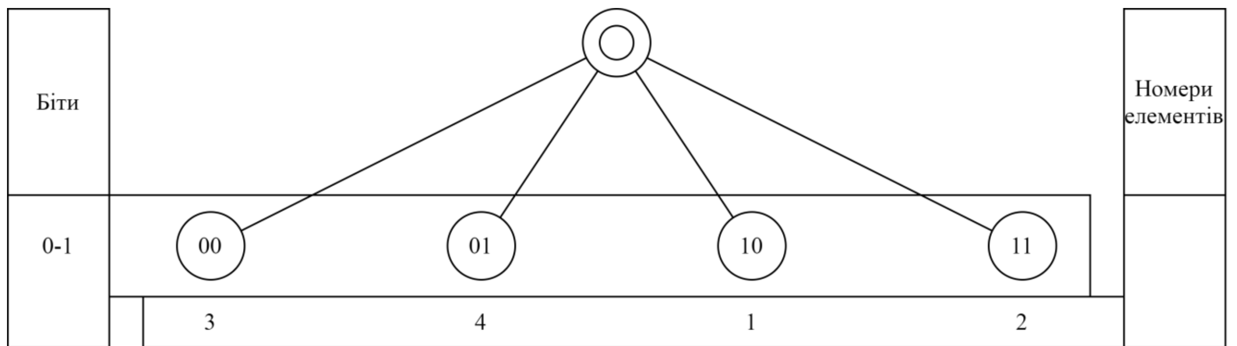

Рис. 2.9. Дерево після опрацювання перших чотирьох елементів

Далі перші два біти п’ятого елемента “01” є такі ж як і в четвертого елемента, тому створюється нове розгалуження, для якого беруться наступні два біти у четвертого елемента “10”, а у п'ятого “11”. Оскільки біти були різними, то елементи розміщено у різних гілках (рис.2.10). Опрацювання продовжується подібним чином для наступних елементів.

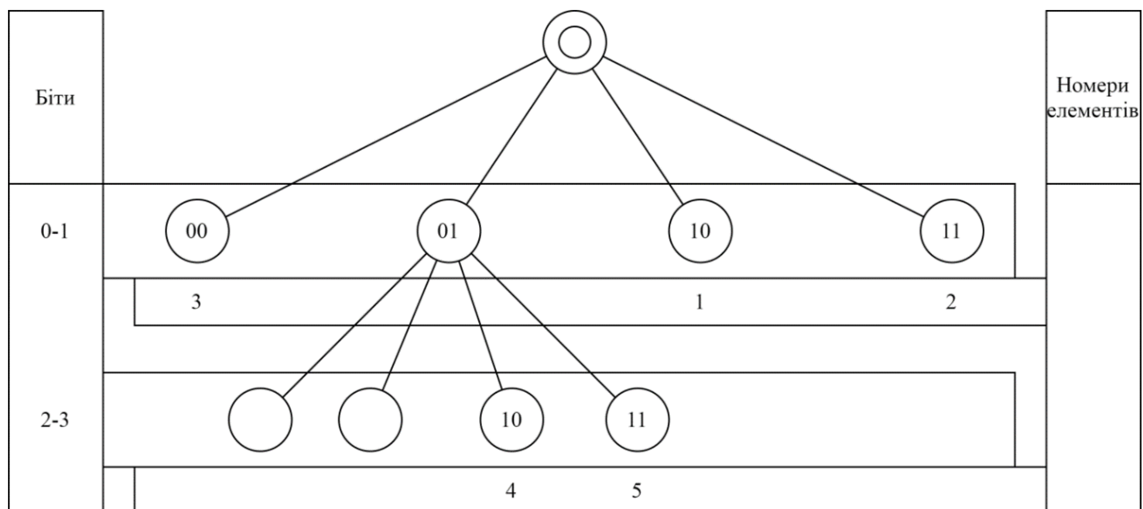

Рис. 2.10. Дерево після добавлення п'ятого елемента

Шостий і сьомий елементи добавляться у дві свобідні гілки дерева, а наступні елементи створять нове розгалуження і так сам як попередні елементи заповнять листки даного дерева (рис.2.11).

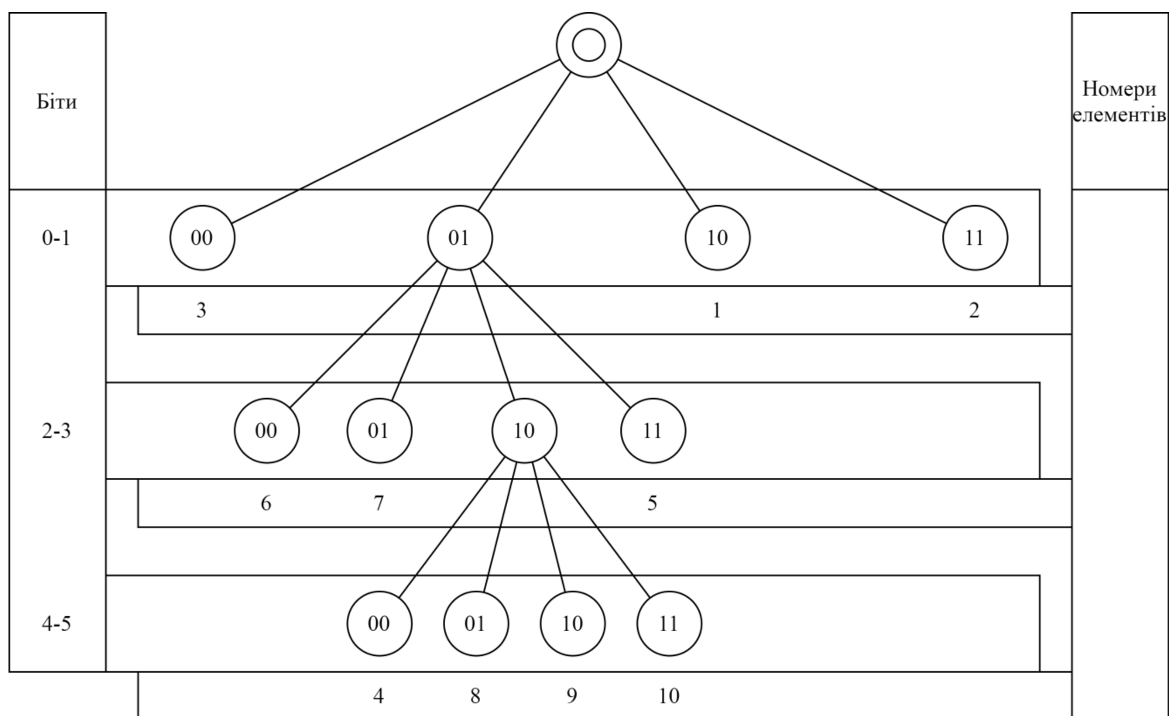

Рис. 2.11. Дерево після опрацювання всіх десяти елементів

Вкінці відбувається прохід по листкам дерева і зчитується елементи у відсортованому порядку (табл.2.2).

Як можна побачити із схеми дерева для деяких елементів не опрацьовано всіх бітів, оскільки для алгоритму це не завжди є необхідним і це означає що він виконується дещо швидше, але результуюча асимптотична складність із зростанням розміру задачі прямує до $O(\mathrm{n} * \log(\mathrm{n}))$.

Таблиця 2.2 (зробити картинкою)

Перелік відсортованих елементів

№ з/п	Елементи у двійковому представленні
1.	001000
2.	010010
3.	010110

4.	011000
5.	011001
6.	011010
7.	011011
8.	011101
9.	101111
10.	111010

Дане дерево яке будується доволі схоже на бінарне пошуку, яке використовується у сортуванні двійковим деревом.

Відмінність даного дерева полягає у тому, що елементи розміщенні завжди на листках, на відміну від бінарного дерева, і кожен елемент при записі у дерево порівнюється максимум із одним елементом, тому кількість всіх порівнянь не більше $O(n)$, але кількість переходів по дереву $O(n * \log(\text{n}))$, звідси алгоритмічна складність даного алгоритму $O(n * \log(\text{n}))$.

Кількість порівнянь і переходів у двійкового дерева пошуку однакова і складає $O(n * \log(\text{n}))$.

Даний алгоритм має більш простий і швидший алгоритм дій при сортуванні, а тому його реалізація повинна працювати дещо швидше за інші алгоритми сортування із такою ж алгоритмічною складністю.

Дана структура ефективна як на малому наборі даних, так на великому і також рівномірно зростає, бо її розмір лінійно зростає на високоентропійних даних і звідси її просторова складність $O(n)$ - це одна із ключових переваг даної структури.

На основі тієї структури і працює алгоритм сортування даних із тією різницею що дана структура формується послідовно у пам’яті. Гілками є адреси

переходів на наступні блоки, а листками – адреси на місце у пам'яті, де розміщенні елементи.

2.3 Висновки до розділу 2

У даному розділі виконано розгляд моделей та принципів сортування даних. Розглянуто структури даних, які використовуються для виконання поставленої задачі.

Визначено швидкі моделі, які оптимізовано і на їхній основі розроблено новий алгоритм сортування даних.

Описано його характеристики та основні подібності і відмінності, від найбільш подібних до нього алгоритмів.

Створено блок-схему для написання програмної реалізації даного алгоритму.

Розглянуто структуру даних яку використовує для своєї роботи цей метод побітового сортування даних.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДУ ПОБІТОВОГО СОРТУВАННЯ ДАНИХ

3.1 Опис коду програмного забезпечення

Реалізацію даного методу побітового сортування даних було вирішено написати низькорівневою мовою програмування Assembler [16]. Це важливо було для отримання високої швидкодії даного алгоритму. Даною мовою програмування пишуться програми команди яких є так званими машинними командами і виконуються одразу самим процесором.

Плюси даної мови програмування:

- можливість написання оптимального коду;
- невеликий розмір вихідної програми;
- безпосередній доступ до периферійних пристроїв;
- максимальна можливість підлаштування коду для певної архітектури, чи то операційної системи.

Серед мінусів можна виділити наступні пункти:

- великий обсяг коду;
- складний у написанні, налагодженні, погана читабельність коду;
- код написаний для певної архітектури не буде працювати з іншою.

Виконання коду програми сортування даних розпочинається із створення діалогового вікна (рис.3.1).

```
    .code
; ##############################################################

entry_point proc
    mov hInstance, rv(GetModuleHandle,0)
    mov hIcon, rv(LoadImage,hInstance,10,IMAGE_ICON,32,32,LR_DEFAULTCOLOR)
    invoke DialogBoxParam,hInstance,100,0,ADDR main,hIcon
    invoke ExitProcess,0
    ret
entry_point endp

; ##############################################################
```

Рис. 3.1. Створення вікна програми

Функцією “GetModuleHandle” [17] повертається дескриптор модуля програми. Він передається функції завантаження картинки і функції завантаження вікна.

Функція завантаження вікна запускає процедуру “main” у якій виконуватиметься основний код опрацювання вікна.

Коли діалогове вікно завершиться, то виконання коду повернеться із процедури і перейде до виконання системної функції “ExitProcess” для завершення програми.

Код процедури “main” розпочинається із визначення змінних, які для роботи даної процедури (рис.3.2).

```
main proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,lParam:QWORD

    LOCAL patn  :QWORD
    LOCAL pMem  :QWORD
    LOCAL fname :QWORD
    LOCAL wmsg  :QWORD
    LOCAL str1  :QWORD
    LOCAL buffer[320]:BYTE

    LOCAL hFile :QWORD
    ;LOCAL cloc  :QWORD
    LOCAL flen  :QWORD
    local flenSourceFile    :qword
    local hMemSourceFile    :qword
    local LenBufSourceFile  :qword
    local hMemDistinationFile   :qword
    LOCAL TimerInSecond[2]      :QWORD
    LOCAL TimerFrequency        :QWORD

    LOCAL hFont :QWORD
    LOCAL nmh   :NMHDR
```

Рис. 3.2. Визначення змінних

Виконання коду розпочинається із опрацювання номерів подій змінної “uMsg” (рис.3.3).

Перший випадок “ WM_INITDIALOG ” опрацьовується при запуску вікна і виконує власну ініціалізацію та усіх своїх примітивів:

- встановлюється іконки;
- встановлення назви вікна;
- ініціалізація гіперпосилань;
- встановлення шрифтів гіперпосилань;
- ініціалізація поля для пошуку елемента.

```
.switch uMsg
  .case WM_INITDIALOG

    invoke SendMessage,hWin,WM_SETICON,1,lParam
    invoke SendMessage,rv(GetDlgItem,hWin,102), \
           STM_SETIMAGE,IMAGE_ICON,lParam
    invoke SetWindowText,hWin,"Sort"

    mov hFont, GetFontHandle("Arial",16,600)

    mov hLink1, rv(GetDlgItem,hWin,103)
    invoke SendMessage,hLink1,WM_SETFONT,hFont,TRUE

    mov hLink2, rv(GetDlgItem,hWin,104)
    invoke SendMessage,hLink2,WM_SETFONT,hFont,TRUE

    mov hLink3, rv(GetDlgItem,hWin,105)
    invoke SendMessage,hLink3,WM_SETFONT,hFont,TRUE

    mov hEdit1, rv(GetDlgItem,hWin,107)

    .return TRUE
```

Рис. 3.3. Оформлення віджетів вікна

У даному випадку йде опрацювання кнопок вікна інформація про кнопку, яка була натиснута, міститься у змінній “wParam”. Визначається натискання однієї із трьох кнопок вікна (рис.3.4).

```
.case WM_COMMAND
  .switch wParam
    .case 106      ;IDC_BTN1   Сортувати

      ;             mov patn, CTXT("всі файли",0,"*.*",0)
    comment*-----------------------------------------------------
      mov fname, OpenFileDlg(hWin,hInstance,"Вибрати файл",patn)
      cmp BYTE PTR [rax], 0
      jne @F
          •         •         •

   .case 101      ;IDC_BTN2   Шукати
   comment*---
       mov      str1, ptr$(buffer)
       mov wmsg, cat$(str1,str$(flen)," байт записано на диск", c
       invoke MessageBox, hWin, ptr$(buffer),reparg("Результат за
          •         •         •

   .case 108      ;IDC_BTN3 Write the sorted array
       include Write the sorted array.asm

   .endsw
```

Рис. 3.4. Опрацювання кнопок вікна

Перша кнопка відповідає за сортування даних, друга кнопка – за пошук елемента, а третя – виконує код запису відсортованого переліку елементів. Код третьої кнопки винесений у файл з назвою “ Write the sorted array.asm ”. В подальшому отриманий код події, від операційної системи, порівнюється і знаходиться випадок якому він відповідає. Код того випадку буде опрацьовано і виконання перейде на кінець даного оператора “.endsw”.

Основний алгоритм програми побітового сортування даних розташований в опрацюванні кнопки “Сортувати”.

З файлу зчитується перелік елементів, який потрібно відсортувати. Виділяється пам’ять під структуру і підготовлюються регістри для роботи із ними, а саме зберігаються у стек, для того щоб вкінці зчитати дані з них (рис.3.5).

```
    invoke exist proc lpszFileName:QWORD
invoke load_file, ADDR SourceFileName
mov hMemSourceFile, rax
mov LenBufSourceFile, rcx

shl rcx, 3
mov flenSourceFile, rcx
mov hMemDistinationFile, alloc(flenSourceFile)


lea     rcx, TimerInSecond
call    QueryPerformanceCounter

push    rbx
push    rsi
push    rdi
push    r12
;push   r13
;push   r14
;push   r15
push    rbp
mov SaveRSP, rsp
```

Рис. 3.5. Читання файлу і збереження регістрів

У підготовлені регістри записуються не обхідні адреси пам’яті і розпочинається виконання основних циклів зчитування елементів і запис їх у вигляді структури в пам’ять відведену для неї.

Починаючи із мітки “AddresRecorded” записується адреса даних і дані елемента по тій адресі. Із мітки “CreatingMetadata” розпочинається цикл створення структури адрес і переходів по ним. Далі зчитуються біти елементів і перевіряються позиції у структурі чи вони є вільними, якщо так, то виконується код по мітці “AddresRecorded”. Коли ж місце у структурі вже зайнято певним елементом, то виконується код по мітці “CompareElements”, який порівнює елементи і розширює структуру до тих пір поки не буде знайде різні біти елементів, це означатиме що тепер для них існують окремі місця у даному дереві переходів.

Вкінці вміст регістрів відновлюється із стеку і виводиться повідомлення із часом роботи і об’єм структури у байтах, який вона займає у пам’яті.

3.2 Результати тестування

Для тестування роботи програми, яка сортує дані, було написано додаткову програму, якою можна згенерувати перелік із випадкових елементів.

Інтерфейс програми показано на рис.3.6.

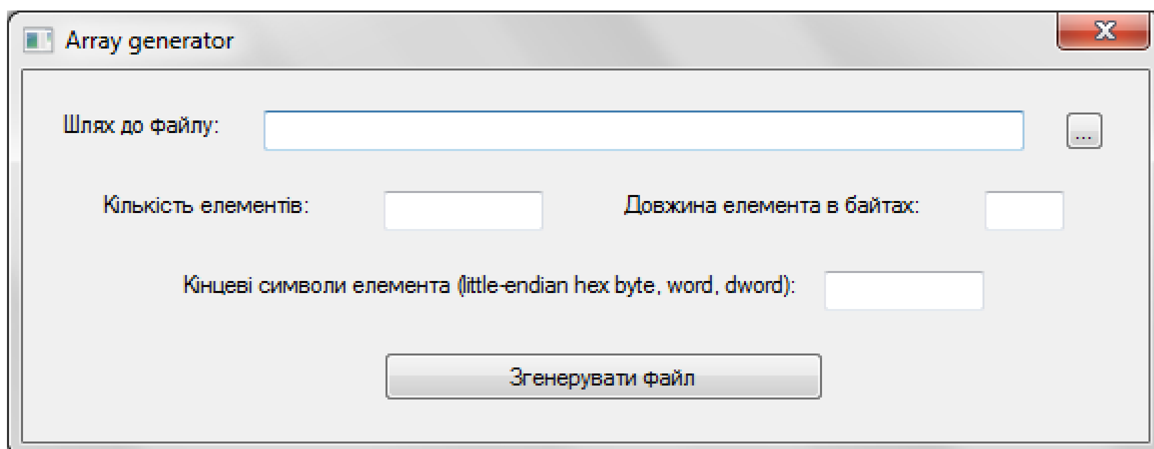

Рис. 3.6. Інтерфейс програми генератора

Програма розроблена з простим інтерфейсом, в стилі мінімалізму, виключно для того щоб можна було виконати тестування програми сортування даних.

Спочатку вказується шлях до файлу “SourceFile.txt” у поле із назвою “Шлях до файлу”, як показано на рис.3.7.

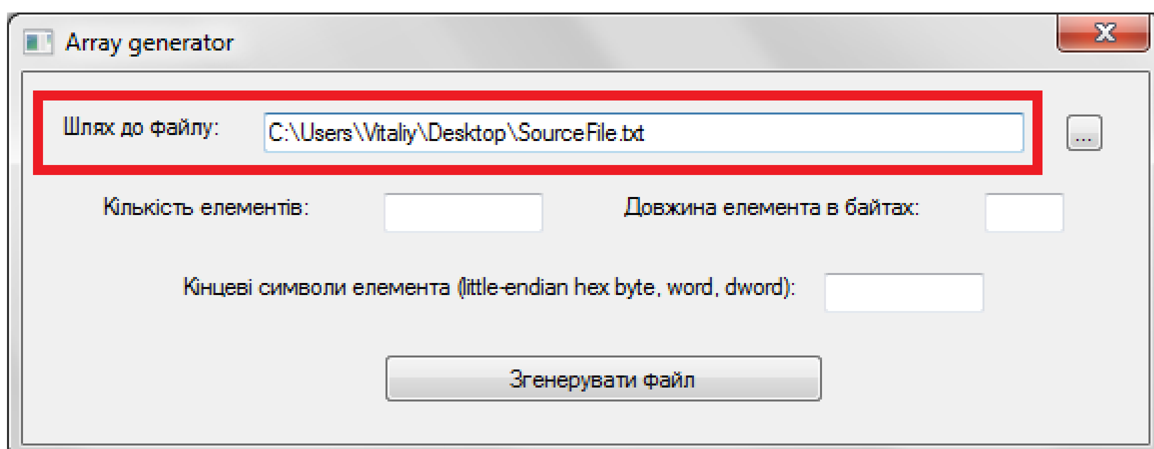

Рис. 3.7. Поле введення шляху до файлу

Якщо потрібно вибрати файл вказавши його у певній директорії, то можна натиснути кнопку підписаною трьома крапками справа, як показано стрілкою на рис.3.8.

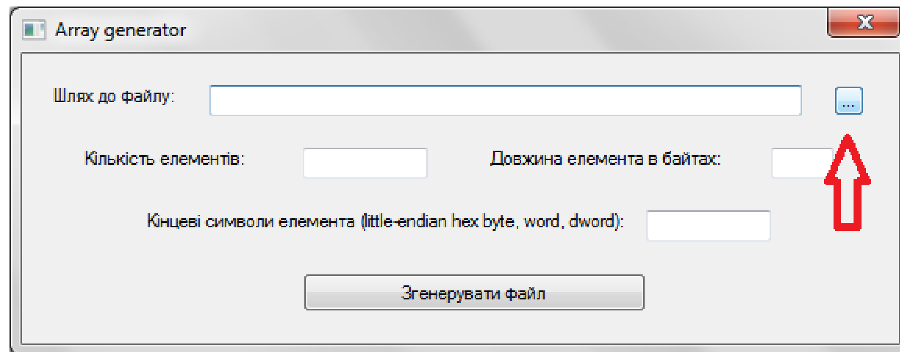

Рис. 3.8. Кнопка вибору файлу

Відкриється системне вікно вибору файлу. Потрібно вказати файл у який програма буде записувати інформацію (рис.3.9).

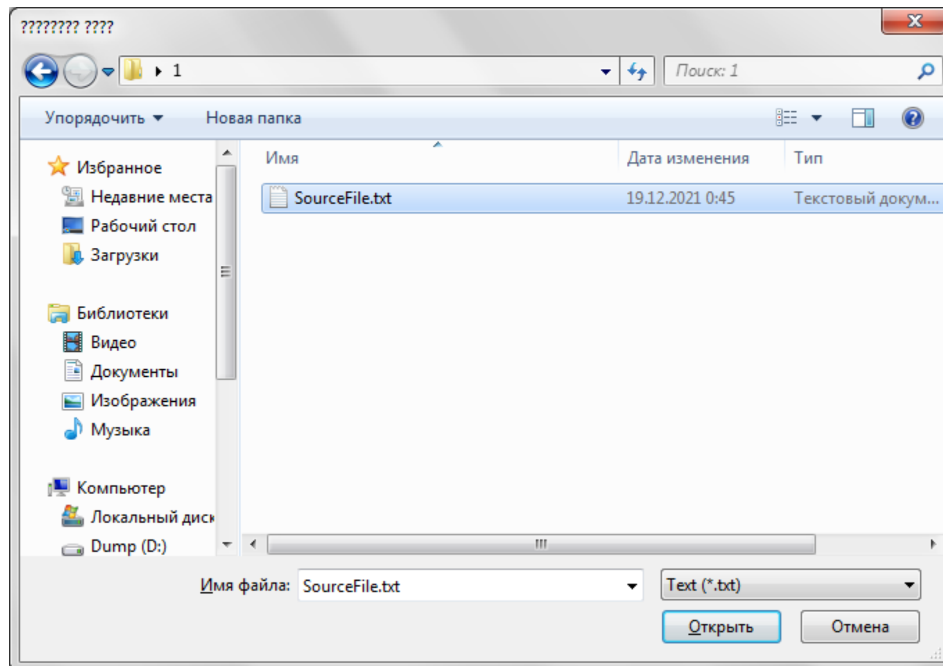

Рис. 3.9. Вибір файлу через системне вікно

Коли вказано шлях до файлу куди потрібно записувати інформацію, потрібно ввести параметри за якими буде створена випадкова інформація і

записана у вказаний файл. Потрібно вказати розмір переліку елементів записавши їхню кількість у поле вказане на рис.3.10.

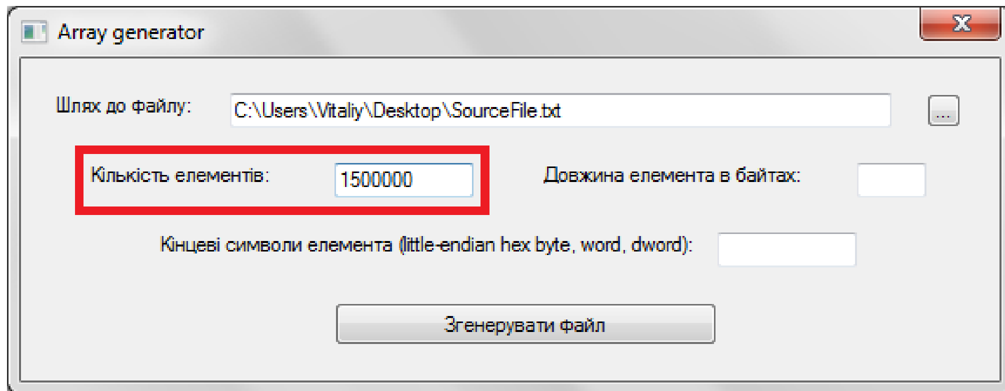

Рис. 3.10. Поле введення кількості випадкових елементів

Також вказується довжина елемента у відповідному полі показаному на рис.3.11. У даній програмі елементи генеруються однакової довжини, визначеної вказаною кількістю у даному полі.

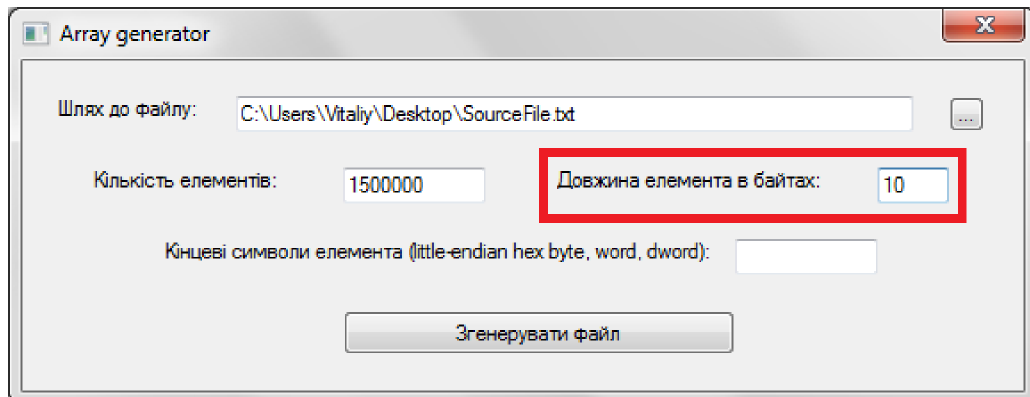

Рис. 3.11. Поле вказівки довжини переліку елементів

На рис.3.12 обводом показано в інтерфейсі поле у якому вводяться спеціальні символи, що позначають кінець рядка. Для операційних систем сімейства “Windows”, використовуються два символи, які позначають кінець рядка. Це 13-ий і 10-ий символи таблиці ASCII. Десятий символ означає зміну рядка, а тринадцятий повернення каретки. Інші операційні системи можуть

використовувати тільки один із символів для позначення переходу на новий рядок, тому програма для сортування даних була розроблена таким чином, щоб могла працювати із файлами незалежно від їхніх символів закінчення.

Вводяться ці символи у вказане поле в шістнадцятковому форматі, у порядку байт від молодшого до найстаршого.

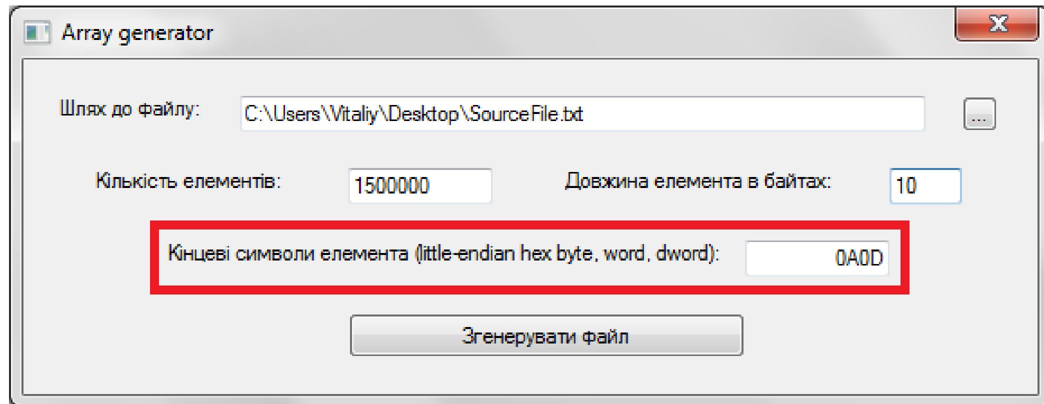

Рис. 3.12. Поле введення службових символі закінчення рядка

Параметри генерування задано залишається тільки натиснути кнопку на рис.3.13 вказаною стрілкою.

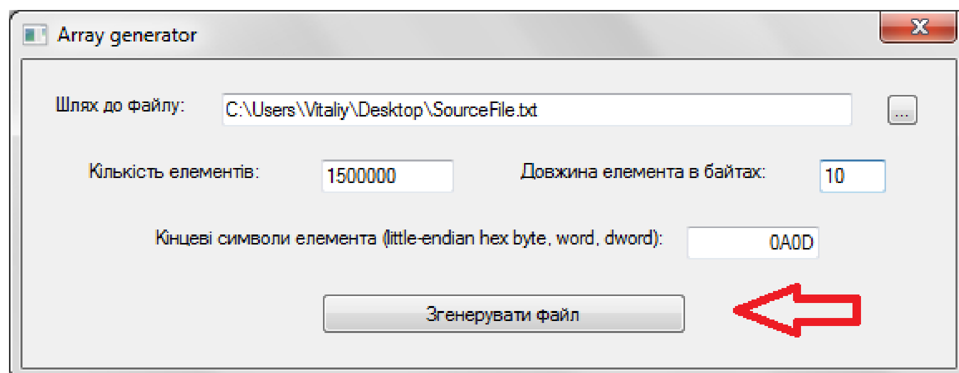

Рис. 3.13. Кнопка генерації випадкових даних

У результаті у файл із назво “SourceFile.txt”, записано перелік із 1.5 мільйона елементів довжиною 10 байт і двома службовими символами закінчення рядка. Розмір файлу відповідно становить 18 мільйонів байт.

Згенеровано масив елементів, які складаються із великих латинських літер, у якості прикладу текстових даних (рис.3.14).

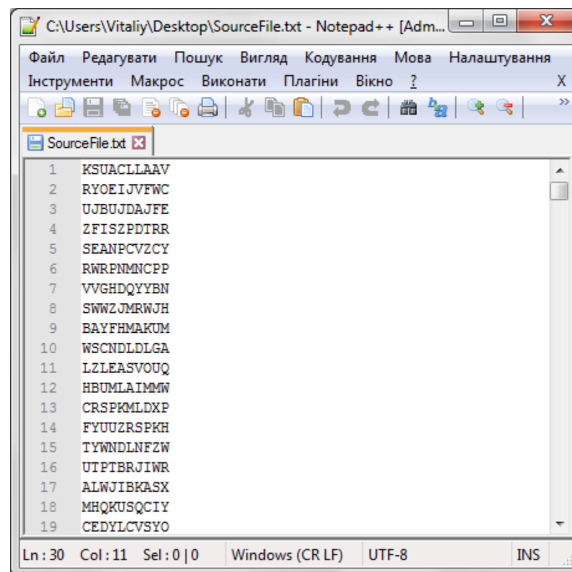

Рис. 3.14. Хаотичні текстові дані

Даний перелік складають елементи однакової довжини генеровані псевдовипадковим алгоритмом.

Дані отримано і залишається виконати їх сортування. Запускаємо програму побітового сортування даних, її інтерфейс зображено на рис.3.15.

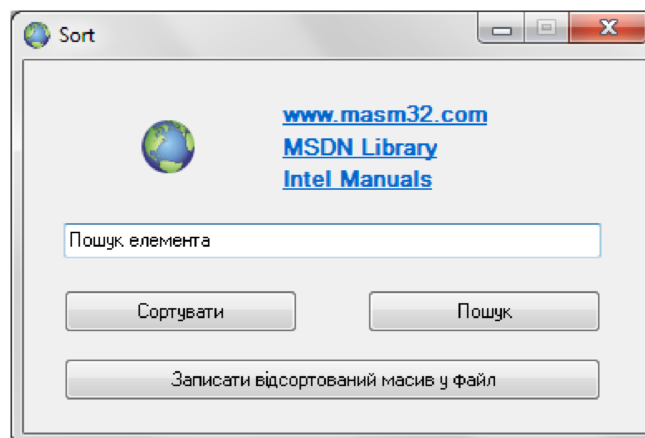

Рис. 3.15. Інтерфейс програми сортування даних

Для сортування даних натискаємо кнопку із назвою “Сортувати” (рис.3.16).

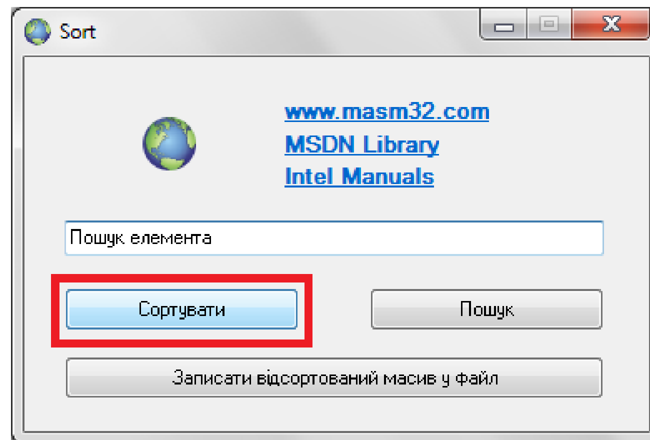

Рис. 3.16. Кнопка виконання сортування даних

Створений файл, із списком елементів, зчитується програмою сортування даних і виконується їхнє впорядкування.

Після завершення роботи видається відповідне повідомлення із виведенням відповідних даних про певні параметри роботи (рис.3.17). У пам'яті було створено спеціальну структуру і зчитано з неї вихідний масив елементів у вигляді списку.

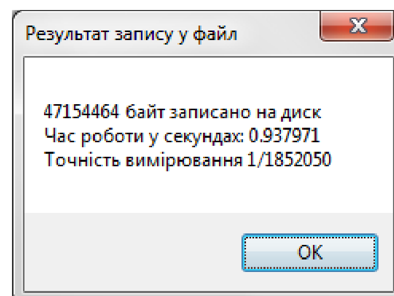

Рис. 3.17. Інформаційне повідомлення

Серед виведених даних у повідомленні вказано інформацію про кількість байт збережених у файл із назвою "DestinationFile.txt". Це структура, яка була утворенна для сортування даних і записана у вигляді масиву, як показано на рис.3.18.

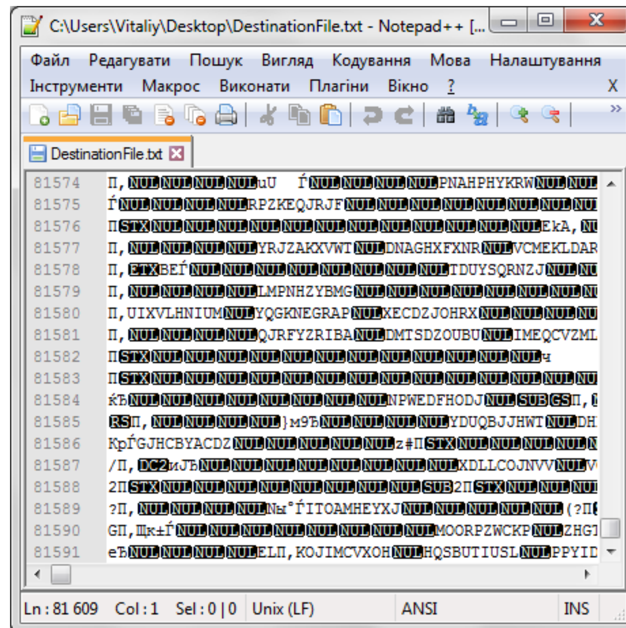

Рис. 3.18. Файл із структурою даних

Для зчитування даних із пам’яті і збереження відсортованого масиву даних, подібним списком як у вхідному файлі, то потрібно натиснути кнопку “Записати відсортований масив у файл” (рис.3.19).

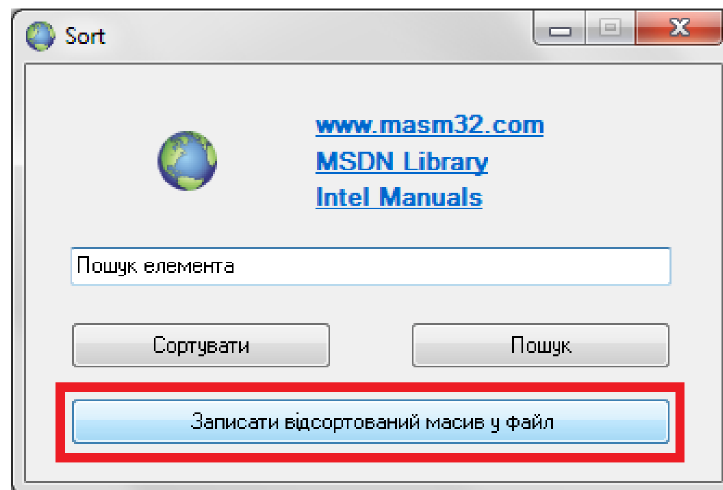

Рис. 3.19. Кнопка запису переліку елементів у файл

У результаті буде створено файл із назвою “SortedArrayFile.txt”.

Нехай для перевірки виберемо елемент із назвою “BZESWGOJNN”, на рис.3.20 рядок із цим елементом виділено темнішим кольором.

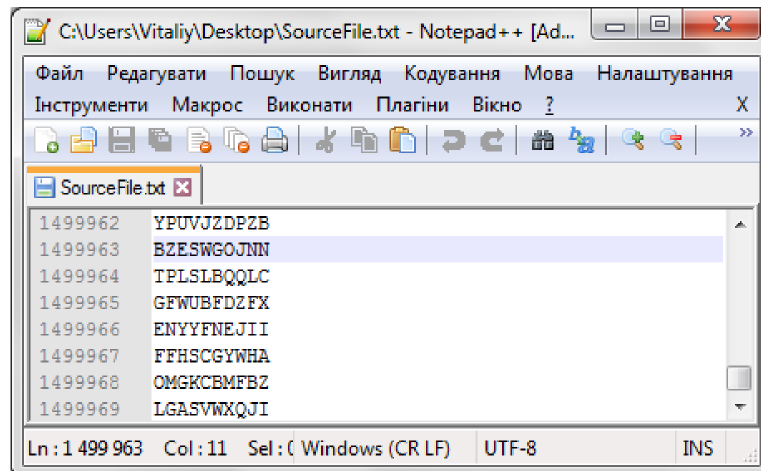

Рис. 3.20. Пошуковий елемент

У програмі сортування даних демонструється пошук елемента по структурі, яка створюється для сортування даного масиву, адже вона швидша за бінарний пошук на певний сталий коефіцієнт. Даний пошук зроблений чисто для демонстрації швидкості роботи пошуку. Він лиш відображає чи шуканий елемент міститься, серед існуючих у структурі, чи – ні. Для практичного застосування такі значення елементів є ключами і містять прикріплену до себе інформацію. Оскільки у даній роботі виконується лиш демонстрація роботи алгоритму сортування і пошуку по його спеціальній структурі, то до значень не прикріплювалася додаткова інформація.

Щоб продемонструвати дієздатність і швидкість роботи пошуку, достатньо лише показати чи такий елемент знайдено у структурі. Для цього у поле “Пошук елемента” вказується елемент, який було обрано для цього і натискається кнопка “Пошук”, як показано на рис.3.21.

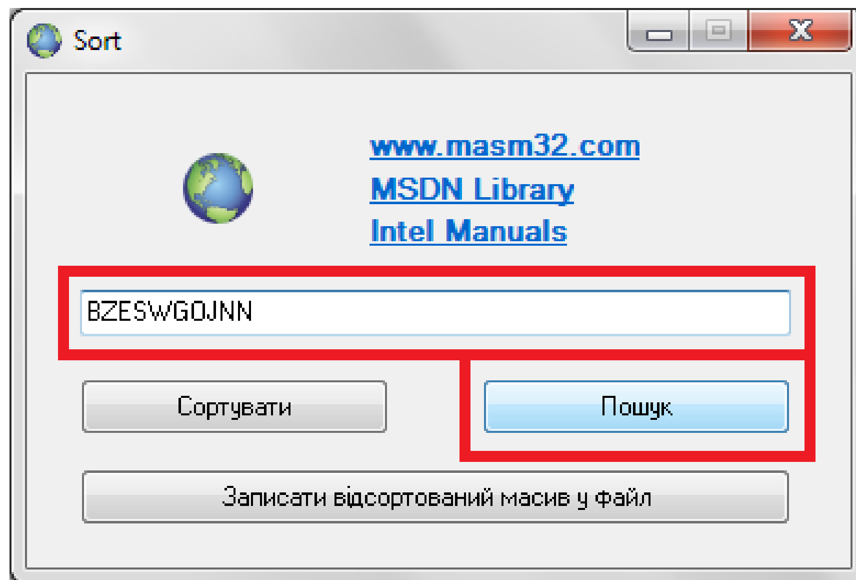

Рис. 3.21. Виконання пошуку елемента

Для замірювання швидкості роботи пошуку використовується системна API (application programming interface) функція із назвою “QueryPerformanceCounter” [18], яка рахує кількість відліків часу із певною точністю. Точність, із якою відбувається вимірювання часу, отримується іншою системною API функцією “QueryPerformanceFrequency” [19]. Програмою обчислюється час і відображається у відповідному повідомленні разом із точністю вимірювання часу та інформацією про те що елемент знайдено (рис.3.22).

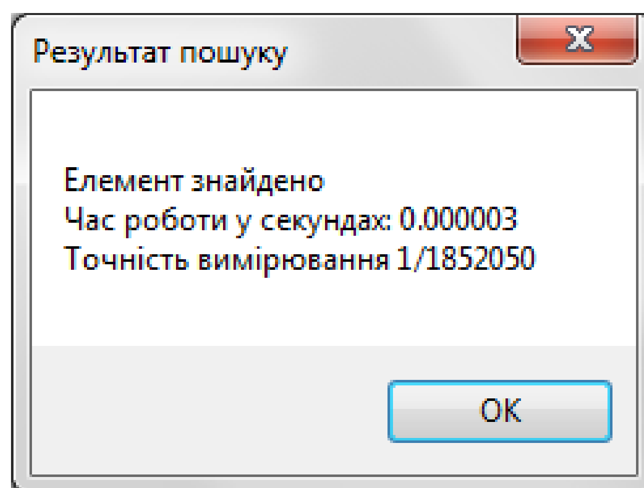

Рис. 3.22. Результат пошуку елемента

Як видно час затрачений на пошук становить приблизно 3 мікросекунди.

Коли елемент не було знайдено, то відображається інше відповідне повідомлення, яке показано на рис.3.23.

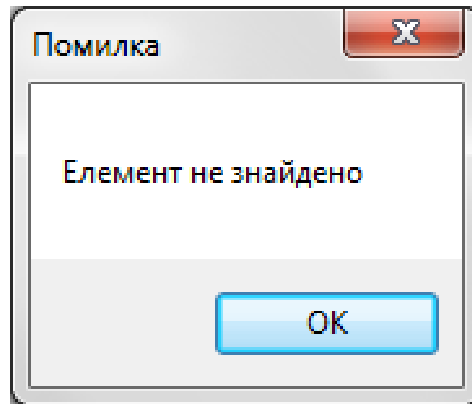

Рис. 3.23. Елемент у переліку відсутній

Перевіримо де цей елемент знаходиться у відсортованому масиві.

Для цього у новоствореному файлі “SortedArrayFile.txt” знайдемо даний елемент. На рис.3.20 зображено, що даний елемент знаходився у 1499963 рядку відсортованого списку, а у переліку відсортованих елементів він вже знаходиться, за порядком, в 113423 рядку (рис.3.24).

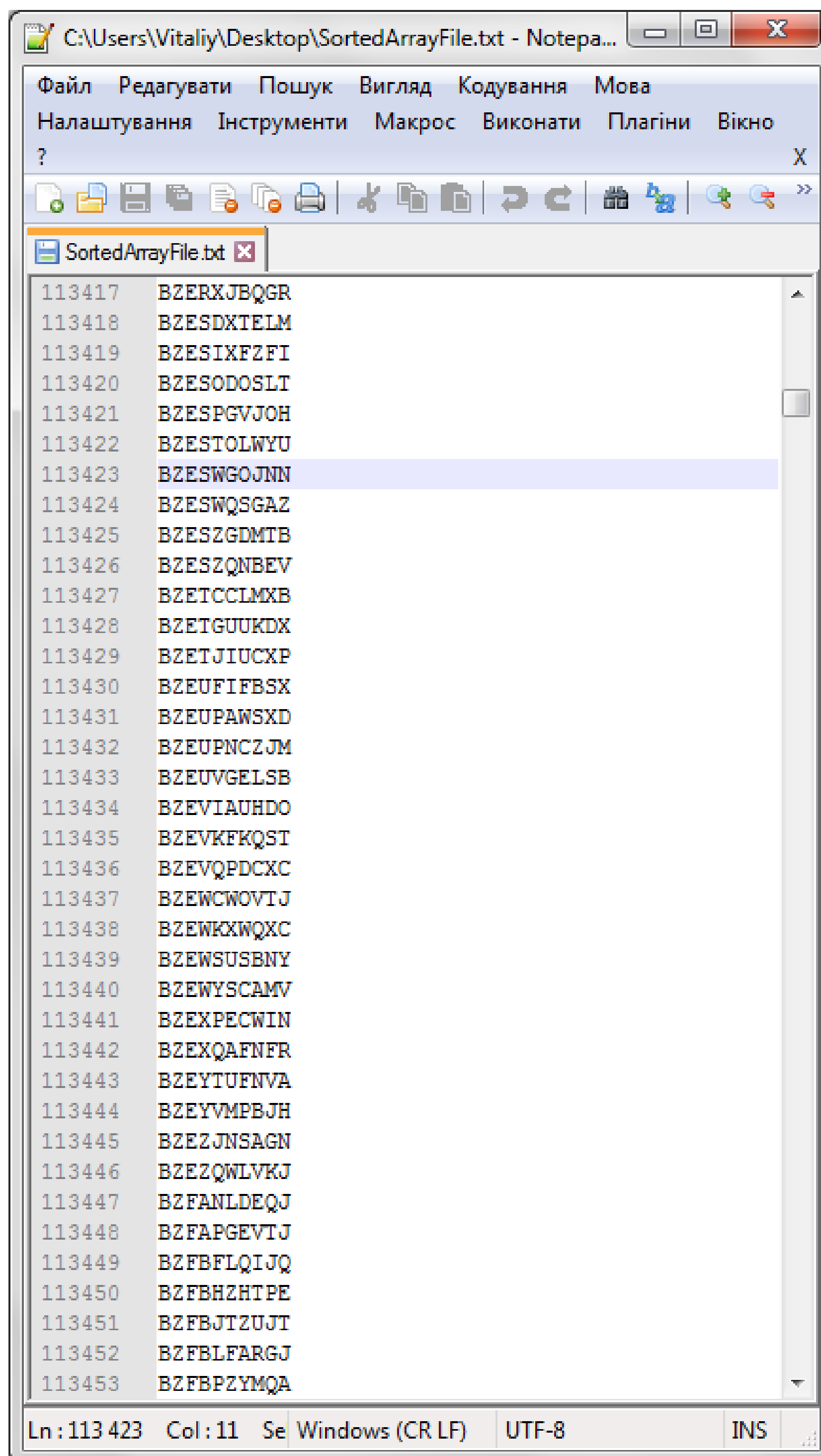

Рис. 3.24. Відсортований перелік елементів

Даний алгоритм також сортує і список із числових даних.

Для прикладу згенерований перелік з 1,5 мільйона числових елементів довжиною 6 байт і два символи закінчення рядка, як показано на рис.3.25.

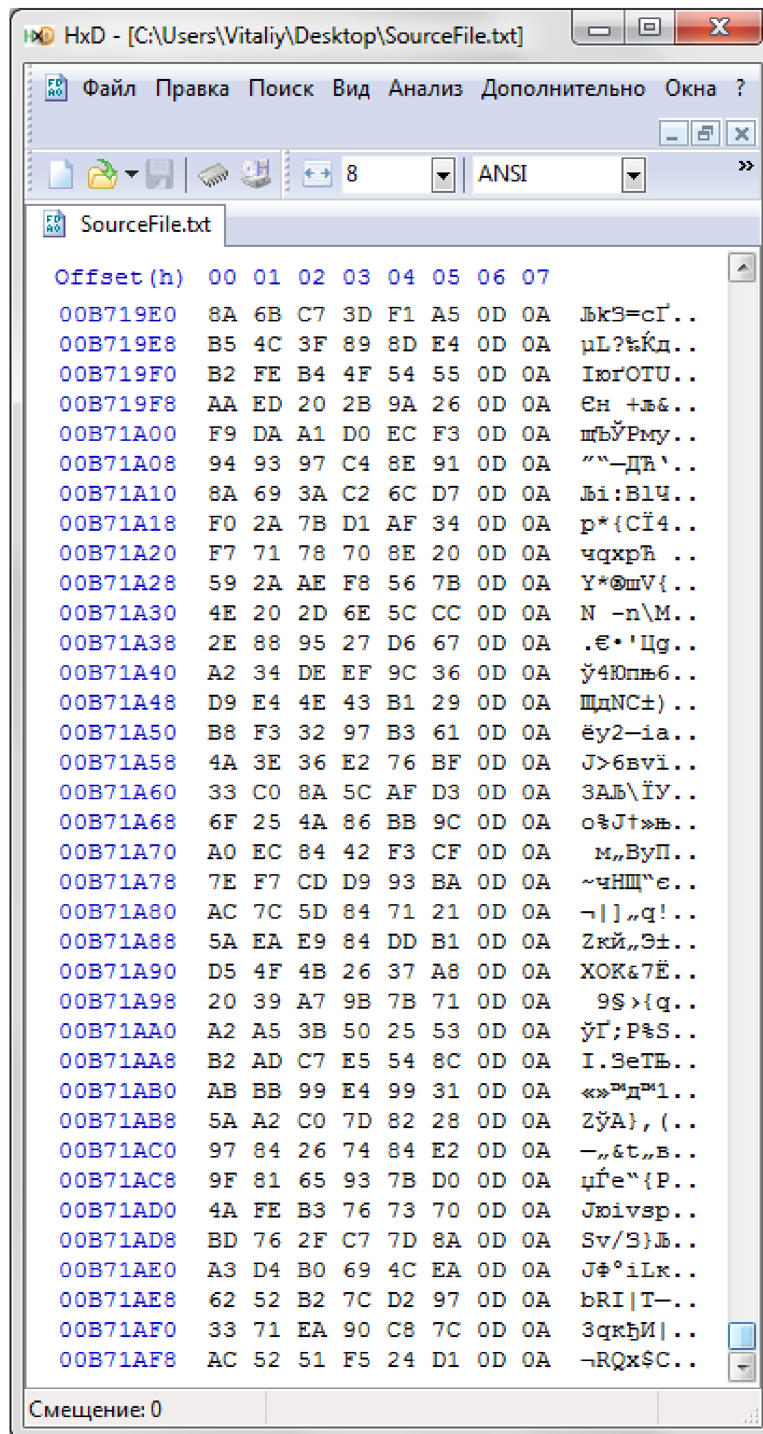

Рис. 3.25. Хаотичний перелік числових елементів

Запускаємо програму і натисненням кнопки виконуємо сортування даних цього переліку елементів. Як можна побачити час сортування даних дещо швидший у даному випадку на відміну від попереднього випадку, хоча кількість елементів така ж (рис.3.26). Це пов’язано із різною довжиною елементів у

першому і другому випадку. Оскільки дана реалізація алгоритму сортування переносить елементи у структуру, яку створює для сортування даних.

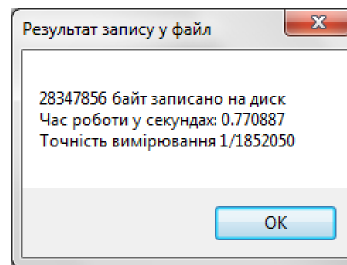

Рис. 3.26. Час сортування числових елементів

Отриманий файл впорядкованих числових елементів можна побачити на рис.3.27, у шістнадцятковому форматі. Для демонстрації збігу цих фалів, було вибрано кінцевий елемент сере невпорядкованого списку і показано його розташування у відсортованому файлі.

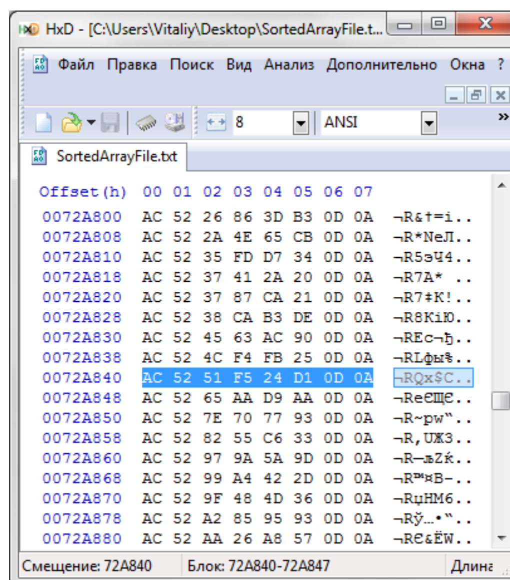

Рис. 3.27. Впорядковані числові дані

Використання мови Assembler для реалізації алгоритму дало змогу створити програмний код який на машинному рівні є вкрай коротким. Це дало змогу отримати високі показники швидкодії і по приблизним оцінкам для кількох десятків тисяч елементів витрачається машинних тактів менше тисячі, що по

міркам комп’ютерних обчислень вважається хорошим показником. Такі параметри даної реалізації алгоритму сортування при потребі дозволяють впроваджувати цей алгоритм і в малопотужні пристрої по двом параметрам малий обсяг машинного коду, та низьке навантаження на процесор.

3.3 Висновки до розділу 3

У даному розділі описано деталі написання коду програми, для методу побітового сортування даних, мовою програмування Assembler і використані джерела для написання низькорівневого коду програми.

Представлено принципи роботи коду програми. Детально описано частини програми і їх функції при виконанні задачі, що реалізує дана програма. Пояснено принципи роботи алгоритму із точки зору низькорівневої реалізації.

Показано інтерфейс програми та описано правила користування для коректної взаємодії із ним.

Виконано тестування програми на пристрої. Результати роботи представлено графічним матеріалом. Показано високу швидкодію даного методу сортування даних на великих масивах вхідних даних.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Комп’ютерні системи та різні прилади для своє роботи використовують електричні джерела живлення. Коли пристрій виконує велику кількість роботи він починає споживати більшу кількість електроенергії. Зростання споживання електроенергії спричиняє нагрівання приладу в цілому або його окремих частин.

Якщо нагрівання приладу надмірно зростає, то це може спричинити підвищення опору певних елементів кола від якого кількість виділення тепла буде зростати ще швидше. Може утворитися неконтрольоване надмірне зростання температури, яке спричинить перегорання елементів приладу.

У сучасних приладах такі ситуації є вкрай рідкісними, але можливими.

Розроблене у даній роботі програмне забезпечення не створює високого навантаження на комп’ютерну систему, якою експлуатується. Відповідно навантаження на систему низьке і ризики спричинення надзвичайної ситуації є майже нульовими. Виходячи з цього дане програмне забезпечення, яке може бути використане у різних комп’ютерних системах, є безпечним для його використання як на малопотужних пристроях, так і на потужних серверних комп’ютерних системах.

Проте, під час виконання різного виду робіт, при яких використовуються персональні комп'ютери, необхідно дотримуватися вимог охорони праці, техніки безпеки та протипожежної безпеки. Одними із основних таких документів є ДСанПін 3.3.2-007-98 “Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин», затверджені постановою Головного державного санітарного лікаря України від 10.12.1998 року № 7 та НПАОП 0.00-7.15-18 "Вимоги щодо безпеки та захисту здоров’я

працівників під час роботи з екранними пристроями", які затверджені наказом Міністерства соціальної політики України від 14.02.2018 № 207 [21].

Обладнання та організація робочого місця ВДТ ЕОМ повинні відповідати вимогам ДСТУ 7299:2013 «Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки» щодо відповідності конструкції всіх елементів робочого місця та їх відносного розташування ергономічним вимогам з урахуванням особливостей та характеру трудової діяльності.

Раціональне планування робочого місця повинно забезпечити найкраще розміщення інструментів та предметів праці, уникати загального дискомфорту, зменшити втому працівників та підвищити продуктивність праці. Також важливим є забезпечення роботодавцем гігієнічних й ергономічних вимог при організації робочих місць, режиму праці і відпочинку при роботі з ЕОМ, відповідності робочого середовища приміщень, де здійснюється експлуатація електронно-обчислювальних машин (ЕОМ) з ВДТ.

Заходи щодо усунення ризику ураження електричним струмом зводяться до правильного розміщення обладнання та електричних кабелів. Для забезпечення пожежної безпеки необхідно у приміщеннях використовувати приховану електромережу, ввімкнення та вимкнення живлення виконувати обладнанням за допомогою стандартних вимикачів, використовувати надійні розетки з важкозаймистих матеріалів. А також - регулярно очищати внутрішні частини комп'ютерів та інше обладнання від пилу. Щодо розташування робочих місць, то вони мають бути розміщені на відстані, що є не меншою 1,5 м від стіни з вікнами, а від інших стін - на відстані близько 1 м, між бічними поверхнями ВДТ – 1,2 м; від задньої площини одного ВДТ до екрану іншого – 2,5 м.

Дотримання вимог цих документів значно знижує наслідки несприятливої дії на працівників шкідливих та небезпечних факторів, які супроводжують роботу з комп'ютерною технікою, зокрема можливість зорових, емоційних напружень, серцево-судинних захворювань та ін.

4.2. Організація та забезпечення заходів щодо розосередження робітників та службовців суб’єктів господарювання, що продовжують свою роботу в особливий період, і евакуація населення.

Розосередження і евакуація населення – це один із заходів захисту населення у надзвичайних ситуаціях мирного і воєнного часу, Під розосередженням мається на увазі організований вивіз з міст і інших населених пунктів, розміщення в зоні за містом вільної від праці зміни і службовців, які продовжують роботу у надзвичайних ситуаціях. До категорії розосередження відноситься також персонал об'єктів, які забезпечують життєдіяльність міста.

Робітники і службовці, які відносяться до категорії розосередження після вивозу і розселення їх, позмінно виїжджають в місто для роботи на своїх підприємствах, а по закінченні роботи повертаються в зону за містом для відпочинку.

Евакуація – це організований вивіз або вивід з міст і інших населених пунктів та розміщення у заміській зоні решти населення, а також вивіз населення із зон можливого затоплення, радіоактивного зараження і інших випадках. На відміну від розосереджених евакуйовані постійно проживають у заміській зоні до особливого розпорядження.

Райони розселення робітників і службовців в заміській зоні повинні знаходитись на такій відстані від міста, яка гарантувала б їх безпеку, а на переїзд людей для роботи в міста і їх повернення в заміську зону для відпочинку, витрачалась би мінімальна кількість часу.

Райони розселення розосереджених доцільно також розміщувати поблизу залізничних станцій і автомобільно-шляхових магістралей.

Розселюють робітників і службовців з збереженням виробничого принципу. При цьому зберігається цілісність підприємства, полегшується відправка робочих

змін в місто на роботу і забезпечення людей продуктами харчування та медичним обслуговуванням.

Робітників і службовців підприємства, яке переносить свою виробничу діяльність у заміську зону, розташовують поблизу виробничих баз за районами розміщення робітників і службовців підприємств, які продовжують працювати в місті. Евакуйоване населення, не зв'язане з виробництвом і яке не є членами сімей розосереджених робітників та службовців, розміщують у більш віддалені райони заміської зони, а населення, евакуйоване із зон можливого затоплення, – в населених пунктах, які знаходяться поблизу цих зон.

Евакуація передбачає вивід і вивіз населення із міст у безпечні райони в усіх напрямках від міста.

Розосередження і евакуація в багато раз знижує густоту населення міст, а зрозуміло й втрати населення можуть бути значно зменшені.

Розосередження і евакуація робітників, службовців і членів їх сімей організовуються і проводяться за виробничим принципом, тобто по лінії об'єктів, а евакуація населення, не зв'язаного з виробництвом, – за територіальним принципом – за місцем проживання через домоуправління і дирекції експлуатації приміщень. Діти переважно евакуюються разом з батьками, але не виключається можливість вивезення їх зі школами і дитячими садками.

Для проведення розосередження і евакуації використовуються всі види громадського транспорту (залізничний, автомобільний, водний), не зайнятого воєнними і невідкладними виробничими і господарськими перевезеннями, а також транспорт індивідуального користування.

Всі роботи по проведенню і організації розосередження та евакуації здійснюються у відповідності з планом і вказівками начальника ЦО об'єкту.

Для керівництва розосередженням і евакуацією населення на об'єкті створюється евакуаційна комісія, а на великих об'єктах, крім цього можуть створюватися збірні евакуаційні пункти (ЗЕП). Наказом начальника ЦО

підприємства створюється адміністрація ЗЕГІ. Головою об’єктової евакуаційної комісії призначається один із заступників начальника ЦО.

Населення міста про початок евакуації повідомляється через підприємства, навчальні заклади, домоуправління, міліцію а також радіотрансляційну мережу і місцеве телебачення.

Отримавши повідомлення про початок розосередження і евакуації, населення повинно підготувати і взяти з собою документи, гроші, необхідні речі і запаси продуктів та з'явитися на збірний евакуаційний пункт в попередньо визначеній годині.

У випадку, якщо робітників і службовців розмістити разом з сім'ями неможливо, членів їх сімей евакуюють окремо в більш віддалені райони за напрямком розосередження і час їх появи на збірний евакуаційний пункт встановлюється окремо.

При передбаченні надзвичайної ситуації проводять заходи по приведенню станцій, пунктів висадки, прийомних евакуаційних пунктів в готовність до прийому населення.

Евакуйоване населення залучається до роботи на підприємствах, вивезених з міста, що продовжують роботу в заміській зоні, у фермерських, колективних і інших господарствах.

4.3. Висновки до розділу 4

У підрозділі охорони праці описано безпечність експлуатації комп’ютерних засобів, які використовують програмне забезпечення, яке розроблене і описане у даній роботі разом із правилами його використання.

У наступному підрозділі проводиться аналіз евакуації населення і розосередження робітників та службовців. Виявлено і описано дії, які потрібно виконувати для зменшення втрат населення. При виконанні таких дій важливо дотримуватись правил, що визначенні у нормативних документах.

ВИСНОВКИ

У роботі виконано поставлені цілі і отримано наступні результати:

1. Проведено аналіз алгоритмів сортування даних.
2. Виділено основні типи алгоритмів та їх принципи роботи.
3. Визначено і оптимізовано основні моделі роботи алгоритмів сортування даних.
4. Розроблено метод побітового сортування даних.
5. Створено блок-схему алгоритму, на основі спроектованого алгоритму сортування даних.
6. Описано швидкодію алгоритму та його теоретичні характеристики роботи.
7. Написано програмне забезпечення, оптимальним чином, яке реалізує розроблений метод побітового сортування даних.
8. Показано результати тестування даного алгоритму і ефективність його роботи на комп’ютерній системі при сортуванні різних типів даних.
9. Визначено і описано області застосування алгоритмів сортування даних для використання в реальних програмах, різних засобах та комп’ютерних системах, які виконують задачі аналізу, опрацювання, класифікації і сортування різного виду даних.

СПИСОК ЛІТЕРАТУРИ

1. В. Семеген, Н. Луцик. Актуальність створення оптимального алгоритму сортування даних. Матеріали IX Науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8–9 грудня 2021 року). Тернопіль: ТНТУ, 2021. С. 127.

2. В. Семеген, Н. Луцик. Метод побітового сортування даних в комп'ютерних системах. Матеріали IX Науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (8–9 грудня 2021 року). Тернопіль: ТНТУ, 2021. С. 128.

3. Robert Sedgewick and Kevin Wayne. Algorithms, 4th edition. AddisonWesley, Upper Saddle River, NJ, USA, 2011, 488 p.

4. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein (2009) Introduction to Algorithms, Third Edition (Cambridge, Massachusetts London, England), 1292 p.

5. Donald E. Knuth (1998) The Art Of Computer Programming: Sorting And Searching. Volume 3 (Addison Wesley Series In Computer Science And Information Processing), 722 p.

6. Рогушина Ю. В. Алгоритм сортування // Велика українська енциклопедія. URL: https://vue.gov.ua/Алгоритм сортування (дата звернення: 01.12.2021).

7. Mikhai J. Atallah, editor. Algorithms and Theory of Computation Handbook/ CRC Press, 1999.

8. Ахо А. Структуры данных и алгоритмы / Ахо А., Хопкрофт Дж., Ульман Дж. ; пер. с англ. А. А. Минько. — М. : Издательский дом «Вильямс», 2000. — 382 с.

9. Кузюрин Н. Н. Эффективные алгоритмы и сложность вичислений / Н. Н. Кузюрин, С. А. Фомин. — М. : Институт системного программирования, 2008. — 357 с.

10. Макконнел Дж. Основы современных алгоритмов / Дж. Макконнел ; пер. с англ. А. К. Малюк. — М. : Техносфера, 2004. — 368 с.

11. Oded Regev. A Subexponential Time Algorithm for the Dihedral Hidden Subgroup Problem with Polynomial Space. URL: https://archive.org/details/arxiv-quant-ph0406151 (дата звернення: 17.12.2021).

12. Is radix sort faster than quicksort for integer arrays?. URL: https://erik.gorset.no/2011/04/radix-sort-is-faster-than-quicksort.html (дата звернення: 18.12.2021).

13. Note 6: Sorting Algorithms in Data Structure for Application. URL: http://faculty.tamuc.edu/dcreider/csci520/Note520/Note%206.htm (дата звернення: 20.12.2021).

14. Пирамидальная сортировка. URL: https://works.doklad.ru/view/I9h4R9kRVx0.html (дата звернення: 20.11.2021).

15. Skiena, Steven S. (2008). "4.5: Mergesort: Sorting by Divide-and-Conquer". The Algorithm Design Manual (2nd ed.). Springer. pp. 120–125. ISBN 978-1-84800-069-8. Sun Microsystems. "Arrays API (Java SE 6)". URL: https://docs.oracle.com/javase/6/docs/api/java/util/Arrays.html

16. Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 2 (2A, 2B, 2C & 2D): Instruction Set Reference, A-Z . URL: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf (дата звернення: 27.11.2021).

17. Build desktop Windows apps using the Win32 API. URL: https://docs.microsoft.com/en-us/windows/win32/ (дата звернення: 03.12.2021).

18. QueryPerformanceCounter function (profileapi.h). URL: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter (дата звернення: 01.12.2021).

19. QueryPerformanceFrequency function (profileapi.h). URL: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancefrequency (дата звернення: 01.12.2021).

20. Sara Baase and Alan Van Gelder/ Computer Algorithms: Introduction to Design and Analis. Addison-Wesley, third edition, 2000.

21. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text (дата звернення: 17.11.2021).

22. Плиско В. Е. Теория алгоритмов / В. Е. Плиско, В. Н. Крупский. — М. : ACADEMIA, 2009. — 38 с

23. Arne Andersson. Balanced search trees made simple. In Proceedings of the Third Workshop on Algorithms and Data Structures, volume 709 of Lecture Notes in Computer Science, pages 60–71. Springer, 1993.

24. Шинкаренко В. І. Особливості практичного застосування показників обчислювальної складності алгоритмів / В. І. Шинкаренко / Проблеми програмування. — 2008. — № 2-3. — С. 53—67.

Додаток А

Тези конференції

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

IX НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

8–9 грудня 2021 року

ТЕРНОПІЛЬ
2021

А.Я. Осадца, Є.В. Тиш
АЛГОРИТМИ ТА КОМП'ЮТЕРИЗОВАНІ ЗАСОБИ ПЕРЕДАЧІ ДАНИХ В БЛОЦІ КЕРУВАННЯ ТА ІНДИКАЦІЇ ДВОДЗЕРКАЛЬНОЇ АНТЕНИ
A.Y. Osadtsa, Ie.V. Tysh, Ph. D. Assoc. Prof.
ALGORITHMS AND COMPUTERIZED MEANS OF DATA TRANSMISSION FOR A TWO-MIRROR ANTENNA'S CONTROL UNIT AND INDICATION DEVELOPMENT 121

О.В. Осійчук, Є.В. Тиш
ПЕРЕВАГИ ТА НЕДОЛІКИ ВИКОРИСТАННЯ КОМП'ЮТЕРНОЇ МЕРЕЖІ З ВИДІЛЕНИМ СЕРВЕРОМ
O.V. Oseechuk, Ie.V. Tysh
ADVANTAGES AND DISADVANTAGES OF USING A COMPUTER NETWORK WITH A DEDICATED SERVER 122

С. Петрук, М. Хвостівський
МЕТОД ТА ПРОГРАМННЕ ЗАБЕЗПЕЧЕННЯ ОБРОБКИ ЕЛЕКТРОГАСТРОЕНЕТРОСИГНАЛУ
S. Petruk, M. Khvostivskyy
METHOD AND SOFTWARE OF ELECTROGASTROENETROSIGNAL PROCESSING 123

Д.В. Романов, Г.М. Осухівська, А.М. Паламар
ФУНКЦІОНАЛЬНА СХЕМА СИСТЕМИ КЕРУВАННЯ ЗОВНІШНІМ ОСВІТЛЕННЯМ НА ОСНОВІ ТЕХНОЛОГІЇ LORA
D.V. Romanov, H.M. Osukhivska, A.M. Palamar
FUNCTIONAL DIAGRAM OF THE OUTDOOR LIGHTING CONTROL SYSTEM BASED ON LORA TECHNOLOGY 124

Б. Семеген, С. Лупенко
АКТУАЛЬНІСТЬ РОЗРОБКИ МЕТОДІВ ПІДВИЩЕННЯ КРИПТОСТІЙКОСТІ СЛАБКИХ АЛГОРИТМІВ ШИФРУВАННЯ
B. Semehen, S. Lupenko
ACTUALITY OF DEVELOPMENT OF METHODS OF INCREASING CRYPTIC RESISTANCE OF WEAK ENCRYPTION ALGORITHMS 125

Б. Семеген, В. Семеген, С. Лупенко
МЕТОД ПІДВИЩЕННЯ КРИПТОСТІЙКОСТІ СИМЕТРИЧНИХ АЛГОРИТМІВ ШИФРУВАННЯ
B. Semehen, V. Semehen, S. Lupenko
METHODS OF INCREASING SYMMETRIC ENCRYPTION ALGORITHMS' CRYPTOSECURITY 126

В. Семеген, Н. Луцик
АКТУАЛЬНІСТЬ СТВОРЕННЯ ОПТИМАЛЬНОГО АЛГОРИТМУ СОРТУВАННЯ ДАНИХ
V. Semehen, N. Lutsyk
ACTUALITY OF CREATING AN OPTIMAL DATA SORTING ALGORITHM 127

В. Семеген, Н. Луцик
МЕТОД ПОБІТОВОГО СОРТУВАННЯ ДАНИХ В КОМП'ЮТЕРНИХ СИСТЕМАХ
V. Semehen, N. Lutsyk
BITWISE DATA SORTHING METHOD IN COMPUTER SYSTEMS 128

УДК 004.424.52:004.42
В. Семеген, Н. Луцик, Ph.D., доцент
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

АКТУАЛЬНІСТЬ СТВОРЕННЯ ОПТИМАЛЬНОГО АЛГОРИТМУ СОРТУВАННЯ ДАНИХ

UDC 004.424.52:004.42
V. Semehen, N. Lutsyk, Ph.D., Assoc. Prof.

ACTUALITY OF CREATING AN OPTIMAL DATA SORTING ALGORITHM

Виробники комп'ютерів 1960-х підрахували що більш ніж 25% часу роботи їх комп'ютерів витрачалося на сортування даних [1].

На сьогоднішній день відсоток ймовірно змінився, але все ж залишився значним. Сучасні комп'ютерні системи для своєї роботи використовують велику кількість програмного забезпечення, яка включає у свою роботу системи управління базами даних, пошукові системи, інтерфейси взаємодії із користувачем. Для пошуку, опрацювання і представлення результатів використовують алгоритми сортування даних. Для таких систем важлива швидкодія, тому для оптимальної роботи вибираються кращі серед існуючих алгоритмів сортування даних.

На даний час розроблено чимало алгоритмів сортування даних серед яких велика кількість знаходить своє практичне застосування у різних засобах опрацювання інформації [2].

Недоліками відомих алгоритмів сортування є складність, або мала швидкодія.

Тому розробка простого і швидкого алгоритму сортування даних для комп'ютерних систем є актуальною та важливою задачею.

Алгоритми сортування бувають двох типів – з додатковою пам'яттю і без неї [1]. Алгоритми без додаткової пам'яті працюють на порівняннях і перестановках елементів. Часто реальні реалізації таких алгоритмів використовують адресне сортування у зв'язку із тим що розмір елементів великий і доцільніше використовувати адреси невеликого розміру замість перестановки самих елементів. Тому на практиці алгоритми сортування використовують додаткову пам'ять лінійної складності, або вище, а також порівняння, перестановки і адресні переходи [3].

У даному дослідженні пропонується максимально ефективно використати додаткову пам'ять для ефективної роботи алгоритму.

В пам'яті створюється структура у вигляді дерева яка має лінійну просторову складність і позволяє деякі етапи роботи алгоритмів скоротити, а інші забрати. Одним із таких принципів роботи алгоритмів є перестановки елементів місцями, які у алгоритмі сортування можуть бути відсутніми, через використання спеціальної структури у пам'яті.

За такими критеріями створюється алгоритм який опрацьовує побітово елементи і створює деревовидну структуру із якої вкінці зчитуються всі дані.

Література.

1. Donald E. Knuth (1998) The Art Of Computer Programming: Sorting And Searching. Volume 3 (Addison Wesley Series In Computer Science And Information Processing), 722 p.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein (2009) Introduction to Algorithms, Third Edition (Cambridge, Massachusetts London, England), 1292 p.
3. Robert Sedgewick and Kevin Wayne. Algorithms, 4th edition. AddisonWesley, Upper Saddle River, NJ, USA, 2011, 488 p.

УДК 004.424.52:004.42
В. Семеген, Н. Луцик, Ph.D., доцент
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

МЕТОД ПОБІТОВОГО СОРТУВАННЯ ДАНИХ В КОМП'ЮТЕРНИХ СИСТЕМАХ

UDC 004.424.52:004.42
V. Semehen, N. Lutsyk, Ph.D., Assoc. Prof.

BITWISE DATA SORTHING METHOD IN COMPUTER SYSTEMS

В даній роботі представлений новий метод побітового сортування даних в комп'ютерних системах. Його відмінність від інших полягає в тому, що кількість порівнянь спрощено до лінійної складності, перестановки елементів місцями відсутні, а сортування відбувається тільки за рахунок створення спеціальної структури, в якій при занесені елементів, лише четвертина з них буде порівняна з іншими елементами [1–3].

На рисунку 1 зображена блок-схема методу побітового сортування даних.

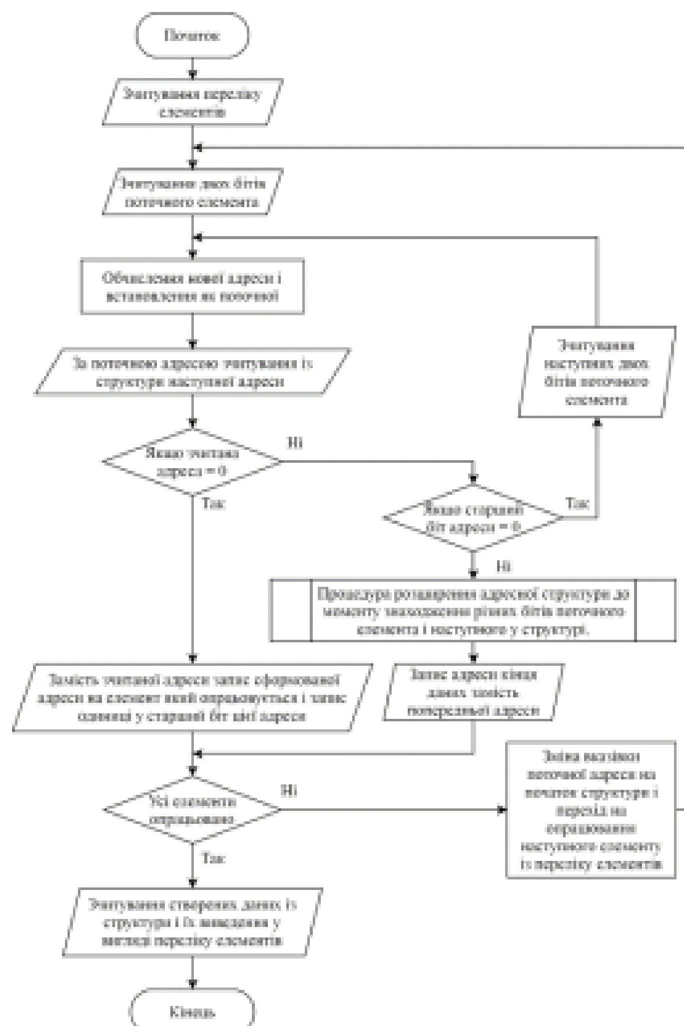

Рисунок 1. Блок-схема алгоритму побітового сортування даних

Даний алгоритм виконує один прохід по елементам із яких створюється оптимальна структура даних, а другий прохід зчитування із структури. Він має квазілінійну часову складність – це пов'язано із структурою яка створюється, бо саме таку складність по часу має обхід структури при зчитуванні усіх елементів.

Література.

1. Donald E. Knuth (1998) The Art Of Computer Programming: Sorting And Searching. Volume 3 (Addison Wesley Series In Computer Science And Information Processing), 722 p.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein (2009) Introduction to Algorithms, Third Edition (Cambridge, Massachusetts London, England), 1292 p.
3. Robert Sedgewick and Kevin Wayne. Algorithms, 4th edition. AddisonWesley, Upper Saddle River, NJ, USA, 2011, 488 p.

Додаток Б

Текст програмного забезпечення методу побітового сортування даних

```
; ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
   include \masm64\include64\masm64rt.inc
   include \masm64\include64\msvcrt.inc
   includelib \masm64\lib64\msvcrt.lib
;OPTION PROLOGUE:NONE
;OPTION EPILOGUE:NONE
;OPTION PROLOGUE:PrologueDef
;OPTION EPILOGUE:EpilogueDef
;NOSTACKFRAME; no prologue and epilogue
;STACKFRAME; prologue and epilogue
   ; --------------------------------------------
   real8$ MACRO r8value:req
     LOCAL buffer, adbuffer
      .data?
        buffer BYTE 320 dup(?)
      IFNDEF r8fmt
      .data
        r8fmt  BYTE "%lf", 0
        adbuffer  dq buffer
      ENDIF
      .code
        mov    buffer[0], 0
        invoke vc_sprintf, ADDR buffer, ADDR r8fmt, r8value
        EXITM <ptr$(buffer)>;adbuffer, addr buffer, reparg(buffer)
   ENDM
   cat1$ MACRO memory:REQ, arguments:VARARG
        ; Приклад використання даного макросу, який перероблено із 32 бітного під 64 біт,
        ; але так як перші 4 значення записуються у регістри, то
        ; будь-яка функція у макросі наприклад (sprintf у макросі real8)
        ; перезаписуватиме регістри. У наведеному прикладі адреса буфера
```

```
        ; перезаписшеться real8
        ;mov wmsg, cat1$(ptr$(buffer),"Час роботи у секундах: ", real8$(TimerInSecond))
      LOCAL txtres
      LOCAL spare
      LOCAL pcount
       spare equ <>
       txtres equ <>
       spare CATSTR spare, tstarg(memory)  ;; test if 1st arg is quoted text
       pcount = 0
        FOR arg, <arguments>
          txtres CATSTR txtres,<,>,reparg(arg)
          pcount = pcount + 1            ;; count arguments
        ENDM
       txtres CATSTR <invoke szmulticat,>,<memory,>, %pcount, txtres       ;; append memory, number to lead string and arguments
       txtres                          ;; put result in code
      EXITM <rax>
     ENDM
     .data
        SourceFileName                 db "SourceFile.txt",0
        DestinationFileName db "DestinationFile.txt",0
     .data?
      hInstance dq ?
      hIcon     dq ?
      hLink1    dq ?
      hLink2    dq ?
      hLink3    dq ?
        SaveRSP       dq ?
        hEdit1 dq ?
     .code
   ; ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
   entry_point proc
```

```
  ; ----------------------
  ; get the instance handle
  ; ----------------------
    mov hInstance, rv(GetModuleHandle,0)
  ; ---------------------------------------------------------
  ; set the icon loaded size here to match the original icon size
  ; ---------------------------------------------------------
    mov hIcon, rv(LoadImage,hInstance,10,IMAGE_ICON,32,32,LR_DEFAULTCOLOR)
  ; --------------------
  ; create a modal dialog
  ; --------------------
    invoke DialogBoxParam,hInstance,100,0,ADDR main,hIcon
  ; -------------------------------
  ; terminate app when dialog closes
  ; -------------------------------
    invoke ExitProcess,0
    ret
entry_point endp
; ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤
main proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,lParam:QWORD
    LOCAL patn  :QWORD
    LOCAL pMem  :QWORD
    LOCAL fname :QWORD
    LOCAL wmsg  :QWORD
    LOCAL str1  :QWORD
    LOCAL buffer[320]:BYTE
    LOCAL hFile :QWORD                    ; хендл файла
    ;LOCAL cloc  :QWORD                                          ; позиція вказіівника
читання/запису
    LOCAL flen  :QWORD                    ; довжина записаних даних
        local flenSourceFile   :qword
        local hMemSourceFile        :qword
        local LenBufSourceFile      :qword
        local hMemDistinationFile   :qword
```

```
        LOCAL TimerInSecond[2]          :QWORD                  ;       час       роботи
алгоритму у секундах
        LOCAL TimerFrequency            :QWORD                  ; частота таймера
    LOCAL hFont :QWORD
    LOCAL nmh   :NMHDR

    .switch uMsg
     .case WM_INITDIALOG
     ; -------------------------------------------------------
     ; lParam is the icon handle passed from DialogBoxParam()
     ; -------------------------------------------------------
      invoke SendMessage,hWin,WM_SETICON,1,lParam     ; set the icon for the dialog
      invoke SendMessage,rv(GetDlgItem,hWin,102), \   ; set the icon in the client area
          STM_SETIMAGE,IMAGE_ICON,lParam
      invoke SetWindowText,hWin,"Sort"
      mov hFont, GetFontHandle("Arial",16,600)
      mov hLink1, rv(GetDlgItem,hWin,103)
      invoke SendMessage,hLink1,WM_SETFONT,hFont,TRUE
      mov hLink2, rv(GetDlgItem,hWin,104)
      invoke SendMessage,hLink2,WM_SETFONT,hFont,TRUE
      mov hLink3, rv(GetDlgItem,hWin,105)
      invoke SendMessage,hLink3,WM_SETFONT,hFont,TRUE
              mov hEdit1, rv(GetDlgItem,hWin,107)     ; second edit control handle

     ; -----------------------------------------------------
     ; set the focus to the first control by returning TRUE
     ; -----------------------------------------------------
      .return TRUE
     .case WM_COMMAND
      .switch wParam
       .case 106     ;IDC_BTN1   Сортувати
        ;           mov patn, CTXT("всі файли",0,"*.*",0)
                comment*-------------------------------------------------------------------------------
        mov fname, OpenFileDlg(hWin,hInstance,"Вибрати файл",patn)
```

```
cmp BYTE PTR [rax], 0
jne @F
return 0
@@:
        -------------------------------------------------------------------------- *
;invoke exist proc lpszFileName:QWORD
invoke load_file, ADDR SourceFileName
mov hMemSourceFile, rax
mov LenBufSourceFile, rcx

shl rcx, 3
mov flenSourceFile, rcx
mov hMemDistinationFile, alloc(flenSourceFile)

; Таймер для засікання часу роботи коду
lea             rcx, TimerInSecond
call    QueryPerformanceCounter

push    rbx
push    rsi
push    rdi
push    r12
push    rbp
mov     SaveRSP, rsp
;>---------------------------<Метод впорядкованого запису>---------------------------<
;       Адресне сортування, купа і індексація
; !---------------------!
; [xx..xx[ <- біти;] ([опис])] - Формат опису наведеного нижче
; [[31 (0 - по адресі розміщена також адреса; 1 - по абресі розміщені дані)][30..0(адреса)]] - опис інформаційних даних (information data) idx - де x - це біт 0 або 1
; idx - це адрес на наступний блок адрес, який склдається з 2^k адрес, де k - це кількість біт, які опрацьовуються за один раз,
; або ж, якщо 31 біт встановлений, то idx - це адрес на дані елемента
; Приклад структури зберігання даних
```

; [id0],[id1],[id00],[id01],[id10],[id11],[Element data 0],[Element data 1],[Element data 2],[Element data 3]

; нульва адреса недопускається

; !---------------------!

; ! У програмі потрібно зробити формальну перевірку чи адреса не білеше ніж 2^30,

; оскільки для адреси використовуються лише 31 біт

```
        mov     rcx, LenBufSourceFile
        mov     rsi, hMemSourceFile
        mov     rdi, hMemDistinationFile
        mov     r11, rdi
        xor     rax, rax
        mov     rdx, rax
;                       Shift_n_bytes_1:;n=4*2^k, де k - кількість біт, 4 - дивжина адреси
        mov     dl, 16                  ;запис зсуву на чотири адреси по 4 байти
        cld                                     ;DirectFlag = 0
        ;{ запис адреси для першого елемента
        lodsb                           ;Зчитування байта з hMemSourceFile в al
;                       Shift_n_bytes_2:;n=4*2^k
        mov     byte ptr [rdi+16], al   ;запис зчитаного байта із зсувом на дві адреси (8 байт)
;                       Shift_m_1:;m=8-k
        shr     al, 6                   ;остані k біт потрібно
;                       Multiple_AL_1:  ;AL - address length і становить 4 байти,
          ;80000010h -- 10h = 16 = n=4*2^k
        mov     dword ptr [rdi+4*rax], 80000010h   ; запис абреси у потрібне місце спочатку адреси
```

```
                                                                        ;
останій біт означає, що це адреса даних, а не іншої адреси
        ; xor         rax, 1
        ; mov         dword ptr [rdi+4*rax], 00000000h
        ; Повинен бути запис нуля, якщо пам'ять не виділятиметься з нулями

    ;                                Shift_nAdd1_bytes_1:;n+1=4*2^k+1
        add           rdi, 17                        ; пропуск двох двойних
слів і одного байта, який зчитано і записано
        dec           rcx
        jz            EndCreateMetadata          ;вихід, якщо завершено
опрацювання даних
        ;}
        jmp           NotCarriageReturnSymbol    ; саме така організація для того
щоб у коді нижче був перехід (на CarriageReturnSymbol)
                                                     ; лише коли є символ
кінця, оскільки він рідше зустрічається, а
                                                     ; при переході, коли умова
вірна, виконується зчитування адреси і перехід,
                                                     ; в іншому випадку - ні
    AddresRecorded::;>----------------------------------<
        ; Коли вільна адреса
        mov           rax, r8                    ; зчитування адреси куди буде
записано дані (це кінцева дреса службових даних)
        sub           rax, r11                   ; мінус адреса початку даних для
обчислення зміщення
        or            eax, 80000000h                 ; позначення старшого біта,
що це адреса даних, а не наступної адреси
        mov           [rdi+4*rbx], eax               ; запис адреси куди буде
записано дані
        ;jmp          CarriageReturnSymbol       ; WriteData

    CarriageReturnSymbol:
        mov           rdi, r8
```

```
        mov         rsi, r9                             ; Зчитування адреси початку
зчитування нового елемента для того щоб вкінці після метаданих записати самі дані
        mov         rcx, r10                                ; кількість неопрацьваних
байтів
                                                            ; !!! У edi повинна бути
адреса кінця запису даних
        jmp         NotCarriageReturnSymbol
    ; Спочатку робота з адресами, потім окремо запис даних
    WriteData:                                              ; запис даних
        stosb
    NotCarriageReturnSymbol:
        lodsb
        cmp         al, 13
        loopnz WriteData                                ; вихід, якщо кінець рядка, або
кінець даних
        jnz         EndCreateMetadataAndWriteEndByte        ; якщо усі дані записано

        lodsb                                               ; зчитування 10 символу по
факту це корекція esi
        sub         al, 10                                  ; різниця для перевірки.
Утворений 0 буде записано і лише, якщо не 0, то тоді обнулення (просто оптизація)
        jnz         NotLineFeedSymbol               ; якщо не 10 символ
        dec         rcx
        jz          EndCreateMetadata   ; вихід, якщо завершено опрацювання даних
        jmp         LineFeedSymbol
    NotLineFeedSymbol:
        dec         rsi
        xor         rax, rax        ; mov   al, ah          ; обнулення
    LineFeedSymbol:
        stosb                                               ; запис нульового байта для
позначення кінця даних
    ;-------
        mov    r10, rcx          ; Збереження лічильника опрацьованих байтів вхідних даних
```

mov r9, rsi ; Збереження адреси початку зчитування нового елемента для того щоб вкінці після метаданих записати самі дані

mov r8, rdi ; Збереження поточної адреси кінця запису даних

mov rdi, r11 ; запис початку масиву даних для зчитування адрес, r11 - адреса початку [hMemDistinationFile]

xor rax, rax ; старші байти повинні бути 0 (пов'язано з add rdi, rdx нижче)

CreatingMetadata: ; Цикл опрацювання елемнта

lodsb ; зчитування нового значення для створення масиву із метаданими

cmp al, 13

jz CarriageReturnSymbol ; якщо 13 символ

;{

;rol al, 2 ;! Цей зсув для множення на 4 так як адреси 4 байтні у файлі

; Mov_8_Div_k_1: ;4=8/k, якщо k=2

mov rbp, 4 ; вісм біт/k, якщо k=2, то цикл повториться чотири рази

;rax, rcx, rdx, rbx, rbp

ProcessingByte: ; у цьому циклі робота із власним масивом даних і метаданих

;xor rdx,rdx

; Rol_k_bit_1: ; k=2

rol al, 2 ; для перевірки по k біт (зліва направо)

mov rbx, rax ; копіювання для застосування маски

; Mask_2_pow_k_sub_1_1: ; (2^k-1)=011b, якщо k=2

and bl, 011b ; маска 1100b

mov edx, [rdi+4*rbx] ; зчитування адреси для відповідного біта

test edx, edx ; якщо не ноль значить це адреса наступоного значення

jz AddresRecorded ;---------- ; оскільки для переходу процесор зчитує адресу і переходить

; по ній, а коли не виконується умова, то швидше завершуться виконання

; переходи по адресам частіше будуть ніж запис адреси

;GoToAddress:

xchg rdi, rdx

shl edi, 1 ;1 для обнулення старшого біта для формування чистої адреси і винесення його у CarryFlag для перевірки

jc CompareElements;---------- ; код винесено за межі циклу loop, бо цикл тільки до 128 байт

;ContinueSearch:;----------------------

shr edi, 1 ;2 для обнулення старшого біта для формування чистої адреси

add rdi, r11 ; зміщення + адреса початку пам'яті

; Коли адреса адреси, то ebp потрібен (8/k біт цикл)

dec rbp

jnz ProcessingByte

;}

loop CreatingMetadata

jmp ErrorCoincidence ; якщо опрацьовано всі байти, а різницю не знайдено

CompareElements:;>----------------------------------<

; тепер у edx адреса місця де записано адресу елемента у службових даних,

; тобто та адреса у якої останій 32 біт, встановлений у 1

;{ ebp - можна використовувати, коли адреса значення

mov rsp, r8

sub rsp, r11

;and esp, 07FFFFFFFh Старший біт esp і так буде 0

mov [rdx+4*rbx], esp ; запис адреси кінця запису на місце адреси елемента.

; старший біт повинен бути 0 через те, що далі будуть вказуватись адреси для бітів, які

; одинакові у обох елементів і лише для кінцевих бітів, які відрізняються старший біт -- 1

; далі edx вільний, у edi - адреса елемента, який порівнюється,

; тобто адреса у службових даних

; у esi відповідно адреса у невдсортованому масиві

shr edi, 1 ;2 для обнулення старшого біта для формування чистої адреси другого елемента

mov r12, rdi ; збереження адреси на елемент так як в кінці її потрібно буде записати

add rdi, r11 ; зміщення + адреса початку пам'яті

; корекція адреси для зчитування поточного елемента, який порівнюється

; у rdi адреса елемента, який порівнюється, тобто адреса у службових даних

; у rsi адреса поточного байта елемента, який порівнюється

; у rbx адреса кінця службових даних, куди і будуть доповнюватись адреси а в кінці

; адреса елемнта і він сам, тобто його дані

; у rbp - номер біта, у байті, до якого виконано порівняння поки були адреси переходів

; у rcx кількість неопрацьованих байт (тобто, які залишилось опрцювати) у невідсортованому масиві

; у al недоопрацьваний байт

; rdx наразі вільний

sub rdi, r9 ; r9 це rsi початку елемнта, різниця для того, щоб при сумуванні

; отримати адресу поточного байта другого елемента, того з яким порівнюється перший

add rdi, rsi

;add esi, [esp + 8] ; [esp + 8] esi поточного байта елемента

;sub esi, [esp + 12] ; [esp + 12] esi початку елемнта

```
dec     rdi                     ; бо rsi вказує на наступний
елемент
        ; далі код із порівнянням двох елментів і записом адрес
        xor     rdx, rdx
        dec     rbp
        jz      ContinueCycle           ; якщо опрацьовувався останій біт
        ; кусок коду нижче це корекція байта елемента, зчитаного із службових даних, для
подальшого порівняння
        xchg    rbp, rcx
;                       Shift_k_sub_1_1:        ;1=k-1
        shl     cl, 1
        mov     dl, [rdi]
        rol     al, cl
        xor     dl, al                  ; для порівняння байтів двох елементів
        ror     al, cl
        inc     cl
        stc
        rcr     dx, cl
        ; mov вже не потрібно оскільки cmp   dh, 80 використовує dh, а не ах
        ;mov    dl, 0           ; оскільки зчитаний байт для порівнняння
                        ; може використовуватись не весь, то у не використану
                        ; частину поттрібно обнулити для того щоб
                        ; після зсуву rol       dx, 2 можна було коректно
                        ; використати cmp       dl, 0
        mov     rcx, rbp
        jmp     NotNewByte
BeginCycleWriteAddresses:
        xor     rdx, rdx
        lodsb
        cmp     al, 13
        jz      ErrorCoincidence        ; якщо 13 символ
        inc     rdi                     ; бо зчитування байта
        mov     dl, [rdi]
        cmp     dl, 0           ; 0 символ, оскільки це вже
```

```
                                        ; службовій структурі даних
        jz          ErrorCoincidence
        xor         dl, al                      ; для порівняння байтів двох елементів
        stc
        rcl         dx, 8
    NotNewByte:     ;!!!!!!push    edi                ; esp-4
    BeginWriteAddresses:
        ; використовуються: rax, rcx, rdx, rbp, rsi, rdi
        ; ebx вільний
        xor         rbx, rbx
    ;                                   Rol_k_bit_2:  ; k=2
        rol         al, 2                               ; для перевірки по k біт
(зліва направо)
        mov         bl, al                          ; копіювання для застосування
маски
    ;                                   Mask_2_pow_k_sub_1_2:   ; 11b=3=2^k-1, для
k=2
        and         bl, 011b
        ;mov        rdi, r8
    ;                                   Shift_n_bytes_3:;16=n=4*2^k
        add         r8, 16                      ; чотири 4 байтних адреси
                                        ; r8 - rdi кінцева адреса запису даних
        mov         rbp, r8
        sub         rbp, r11
    ;                                   Shift_n_bytes_4:;16=n=4*2^k
        mov         dword ptr [4*rbx+r8-16], ebp
        ; !!!якщо пам'ять не заповнена нулями, то нульові адрест потрібно записати
        ; xor       bl, 04h
        ; mov       dword ptr [ebx+edi], 0
    ;                                   Rol_k_bit_2:  ; k=2
        rol         dx, 2
        cmp         dl, 0
        ;mov        dl, 0   ; хоч використовується останіх 9 біт,
                            ; але обнулити dl не потрібно, оскільки
```

```
                        ; використовується зсув не менше ніж на 1 біт
        jne             EndWriteAddresses
                                                ;!Вирішено! Починаючи з 0813h байта у
destinationfile
        cmp             dh, 80h         ; перевірка виключно dh, оскільки у dx
                        ; використовується останіх 9 біт, але потім використовується
                        ; зсув не менше ніж на 1 біт, тому dh може бути рівним 0 лише
                        ; тоді коли всі біти опрацьвано, а молодші біти можуть
                        ; бути бідь-які, після восьми бітів які йдуть
                        ; (у спадному порядку) за останім одиничним бітом,
                        ; оскільки dh = 0 перед зсувом rcr        dx, cl
                        ; це означає, що dh завжди рівний 0, тільки коли опрацьовано
                        ; усі потрібні біти
        jne             BeginWriteAddresses;NotTerminalBit
    TerminalBit:    ;!!!!!!pop          edi
    ContinueCycle:
        loop    BeginCycleWriteAddresses
        jmp             CarriageReturnSymbol
    EndWriteAddresses:
    ;                                       ShiftAddress_minus_4_mul_2_pow_k_add_3_1:
    ;-4*2^k+4-1, де 4 довжина адреси
        or              byte ptr [4*rbx+r8-16+3], 080h      ; встановлення біту 31 біта в 1.
                                                ;   +3 через те, що редагування останього з
чотирьох байтів
        xor             bl, dl
        ;pop            ebp                     ; просто корекця вказівника стеку
        or              r12d, 080000000h        ; адреса іншого елемента
        mov             dword ptr [4*rbx+r8-16], r12d
        jmp             CarriageReturnSymbol
    ErrorCoincidence:
        mov             rsp, SaveRSP
        pop     rbp
        pop     r12
        pop     rdi
```

```
pop rsi
pop rbx
mfree hMemDistinationFile ; free
mfree hMemSourceFile ; free
invoke MessageBox, hWin, reparg("Знайдено повторення"),reparg("Помилка"),MB_OK
jmp EndSelect
EndCreateMetadataAndWriteEndByte:
stosb ; запис останього байта !кінцевго елемента!,
; якщо вкінці не було симлу 13
EndCreateMetadata: ; Кінець алгоритму
mov rsp, SaveRSP
xor rax, rax
stosb ; запис нульового байта вкінці, оскільки вихід
; (завершення опрацьвання) виконується перед записом символа 0
sub rdi, r11 ;обчислення довжини даних, які потрібно записати у файл, r11 - адреса початку [hMemDistinationFile]
;замість стеку використовується r11 ;add rsp, 8 ; бо стек наразі містить адресу початку вихідного файлу

;>---------------------------<=====================>---------------------------<
pop rbp
mov flen, rdi
pop r12
pop rdi
pop rsi
pop rbx
sub rsp, 16
mov rcx, rsp
call QueryPerformanceCounter
pop rax
sub TimerInSecond, rax
neg TimerInSecond
mov rcx, rsp
```

```
        call            QueryPerformanceFrequency
        fild            TimerInSecond
        fidiv           dword ptr [rsp]
        fstp            TimerInSecond
        pop             TimerFrequency
        comment*-----------------------------------------------------------------------------
        mov fname, SaveFileDlg(hWin,hInstance,"Збереження файлу",patn)
        cmp BYTE PTR [rax], 0
        jne @F
        return 0
       @@:
        -----------------------------------------------------------------------------*
        invoke exist, DestinationFileName                    ; якщо файл вже існує
        test rax, rax
        jnz @F
                test fldelete( addr DestinationFileName), rax        ; то видалити
        @@:

        mov     flen, rv(save_file, addr DestinationFileName, hMemDistinationFile, flen)
        mfree hMemDistinationFile   ; free
        mfree hMemSourceFile                    ; free
        mov    rax, real8$(TimerInSecond)
        mov str1, ptr$(buffer)
        mov wmsg, cat$(str1,str$(flen)," байт записано на диск", chr$(13), chr$(10), "Час роботи у секундах: ", real8$(TimerInSecond), chr$(13), chr$(10), "Точність вимірювання 1/", str$(TimerFrequency))
        invoke MessageBox, hWin, ptr$(buffer),reparg("Результат запису у файл"),MB_OK
    .case 101     ;IDC_BTN2   Шукати
    comment*---
        mov             str1, ptr$(buffer)
        mov wmsg, cat$(str1,str$(flen)," байт записано на диск", chr$(13), chr$(10), "Час роботи у секундах: ", real8$(TimerInSecond), chr$(13), chr$(10), "Точність вимірювання 1/", str$(TimerFrequency))
        invoke MessageBox, hWin, ptr$(buffer),reparg("Результат запису у файл"),MB_OK
```

!!!!!!!!!!!!!
---------- *

```
;	invoke exist, addr DestinationFileName			; якщо файл вже існує
;	test rax, rax
;	jz EndSelect
	invoke GetWindowTextLength,hEdit1
	cmp		rax, 0
	je		Search_SearchInFile
;shl	rax, 3
	inc		rax
	mov LenBufSourceFile, rax
	mov		hMemSourceFile, alloc(LenBufSourceFile)
	invoke GetWindowText,hEdit1,hMemSourceFile,LenBufSourceFile
;debug 000000014000172C
	mov LenBufSourceFile, rax
	;mov	flen,	rv(save_file,	addr	DestinationFileName,	hMemSourceFile,
LenBufSourceFile)
;	mfree		hMemSourceFile
	jmp		Search_ContinueSearchElement
Search_SearchInFile:
	invoke load_file, ADDR SourceFileName
	mov		hMemSourceFile, rax
;	mov		flenSourceFile, rcx
	mov		LenBufSourceFile, rcx
Search_ContinueSearchElement:
   invoke load_file, ADDR DestinationFileName
   mov hMemDistinationFile, rax
	; Таймер для засікання часу роботи коду
	lea		rcx, TimerInSecond
	call	QueryPerformanceCounter
	push	rbx
	push	rsi
	push	rdi
	push	r12
```

```
		;push	r13
		;push	r14
		;push	r15
		push	rbp
		mov	SaveRSP, rsp
	;>--------------------------------<Пошук>---------------------------------<
		mov		rcx, LenBufSourceFile
		mov		rsi, hMemSourceFile
		mov		rdi, hMemDistinationFile
		mov		r11, rdi
		xor		rax, rax
		mov		rdx, rax
		cld						;DirectFlag = 0 for
lods, stos
	Search_CarriageReturnSymbol:
		mov	r10, rcx	; Збереження лічильника опрацьованих байтів вхідних даних
		mov	r9, rsi		; Збереження адреси початку зчитування нового елемента для
того щоб вкінці після метаданих записати самі дані
		mov	r8, rdi		; Збереження поточної адреси кінця запису даних
		mov		rdi, r11		; запис початку масиву даних для
зчитування адрес, r11 - адреса початку [hMemDistinationFile]
		xor		rax, rax		; старші байти повинні бути 0
(пов'язано з add rdi, rdx нижче)
	Search_CreatingMetadata:		; Цикл опрацювання елемнта
		lodsb					; зчитування нового
значення для створення масиву із метаданими
		cmp	al, 13
		jz		Search_CarriageReturnSymbol		; якщо 13 символ
		;{
		;rol		al, 2		;! Цей зсув для множення на 4 так як адреси 4 байтні у
файлі
	;				Mov_8_Div_k_2:	;4=8/k, якщо k=2
		mov		rbp, 4				; вісм біт/k, якщо k=2, то
цикл повториться чотири рази
```

```
;rax, rcx, rdx, rbx, rbp
Search_ProcessingByte:                                  ; у цьому циклі робота із власним масивом даних і метаданих
;xor            rdx,rdx
;                               Rol_k_bit_2:  ; k=2
rol             al, 2                                   ; для перевірки по k біт (зліва направо)
mov             rbx, rax                                ; копіювання для застосування маски
;                               Mask_2_pow_k_sub_1_3:   ; (2^k-1)=011b, якщо k=2
and             bl, 011b                                ; маска 1100b
mov             edx, [rdi+4*rbx]                        ; зчитування адреси для відповідного біта
test            edx, edx                                ; якщо не ноль значить це адреса наступоного значення
jz              Search_ErrorNotFound            ;----------     ; оскільки для переходу процесор зчитує адресу і переходить
                                                ; по ній, а коли не виконується умова, то швидше завершуться виконання
;GoToAddress:
xchg            rdi, rdx
shl             edi, 1                                  ;1 для обнулення старшого біта для формування чистої адреси і винесення його у CarryFlag для перевірки
jc              Search_CompareElements;----------  ; код винесено за межі циклу loop, бо цикл тільки до 128 байт

;ContinueSearch:;----------------------
shr             edi, 1                                  ;2 для обнулення старшого біта для формування чистої адреси
add             rdi, r11                        ; зміщення + адреса початку пам'яті
; Коли адреса адреси, то ebp потрібен (8/k біт цикл)
dec             rbp
jnz             Search_ProcessingByte
```

```
;}
loop        Search_CreatingMetadata
jmp         Search_ErrorNotFound                    ; коли опрацьовано всі байти, а елемент не знайдено
Search_CompareElements:;>---------------------------------<
            ; тепер у edx адреса місця де записано адресу елемента у службових даних,
; тобто та адреса у якої останій 32 біт, встановлений у 1
;{ ebp - можна використовувати, коли адреса значення
; далі edx вільний, у edi - адреса елемента, який порівнюється,
; тобто адреса у службових даних
; у esi відповідно адреса у невдсортованому масиві
shr         edi, 1              ;2 для обнулення старшого біта для формування чистої адреси другого елемента
mov         r12, rdi            ; збереження адреси на елемент так як в кінці її потрібно буде записати
add         rdi, r11            ; зміщення + адреса початку пам'яті
; корекція адреси для зчитування поточного елемента, який порівнюється
; у rdi адреса елемента, який порівнюється, тобто адреса у службових даних
; у rsi адреса поточного байта елемента, який порівнюється
; у rbx адреса кінця службових даних, куди і будуть доповнюватись адреси а в кінці
; адреса елемнта і він сам, тобто його дані
; у rbp - номер біта, у байті, до якого виконано порівняння поки були адреси переходів
; у rcx кількість неопрацьованих байт (тобто, які залишилось опрцювати) у невідсортованому масиві
; у al недоопрацьваний байт
; rdx наразі вільний
sub         rdi, r9             ; r9 це rsi початку елемнта, різниця для того, щоб при сумуванні
                                ; отримати адресу поточного байта другого елемента, того з яким порівнюється перший

; далі код із порівнянням двох елментів

;xor        rdx, rdx
```

```
        dec         rbp
        jnz         Search_DoNotContinueCycle                  ;              якщо
опрацьовувався останій біт
        loop        Search_ContinueCycle
        jmp         Search_EndSearch
    Search_DoNotContinueCycle:
        dec         rsi                        ; бо rsi вказує на наступний
елемент
    Search_ContinueCycle:
        add         rdi, rsi
        ;Search_SearchInDataStrucure:
        repe   cmpsb ; цикл зупинитиметься у будь-якому випадку вкінці елементів,
                    ; оскільки в них різні символи закінчення 0 - структурі даних
                    ; і 13 - елемента, який шукається або ж його немає але тоді rcx=0
                    ; зупинить цикл
        je          Search_ErrorNotFound_If        ; якщо елементи рівні, тоді ecx = 0
                            ; і кінець опрацювання, тому що елменти хоч можуть
                            ; бути рівні, але їхні кінцеві символи - ні.
                            ; В результаті для перевірки рівності самих елментів
                            ; (без символів кінця) потрібні додаткові перевірки
    ;Search_NotEqualBytes:
        cmp         byte ptr [rsi-1], 13
        jne         Search_ErrorNotFound
        cmp         byte ptr [rdi-1], 0
        jne         Search_ErrorNotFound
        jrcxz       Search_EndSearch
        cmp         byte ptr [rsi], 10
        jne         Search_NotLineFeed
        inc         rsi
        dec         ecx
        jrcxz       Search_EndSearch
    Search_NotLineFeed:
        jmp         Search_CarriageReturnSymbol
```

```
Search_ErrorNotFound_If:
        cmp             byte ptr [rdi], 0
        je              Search_EndSearch
Search_ErrorNotFound:
        mov             rsp, SaveRSP
        pop     rbp
        pop     r12
        pop     rdi
        pop     rsi
        pop     rbx

        mfree hMemDistinationFile   ; free
        mfree hMemSourceFile                    ; free

        invoke MessageBox,                  hWin,                   reparg("Елемент                    не
знайдено"),reparg("Помилка"),MB_OK
        jmp             EndSelect
Search_EndSearch:                                               ; Кінець пошуку
        mov             rsp, SaveRSP
        ;замість стеку використовується r11         ;add                rsp, 8                  ; бо стек
наразі містить адресу початку вихідного файлу
;>-------------------------<=====================>-------------------------<
        pop     rbp
        mov             flen, rdi
        pop     r12
        pop     rdi
        pop     rsi
        pop     rbx

        sub             rsp, 16
        mov             rcx, rsp
        call            QueryPerformanceCounter
        pop             rax
        sub             TimerInSecond, rax
```

```
neg        TimerInSecond
mov        rcx, rsp
call       QueryPerformanceFrequency
fild       TimerInSecond
fidiv      dword ptr [rsp]
fstp       TimerInSecond
pop        TimerFrequency

comment*-----------------------------------------------------------------------------
mov fname, SaveFileDlg(hWin,hInstance,"Збереження файлу",patn)
cmp BYTE PTR [rax], 0
jne @F
return 0
@@:
-----------------------------------------------------------------------------*

mfree hMemDistinationFile   ; free
mfree hMemSourceFile        ; free

mov    rax, real8$(TimerInSecond)
mov str1, ptr$(buffer)
mov wmsg, cat$(str1,"Елемент(-и) знайдено", chr$(13), chr$(10), "Час роботи у секундах: ", real8$(TimerInSecond), chr$(13), chr$(10), "Точність вимірювання 1/", str$(TimerFrequency))
invoke MessageBox, hWin, ptr$(buffer),reparg("Результат пошуку"),MB_OK
        .case 108    ;IDC_BTN3      Write the sorted array
include Write the sorted array.asm
.endsw
.case WM_NOTIFY
mov r11, lParam
mov edx, (NMHDR PTR [r11])._code
.if edx == NM_CLICK
 mov rcx, (NMHDR PTR [r11]).hwndFrom
 .if rcx == hLink1
```

```
                invoke ShellExecute,hWin,"open","www.masm32.com",0,0,SW_SHOW
              .elseif rcx == hLink2
                invoke
ShellExecute,hWin,"open","https://msdn.microsoft.com/library",0,0,SW_SHOW
              .elseif rcx == hLink3
                .data
                  intel  db  "http://www.intel.com/content/www/us/en/processors/architectures-software-
developer-manuals.html"
                .code
                invoke ShellExecute,hWin,"open",ADDR intel,0,0,SW_SHOW
              .endif
            .endif
          .case WM_CLOSE
            exit_dialog:
            invoke EndDialog,hWin,0        ; exit from system menu
        .endsw
    EndSelect:
        xor rax, rax
        ret
    main endp
        end
```