Міністерство освіти і науки України

Одеський національний політехнічний університет

Навчально-науковий інститут комп’ютерних систем

Кафедра системного програмного забезпечення

Нікітченко Максим Ігоревич

студент групи АС-151

**КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА**

Порівнювальний аналіз відомих та запропонованого методів паралельного сортування

Спеціалізація:

Інженерія програмного забезпечення

Керівник:

Паулін Олег Миколайович

докт. техн. наук, доцент

Одеса – 2020

Міністерство освіти і науки України

Одеський національний політехнічний університет

Навчально-науковий інститут комп’ютерних систем

Кафедра системного програмного забезпечення

Рівень вищої освіти: другий (магістерський)

Спеціальність: 121 – Інженерія програмного забезпечення

Спеціалізація: Інженерія програмного забезпечення

**ЗАТВЕРДЖУЮ**

**Завідувач кафедри**

\_\_\_\_\_\_\_\_\_\_\_\_\_ Крісілов В. А.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ р.

# **ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Нікітченко Максима Ігоровича, група АС-151

1. Тема роботи: Порівняльний аналіз відомих та запропонованого методів паралельного сортування

Керівник роботи: Паулін Олег Миколайович, доктор технічних наук, доцент затверджені наказом ректора від «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_ р. № \_\_\_\_\_\_\_

1. Зміст роботи: критичний огляд відомих методів сортування, технічне завдання на розробку, розробка методу, алгоритму та пристрою сортування, проектування і розробка програмного продукту, тестування, встановлення та використання системи, експеримент для визначення параметрів програми
2. Перелік ілюстративного матеріалу: **слайд 1, слайд 2, … слайд N (узгодити із презентацією)**
3. Консультанти розділів роботи

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
| завдання видав | завдання прийняв |
|  | доц. Москалюк А. Ю. |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Дата видачі завдання «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів кваліфікаційної роботи | Строк виконання | Примітка |
| 1 | Критичний огляд відомих методів сортування | 25.09.2020 | Виконав |
| 2 | Технічне завдання на розробку | 05.10.2020 | Виконав |
| 3 | Розробка методу сортування | 18.10.2020 | Виконав |
| 4 | Проектування і розробка програмного продукту | 15.11.2020 | Виконав |
| 5 | Тестування, встановлення та використання системи | 25.11.2020 | Виконав |
| 6 | Експеримент для визначення параметрів програми | 30.11.2020 | Виконав |

**Здобувач вищої освіти**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. І. Нікітченко

**Керівник роботи**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О. М. Паулін

# **ЗАВДАННЯ**

# на розробку розділу «Охорона праці та безпека в надзвичайних ситуаціях»

Нікітченко Максима Ігоровича, група АС-151

Навчально-науковий інститут комп’ютерних систем

Кафедра системного програмного забезпечення

Тема роботи: Порівнювальний аналіз відомих та запропонованого методів паралельного сортування

Зміст розділу:

1. Організація та управління охороною праці
2. Визначення параметрів умов праці
3. Розрахунок заземлення
4. Розрахунок тривалості перебування людей на радіоактивно зараженій місцевості при встановленій дозі опромінення
5. Безпека у суспільстві в умовах загрози COVID-19
6. Заходи з охорони праці та безпеки у надзвичайних ситуаціях

Керівник роботи Консультант з охорони праці та БНС

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О. М. Паулін \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. Ю. Москалюк

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ р. «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ р.

РЕФЕРАТ

Пояснювальна записка до дипломної роботи магістра: 112 с., 32 рис., 2 додаток, 8 джерел.

Предметною областю роботи є алгоритми сортування даних, в перше чергу створені чи оптимізовані для використання на багатопроцесорних пристроях.

Метою роботи є зменшення складності роботи сортувальної мережі.

Методи розробки базуються на мові програмування C# і бібліотеці графічного використання WPF.

В результаті роботи була виконана програмна реалізація алгоритму тасувальної мережі.

Ключові слова: сортувальна мережа, компаратор, послідовність, принцип «кожен з кожним», циклічний зсув, тасувальна мережа, теорема, складність, перестроювання схеми тасування.

**ABSTRACT**

Explanatory note to the master's thesis: 112 p., 32 figs., 2 appendixes, 8 sources.

The subject area of work is data sorting algorithms, primarily created or optimized for use on multiprocessor devices.

The aim of the work is to reduce the complexity of the sorting network.

The development methods are based on the C # programming language and the WPF graphical usage library.

As a result of the work, a software implementation of the shuffling network algorithm was performed.

Key words: sorting network, comparator, sequence, every-with-every principle, cyclical shift, shuffling network, theorem, complexity, rebuilding of the shuffle scheme.

**ЗМІСТ**

ВСТУП 9

1 КРИТИЧНИЙ ОГЛЯД ВІДОМИХ МЕТОДІВ СОРТУВАННЯ 11

1.1 Порівняльний аналіз відомих методів сортування 11

1.1.1 Сортувальна мережу 15

1.1.2 Різновиди сортувальних мереж 19

1.1.3 Паралельний варіант сортування Шелла 21

1.1.4 Паралельний варіант швидкого сортування 21

1.2 Висновки до розділу 23

2 ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ 24

2.1 Постановка завдання розробки методу сортування 24

2.2 Призначення розробки 24

2.3 Вимоги до розроблювального методу та програмної системи 25

2.4 Нефункціональні вимоги 26

2.5 Стадії розробки системи 27

2.6 Вимоги до програмної документації 30

2.7 Технічно-економічні показники 30

2.8 Порядок проведення контролю та якості програмної системи 30

2.9 Технічні вимоги 31

2.10 Висновки до розділу 33

3 РОЗРОБКА МЕТОДУ СОРТУВАННЯ 34

3.1 Розробка методу сортування 34

3.2 Загальні теоретичні відомості щодо регістрів та тригерів 38

3.3 Розробка тасувальної мережі та параметрів 39

3.4 Висновки до розділу 43

4 ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ 44

4.1 Функціональні вимоги 44

4.2 Оцінка тривалості розробки 51

4.3 Розробка плану робіт проекту 57

4.4 Проектування архітектури програмної системи 61

4.5 Детальне логічне проектування 63

4.6 Опис використаних програмних технологій 68

4.7 Опис програмних бібліотек 70

4.8 Висновки до розділу 70

5 ТЕСТУВАННЯ, ВСТАНОВЛЕННЯ ТА ВИКОРИСТАННЯ СИСТЕМИ 71

5.1 Функціональне тестування 71

5.2 Модульне тестування 78

5.3 Розгортання програмного продукту 82

5.4 Інструкція до встановлення 83

5.5 Інструкція до використанню 83

5.6 Висновнки до розділу 88

6 ЕКСПЕРИМЕНТ ДЛЯ ВИЗНАЧЕННЮ ПАРАМЕТРІВ ПРОГРАМИ 89

7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ 92

ВИСНОВКИ 94

СПИСОК ЛІТЕРАТУРИ 95

Додаток А. ПРОГРАМНИЙ КОД СИСТЕМИ 96

Додаток Б. ПИТАННЯ З ОХОРОНИ ПРАЦІ ТА БЕЗПЕКИ У НАДЗВИЧАЙНИХ СИТУАЦІЯХ 101

Б.1 Охорона праці. Організація та управління 101

Б.2 Охорона праці. Визначення параметрів умов праці згідно індивідуального завдання 103

Б.3 Охорона праці. Індивідуальне завдання. Розрахунок заземлення 105

Б.4 Безпека у надзвичайних ситуаціях техногенного характеру. Індивідуальне завдання. Розрахунок тривалості перебування людей на радіоактивно зараженій місцевості при встановленій дозі опромінення 108

Б.5 Безпека у суспільстві в умовах загрози COVID-19 109

Б.6 Заходи з охорони праці та безпеки у надзвичайних ситуаціях 110

Б.7 Висновки 111

**ВСТУП**

У сучасному світі ми часто стикаємося з великими обсягами даних. І серед усього цього величезного масиву нерідко складно буває знайти необхідну інформацію. Тому, щоб уникнути цього і прискорити пошук потрібних даних, використовуються різні методи та алгоритми сортувань.

На сьогоднішній день, сортування є однією з найважливіших процедур обробки структурованої інформації, так як вона використовується практично в будь-якій програмній системі. Сортування характеризується узагальненої операцією «порівняти – переставити». Надалі, будуть розглядатися різні види сортувань по зростанню.

Процес сортування - це упорядкування послідовності чисел за фіксованими правилами, які визначаються методом сортування. Упорядковуються різного роду однорідні об'єкти, наприклад, контейнери різного розміру, що завантажуються в трюм корабля, яким присвоюються цілочислені значення, що називаються ключами. Сортуються саме ключі, тобто вони шикуються в певному порядку.

Основне призначення сортувань – спрощення процесу пошуку об'єкта із заданою властивістю і прискорення цього процесу. Таким чином, найважливішою характеристикою будь-якого алгоритму сортування є швидкість його роботи, що визначається функціональною залежністю середнього часу сортування послідовностей елементів даних певної довжини, від цієї кількості об’єктів послідовності. Час сортування буде пропорційним кількості порівнянь і перестановки елементів даних в процесі їх сортування. Тому дуже важливо постійно покращувати алгоритми для більш ефективної роботи.

В роботі розглядається апаратна реалізація бітонічного сортування. Основним його недоліком є складність схеми, яка визначається двоетапною організацією сортування і надмірним числом компараторів. Необхідність спрощення схеми визначає актуальність розробки нових методів паралельної сортування.

Мета роботи – спрощення схеми паралельного бітонічного сортування за рахунок нового підходу до організації процедури сортування.

Для досягнення цієї мети в роботі вирішуються наступні завдання:

* + - 1. аналіз методу організації бітонічного сортування;

1. розробка нового методу паралельної організації сортування;
2. визначення параметрів нового методу;
3. проведення комп'ютерного експерименту для порівняння часової складності запропонованого методу і відомого методу;
4. розробка програмної системи для реалізації запропонованого методу.

Об'єктом дослідження є процес сортування числових послідовностей.

Предметом дослідження є засоби для розробки та побудови тасувальної мережі.

Для досягнення поставленої мети були використані теорія алгоритмів для опису методів сортувань і розробки тасувальної мережі.

Наукова новизна полягає у тому, що вперше розроблено метод тасування даних, який дозволяє проводити сортування даних за спрощеною схемою. Крім того, отримала подальший розвиток модель сортувальної мережі для її подальшої доробки.

Практична цінність даної роботи полягає в наступному: на основі запропонованого методу тасування розроблена програмна система для його реалізації, що дозволяє сортувати послідовність даних за спрощеною схема.

Публікація: стаття в збірнику Ref. BE13-033 October 14, 2020

**1 КРИТИЧНИЙ ОГЛЯД ВІДОМИХ МЕТОДІВ СОРТУВАННЯ**

**1.1 Порівняльний аналіз відомих методів сортування**

Відомі такі прості методи послідовних сортувань:

* бульбашкове;
* вставками;
* вибором;
* їх модифікації.

Всі вони неефективні при великій довжині послідовності – оцінка часу сортування для них становить T = О (n2). Нижче вони будуть описані трохи детальніше [2].

Бульбашкове сортування – найпростіший алгоритм сортування, який є ефективним тільки для невеликих послідовностей даних. Також варто відзначити, що даний алгоритм певною мірою є навчальним і вкрай рідко використовується в будь-яких реальних проектах. Проте, його принцип лежить в основі таких алгоритмів, як шейкерна, пірамідальна і швидке сортування.

Приклад роботи вказаний на рисунку 1.1.

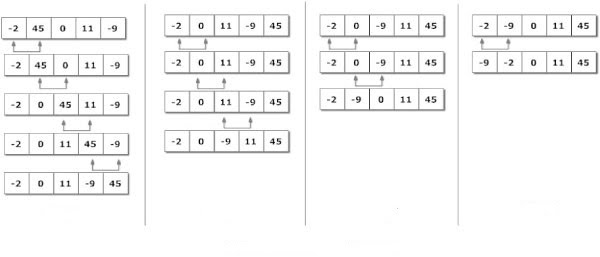


Рисунок 1.1 – Бульбашкове сортування

Сортування вставками – алгоритм сортування, в якому кожен елемент послідовності проглядається і вставляється в відповідне місце серед уже відсортованих раніше даних.

Також даний алгоритм можливо прискорити за рахунок використання бінарного пошуку, який би шукав місце поточного розглянутого елементу в відсортованій послідовності.

Приклад даного виду сортування вказаний на рисунку 1.2.

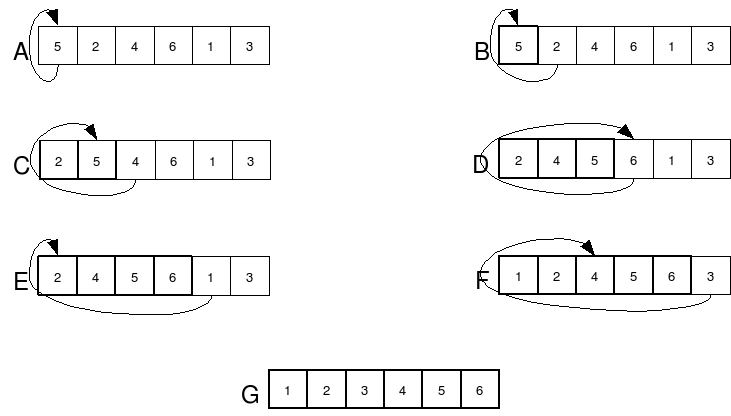


Рисунок 1.2 – Сортування вставками

Сортування вибором – алгоритм сортування шляхом пошуку найменшого елемента в послідовності і обміні цього значення з першим елементом послідовності. Другий мінімум буде ставитися на друге місце і так далі.

Існує також двонаправлений варіант сортування методом вибору, в якому на кожному проході відшукуються і встановлюються на свої місця і мінімальне, і максимальне значення.

Приклад виконання алгоритму вказано на рисунку 1.3.

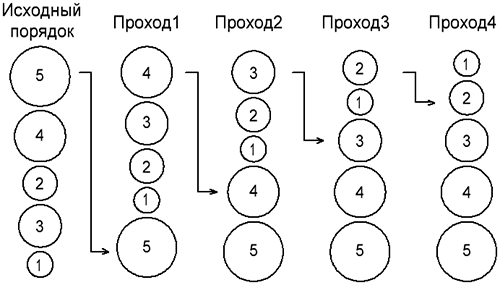


Рисунок 1.3 – Сортування вибором

Нехай n позначає число елементів, що сортуються, а C і M – відповідно кількість необхідних порівнянь ключів і пересилань елементів. Для всіх трьох простих методів сортування Н. Віртом отримані замкнуті аналітичні формули (табл. 1.1). Заголовки стовпців "Максимальні", "Мінімальні" і "Середні" визначають відповідно мінімальні, максимальні і очікувані середні значення для всіх n! перестановок n елементів [1]

Таблиця 1.1 – Замкнуті аналітичні формули

|  |  |  |  |
| --- | --- | --- | --- |
| Метод сортування | Значення | | |
| Мінімальні | Максимальні | Середні |
| Просте включення | C = n-1  M = 2(n-1) | (n2-n)/2-1 (n2+3n-4)/2 | (n2+n-2)/4  (n2-9n-10)/4 |
| Простий вибір | C = (n2-n)/2  M = 3(n-1) | (n2-n)/2  n2/4+3(n-1) | (n2-n)/2| n(ln(n)+0.57) |
| Простий обмін | C = (n2-n)/2  М = 0 | (n2-n)/2  (n2-n)1.5 | (n2-n)/2  (n2- n)0.75 |

Для вдосконалених методів немає достатньо простих і точних формул. Можна тільки сказати, що вартість обчислень дорівнює *c(i)n1.2* в разі сортування Шелла та *c(i)nln (n)* – у випадках швидкого сортування.

Також потрібно відмітити наступні ключові моменти.

1. Перевага сортування бінарними включеннями в порівнянні з сортуванням простими включеннями мала, а в разі вже існуючого порядку взагалі відсутня.
2. Бульбашкове сортування безумовно є найгіршою серед усіх порівнюваних методів. Її поліпшена версія – шейкер-сортування все-таки гірше, ніж сортування простими включеннями і простим вибором (крім виродженого випадку сортування вже сортованого масиву).
3. Швидке сортування перевершує пірамідальну сортування у відношенні 2 до 3. Воно сортує масив з елементами, розташованими в зворотному порядку, практично так само, як вже сортований.

Варто відзначити, що сортування діляться на внутрішні і зовнішні в залежності від виду використовуваної пам'яті, оперативної або дискової. В роботі розглядаються тільки внутрішні сортування.

З метою зменшення накладних затрат на організацію сортувань були розроблені методи з поліпшеними параметрами: сортування Шелла, швидке і інші. Аналіз показав, що прості сортування за своєю організацією (покрокові реалізації узагальненої операції «порівняти - переставити» елементи) є суто послідовними і не піддаються розпаралелюванню. Однак модифіковані методи вже краще пристосовані до розпаралелювання. Відомі 4 варіанти таких сортувань: сортувальні мережі (бітонічне сортування, метод «парне-непарне», сортування Шелла і швидке сортування.

Однак в цих роботах не розглядається реалізація паралельних сортувань на сучасних обчислювальних засобах – на багатоядерних процесорах і при використанні графічних процесорів для неграфічних обчислень.

Нижче будуть розглянуті особливості указаних вище методів паралельного сортування.

* + 1. **Сортувальна мережа**

Сортувальна мережа (СМ) – це пристрій, що містить деяку сукупність компараторів (пристроїв порівняння двох чисел і їх перестановки в порядку зростання) і проводів, що зв'язують компаратори; крім того, є вхідні і вихідні дроти. Як випливає з назви, СМ призначена для сортування довільної послідовності чисел.

Особливістю процедури сортування є те, що порівняння виконуються послідовно по горизонталі і паралельно – по вертикалі. Відзначимо, крім того, що відсортована послідовність буде розташовуватися на виході СС по зростанню зверху вниз [3].

Компаратор - це пристрій з двома входами, *x* і *y*, і двома виходами, *x*' та *y*', яке виконує узагальнену операцію «порівняти - переставити», тобто сортує пару чисел в потрібному порядку:

*x*' = min (*x*, *y*),

*y*' = max (*x*, *y*).

Таким чином, даний пристрій порівнює два вхідних значення і сортує їх у потрібному порядку (як правило, за зростанням).

На рис. 1.4 показаний компаратор [3], який реалізує наведену вище операцію, а також його умовне графічне зображення для випадку найпростішого сортування – сортування пари чисел (7, 3).

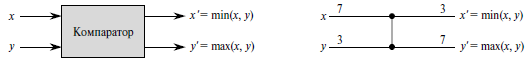


Рисунок 1.4 – Компаратор і його умовне графічне позначення

Схема СС утворюється при з'єднанні виходів всіх даних компараторів з відповідними входами всіх наступних компараторів.

На СС зазвичай реалізується бітонічне сортування, яке передує основне сортування, спрощуючи його; при цьому вхідна послідовність перетвориться в бітонічну.

Бітонічна послідовність – це кінцевий впорядкований набір (кортеж) з дійсних чисел, в якому вони спочатку монотонно зростають, а потім монотонно зменшуються, або набір, який приводиться до такого виду шляхом циклічного зсуву.

Варто відзначити, що будь-яка послідовність, що складається всього з одного числа або двох чисел, є бітонічною. Проте, такі послідовності є свого роду граничним випадком.

В [3] наводиться лема 28.3, сенс якої полягає в тому, що будь-яка частина бітонічної послідовності також є бітонічною.

Бітонічна сортувальна мережа – це мережа компараторів, які коректно сортують бітонічну послідовність нулів і одиниць. Бітонічна сортувальна мережа складається з декількох каскадів, кожен з яких називається полуфільтром.

Таким чином, нуль-одиничну бітонічну послідовність можна відсортувати за допомогою бітонічної сортувальної мережі, глибина якої дорівнює lg(n). За допомогою такої мережі можна впорядкувати будь-яку бітонічну послідовність, що складається з довільних чисел.

Час, який витрачає даний бітонічний сортувальник, дорівнює T = O(n\*log2(n)).

Основна процедура СМ полягає в наступному. Залежно від кількості елементів окремої послідовності проводяться такі маніпуляції:

1. порівнюються симетрично віддалені елементи послідовності;
2. послідовність, що сортується, ділиться на дві частини і в кожній порівнюються мінімальні значення пар;
3. кожна з отриманих послідовностей знову ділиться на дві частини, відбувається порівняння отриманих пар.

На виході отримуємо готову відсортовану послідовність.

Принцип роботи зазначено на рисунку 1.5, який як раз-таки і демонструє сортування вхідної послідовності.

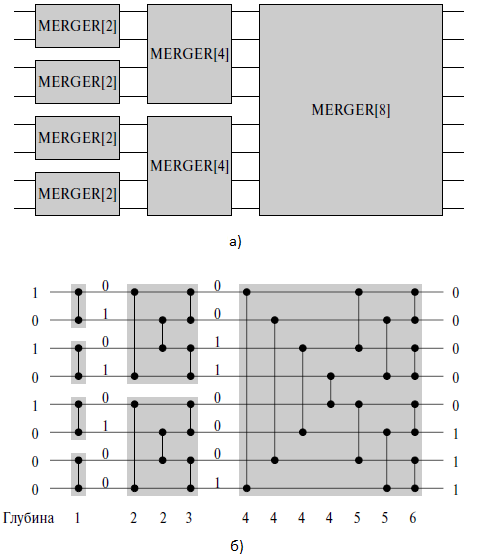


Рисунок 1.5 – Бітонічна сортувальна мережа

В даному алгоритмі дані проходять log(n) етапів, де n – це кількість елементів послідовності. Таким чином, на кожному окремому етапі подається m пар послідовностей, що складаються з рівних кількостей елементів, на вихід з якої повертається об'єднана відсортована послідовність.

На першому етапі пари об'єднуються в четвірки, на другому – четвірки об'єднуються в вісімки, на третьому – в шістнадцятки і так далі. Таким чином, на кожному етапі пари послідовності об'єднуються в одну до тих пір, поки не дійде до цільної відсортованої.

На виході останнього модуля видається одна відсортована послідовність, що містить всі вхідні значення. За індукції можна показати, що така мережа сортує всі послідовності, що складаються з нулів та одиниць, а, отже, відповідно до нуль-одиничного принципу, вона здатна сортувати довільні значення.

У загальному випадку, модуль під номером k, де k = 1,2, ..., log(n), складається з n/2k копій підсистеми Merger [2k], які об'єднують пари 2k-1 елементних відсортованих послідовностей в відсортованій послідовності довжини 2k.

Як приклад принципу роботи СМ при вхідній послідовності з 4-х чисел наведено рисунок 1.6.

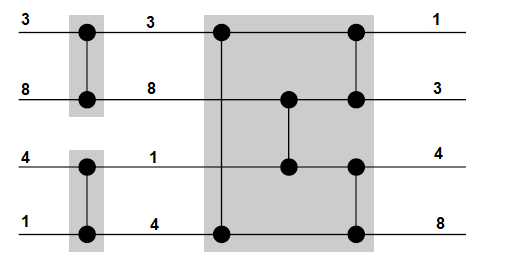


Рисунок 1.6 – Робота сортувальної мережі при послідовності з 4 чисел

Параметри сортувальної мережі складають такі результати: час сортування Т = O(log2(n)), а глибина сети D(n) = lоg(n). Тому даний алгоритм має логарифмічну складність.

Але недоліком такої мережі є досить складна організація процесу сортування.

* + 1. **Різновиди сортувальних мереж**

Особливий інтерес представляє сортувальна мережа, яка реалізує метод «парне-непарне» в порівнянні з однойменною методом, розглянутому в [4]. В даному випадку вхідні послідовність перестають вибудовуватися в бітонічну.

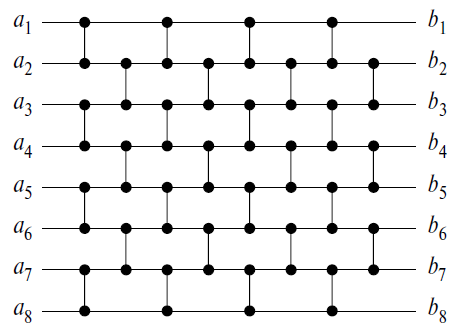


Рисунок 1.7 – Непарна-парна сортувальна мережа на 8 входів

Непарна-парна сортувальна мережа – це транспозиційна сортувальна мережу з *n* рівнями компараторів, які з'єднані між собою за схемою «цегляної кладки». Приклад такої мережі вказано на малюнку 1.7.

Транспозиційна сортувальна мережу – це така мережа, в якій кожен сортувальний пристрій з'єднує сусідні лінії.

Отже, як видно з рисунку, зображеного вище, при *i* = 1, 2, 3, ..., n и d = 1, 2, 3, ..., *n* лінія *i* з'єднана порівняльним пристроєм на глибині *d з* лінією *j* = *i* + (-1)*i*+*d*, якщо 1 ⩽ *j* ⩽ *n*.

Враховуючи розглянуті в першій частині цього розділу види сортувань, також варто звернути на ще один вид сортувальної мережі, яка побудована на методі сортування вставками. Вона зображена на рисунку 1.8.

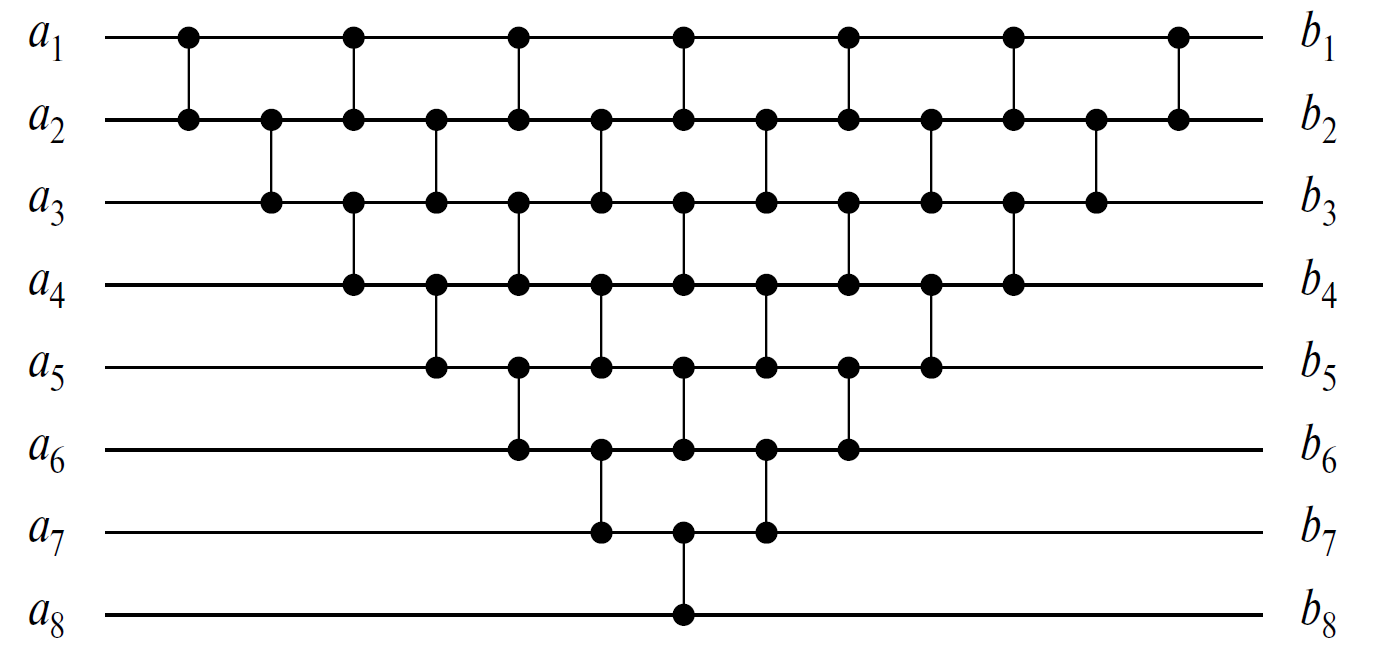


Рисунок 1.8 – Сортувальна мережа, побудована на методі сортування вставками

Дана мережа також являється сортувальною, адже являє собою сукупність компараторів на певну кількість входів на виходів, що поєднуються проводами у певному порядку, визначеному алгоритмом.

Основна схожість полягає в послідовному переміщенні вздовж вхідної послідовності з одночасним обміном елементів у потрібному порядку. Відмінність полягає в тому, що в класичному варіанті сортування вставками прохід відбувається лише один, від першого елементу до останнього, причому лише розглянутий елемент порівнюється з усіма попередніми. А у випадку сортувальної мережі порівняння відбувається усіх сусідніх елементів спершу по зростанню, поки не дійде до останнього елементу, а потім – по зменшенню, поки не дійде до порівняння тільки першої пари чисел.

* + 1. **Паралельний варіант сортування Шелла**

Як і в послідовному варіанті, тут в першу чергу упорядковані пари елементів, які розміщені на певній відстані між собою. Після проходу вздовж всієї послідовності, процес повторюється, але з уже меншою відстанню. Завершується метод при порівнянні/перестановці з кроком 1.

Паралельний варіант складається з двох етапів:

1. При певній ітерації, між собою взаємодіють сусідні процесори в гіперкубі.
2. Відбувається виконання ітерацій паралельного алгоритму парної-непарної перестановки. Виконується аж до отримання відсортованої послідовності на кожному етапі. [3]
   * 1. **Паралельний варіант швидкого сортування**

Даний алгоритм є одним з найшвидших: середня кількість обмінів дорівнює O (n / log (n)) при послідовності в n елементів.

Принцип роботи послідовного варіанта алгоритму наступний:

1. з масиву вибирається деякий опорний елемент a[i];
2. запускається процедура поділу масиву, яка переміщує всі значення, менші або рівні a[i], ліворуч від нього, а все значення, більші або рівні a[i] – вправо;
3. тепер масив складається з двох підмножин, причому ліва менше або дорівнює правій;
4. для обох підмножин: якщо в підмножині більше двох елементів, рекурсивно запускаємо для неї ту ж процедуру.

Приклад роботи методу алгоритму швидкого сортування в його послідовному варіанті вказаний на рисунку 1.9

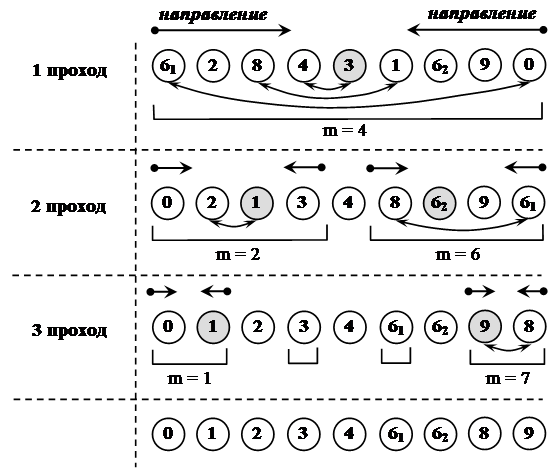


Рисунок 1.9 – Сортування даних швидким сортуванням

У разі паралельної обробки, при якому дані рівномірно розподілені між окремими процесорами, послідовність дій буде наступною:

1. процесор вибирає ключовий елемент, після чого сортує інші елементи щодо нього;
2. менша частина віддається іншому процесору;
3. процедура повторюється при кожному новому розподілі послідовності.

Таким чином, продуктивність буде максимальної, коли всі процесори отримають свою підпослідовність.

**1.2 Висновки до розділу**

У цьому розділі була надана загальна теоретична інформація про методи сортувань даних, зокрема як послідовні варіанти, так і паралельні. Були розглянуті такі види паралельних сортувань як сортувальна мережа та її різновиди, бітонічна сортувальна мережу, сортувальна мережа на основі методу «парна-непарна», сортувальну мережу на основі методу сортування вставками, паралельний варіант сортування Шелла і паралельний варіант швидкого сортування. Для кожної з цих мереж був вказаний принцип роботи алгоритму і параметри, що характеризують ефективність і складність мережі.

2 ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ

**2.1 Постанова завдання на розробку методу сортування**

Постановою для розробки є завдання кафедри системного програмного забезпечення на створення дипломної роботі. Темою роботи була обрана наступна: «Порівнювальний аналіз відомих та запропонованого методів паралельного сортування».

**2.2 Призначення розробки**

Призначення розробки нового методу паралельного сортування полягає в спрощенні схеми цифрового обладнання, оптимізації схеми паралельного бітонічного сортування за рахунок нового підходу до організації процедури сортування та більш раціональному використанні. Схожий принцип якої використовується в сортувальних мережах, а саме – в типі мережі, побудованій на основі використання мережі Sorter[n], що побудована шляхом рекурсивного суміщення об'єднувальних мереж.

Призначенням програмної системи є реалізація тасувальної мережі наряду з іншими найбільш популярними методами сортувальної паралельної обробки інформації, що допоможе використати найбільш підходящий алгоритм в певній ситуації.

Для реалізації програмної системи необхідно:

1. проаналізувати існуючі методи сортування, особливості послідовних та паралельних сортувань, їх параметри та алгоритми дій
2. створити та описати створений алгоритм сортування за аналогією з уже існуючими методами;
3. вибрати необхідні програмні технології, завдяки яким буде можливо впровадити функціональну частину у повному обсязі.

**2.3 Вимоги до розроблювального методу та програмної системи**

При розробці методу сортування даних треба врахувати наступні вимоги:

1. дані повинні оброблюватися рекурсивно;
2. схема цифрового обладнання має бути універсальною для вхідних даних;
3. розроблюваний метод має бути побудованим на основі сортувальної мережіо
4. метод повинен бути в змозі оброблювати дані як послідовно, так і паралельно;
5. складність алгоритму не повинна бути надмірною.

Враховуючи, що окрім розробки методу сортування, буде реалізовано і програмна система, що буде знатна сортувати дані з допомогою тасувальної мережі, необхідно задовольнити наступні вимоги:

1. детальна інформація щодо результатів сортування;
2. можливість збереження вхідну послідовність та результати сортування в окремий тестовий файл;
3. наявність декількох найбільш ефективних методів сортування;
4. можливість перевірки вхідних даних на сторонні символи з подальшим видаленням;
5. наявність можливості використання як послідовного, так і паралельного способів сортування.

Окрім того, інтерфейс має бути зручним, комфортним та дружелюбним по відношенню до користувача, то треба уникнути зайвих нагромаджень елементів інтерфейсу та ретельно спроектувати розміщення всіх ключових деталей інтерфейсу [6]. Також варто використати правило «золотого перетину». Золотий перетин – це пропорційний поділ відрізка на дві нерівні частини, при якому менший відрізок так відноситься до більшого, як більший до всіх. Вважається, що ця пропорція являється найбільш приємною для ока.

**2.4 Нефункціональні вимоги**

Також за стандартом ISO 9126, із зазначеними у ньому характеристиками та їх атрибутами, були сформульовані нефункціональні вимоги для даної програмної системи.

Нефункціональні вимоги - вимоги, що визначають властивості, які система повинна демонструвати, або обмеження, які вона повинна дотримуватися, що не відносяться до поведінки системи.

Нижче приведена таблиця 2.1, в якій, в свою чергу, описуються такі вимоги до системи як функціональність, ефективність, надійність, портативність та супроводжуваність.

Таблиця 2.1 – Нефункціональні вимоги

|  |  |
| --- | --- |
| Характеристика по ISO 9126 | Характеристика атрибутів якості |
| Функціональність | Система здатна сортувати дані, загальним обсягом більше ніж 4096 елементів;  Вихідні дані не можуть бути змінені користувачем, що гарантує їх достовірність. |
| Ефективність | здатна використовувати не більше 1024 МБ оперативної пам’яті;  Час реакції програмної системи на дії користувача не довше 1,2 сек. |
| Надійність | відновлення процесу сортування даних при збої у 93% випадків;  середній час роботи без збоїв – 10 хвилин. |

Продовження таблиці 2.1

|  |  |
| --- | --- |
| Характеристика по ISO 9126 | Характеристика атрибутів якості |
| Портативність | для встановлення достатньо перенести ключові файли на необхідний комп’ютер або запустити спеціальний інсталятор, який автоматично розархівує всі файли, потрібні для роботи;  система здатна працювати як на сімействі операційних систем Windows, так і на операційних системах Ubuntu, починаючи з 15.04. |
| Супроводжуваність | систему легко підтримувати та модернізувати завдяки архітектурі MVC. |

**2.5 Стадії розробки системи**

Нижче, у таблиці 2.2, наведені стадії та час, затрачений на розробку у рамках дипломного проекту.

Таблиця 2.2 – Стадії розробки

|  |  |  |  |
| --- | --- | --- | --- |
| Стадія | Етап | Зміст | Дати виконання |
| Огляд існуючих рішень | Пошук та визначення відомих аналогів в області сортувань даних | Аналіз та визначення особливостей та параметрів існуючих аналогів серед методів сортування даних | 10.09.2020  –  25.09.2020 |

Продовження таблиці 2.2

|  |  |  |  |
| --- | --- | --- | --- |
| Стадія | Етап | Зміст | Дати виконання |
| Технічне завдання | Визначення вимог до створюваних методу та програмної системи | Написання документації, що описує вимоги для програмної системи та методу | 27.09.2020  –  05.10.2020 |
| Технічний проект | Створення методу тасувальної мережі | Розробка нового методу сортування даних на основі оптимального використання компараторів | 06.10.2020  –  18.10.2020 |
| Проектування програмного продукту | Визначення функціональних вимог, визначення строків розробки на основі спеціальних формул, створення необхідних діаграм та опис архітектури | 19.10.2020  –  28.10.2020 |
| Робочий проект | Реалізація програмного продукту | Написання програмного коду до розроблюваної системи, завершення створення технічної документації до програмного продукту | 29.10.2020  –  15.11.2020 |

Продовження таблиці 2.2

|  |  |  |  |
| --- | --- | --- | --- |
| Стадія | Етап | Зміст | Дати виконання |
| Робочий проект | Тестування створеної програмної системи | Перевірка коректності виконання окремих модулів програми та системи в цілому, написання інструкцій з встановлення та використання ПЗ та їх перевірка | 15.11.2020  –  25.11.2020 |
| Науковий експеримент | Проведення наукового експерименту | З метою перевірки ефективності створеної системи проводиться експеримент для встановлення даних | 25.11.2020  –30.11.2020 |
| Охорона праці та безпека у надзвичайних ситуаціях | Огляд ситуації охорони праці на підприємстві | Аналіз окремих складових охорони праці на фірмі, перевірка на наявність необхідних нормативно-правових актів, виконання індивідуальних завдань та рекомендації з покращення умов і безпеки праці, а також щодо вірусу COVID-19 | 30.11.2020  –  10.12.2020 |

**2.6 Вимоги до програмної документації**

До складу документації щодо програмної системи повинна входити наступні дані:

1. опис програми;
2. опис функціональних вимог;
3. опис програмної архітектури;
4. керівництво користувача зі встановлення та викорстання;
5. лістинг програмного коду.
6. Результати тестування

Варто зазначити, що документація може змінюватися в процесі розробки програмної системи

**2.7 Технічно-економічні показники**

Розроблені система та, зокрема, метод можуть бути використаними різними особами з метою сортування даних в реальних чи навчальних задачах, або з метою вивчення та удосконалення.

Система може бути розміщена на будь-якому електронному ресурсі або залишити можливість завантаження на домашні електронно-обчислювальні машини користувачів.

**2.8 Порядок проведення контролю та якості програмної системи**

Після розробки програмної системи до масового впровадження мають бути проведені попереднє тестування та перевірка, досліджувана експлуатація та введення в дію.

Тестування повинно проводитися розробниками на їх робочих місцях, у якості вхідних даних мають бути заздалегідь перевірені послідовності.

Первинна експлуатація має проводитися розробниками ПЗ на робочому місці користувача, в IT-підрозділі фірми. Ціль цієї перевірки – випробовування програмної системи в реальних робочих умовах. Також це забезпечить можливість швидкого та ефективного коректування продукту, опираючись на потреби користувачів.

**2.9 Технічні вимоги**

Технічні характеристики - це опис зразкових характеристик, яким повинен відповідати комп'ютер для того, щоб на ньому могло використовуватися будь-яке визначене програмне забезпечення.

Зазвичай, виконується перевірка таких основних частин як: операційна система, процесор, оперативна пам’ять, відеокарта. Також варто брати до уваги такі характеристики як жорсткий диск, мінімальна роздільна здатність екрану та керування.

Нижче, у таблиці 2.3 наведені такі мінімальні технічні вимоги щодо розроблюваного програмного продукту.

Таблиця 2.3 – Мінімальні технічні вимоги

|  |  |
| --- | --- |
| Складові частини | Технічні вимоги |
| Операційна система | Windows 7 / Windows 8 / Windows 10 / Ubuntu 15.04 |
| Процесор | Мінімальна тактова частота – 1,2 ГГц |
| Відеокарта | Сумісність із DirectX 9 |

Продовження таблиці 2.3

|  |  |
| --- | --- |
| Складові частини | Технічні вимоги |
| Оперативна пам’ять | Для x86: 256 MB ОЗУ  Для x64: 512 ГБ ОЗУ |
| Жорсткий диск | 250 MB вільного місця |
| Мінімальна роздільна здатність екрану | 800x600 |
| Керування | Миша, клавіатура |

Враховуючи, що програма має виконувати паралельні обчислені, тому нижче наведені рекомендовані технічні вимогу.

Таблиця 2.4 – Рекомендовані технічні вимоги

|  |  |
| --- | --- |
| Складові частини | Технічні вимоги |
| Операційна система | Windows 7 / Windows 8 / Windows 10 / Ubuntu 15.04 |
| Процесор | Чотирьох ядерний процесор з мінімальною тактовою частотою не менше 2,0 ГГц |
| Відеокарта | Сумісність із DirectX 9 |
| Оперативна пам’ять | Для x86: 512 MB ОЗУ  Для x64: 1 ГБ ОЗУ |
| Жорсткий диск | 250 MB вільного місця |
| Мінімальна роздільна здатність екрану | 1024x768 |
| Керування | Миша, клавіатура |

**2.10 Висновки до розділу**

В цьому розділі було висунуто технічне завдання на розробку методу сортування даних та програмної системи, що його реалізовує. Зокрема, були проаналізовані підстави та призначення розробки, розглянуті вимоги та етапи та їх час розробки. Були висунуті вимоги до програмної документації та технічні вимоги.

**3 РОЗРОБКА МЕТОДУ СОРТУВАННЯ**

**3.1 Розробка методу сортування**

Нижче розглядається апаратна реалізація запропонованого методу сортування, що виконується рекурсивно.

Пропонований нижче метод сортування полягає в наступному:

1. рекурсивна побудова процедури сортування з використанням компараторів і паралельного тасування;
2. на кожному етапі рекурсії проводиться поділ попередньої послідовності на дві половини і обробка кожної половини паралельного тасування;
3. на останньому етапі рекурсії, коли все підпослідовності містять по 2 елементи, тасування вироджується у впорядкування всіх пар.

Нехай задана послідовність чисел довжиною *n*=2*k*. Необхідно представляти цю послідовність вертикально з першим елементом вгорі. Потрібно впорядкувати задану послідовність по зростанню зверху вниз за допомогою методу паралельного тасування.

Паралельним тасуванням буде називатися процес одночасного попарного сортування елементів верхньої половини (ВП) послідовності з елементами нижньої половини, що здійснюються в n/2 кроків. При цьому порівнюються елементи з номерами *i* та *i+n/2*, *i* = 1..*n*/2-1; у випадку *xi*>*xi+n*/2 ці елементи міняються місцями. При переході до кожного наступного кроку тасування відбувається циклічний зсув нижньої половини послідовності (НП) на одну позицію вгору і далі повторюється попарне сортування всіх елементів половин послідовності. При цьому номери елементів, що з'єднуються в пари, визначаються за правилом: на k-м шаге перший елемент ВП зв’язується із (1+*n*/2+*k*-1=*k*+*n*/2)-м елементом НП, другий елемент ВП зв'язується відповідно з (1+*k*+*n*/2)-м елементом НП и так далі.

Процедура сортування складається з *k* (*k*=log2*n*) етапів тасування, виконуваної рекурсивно, причому на *k*-1 етапах здійснюється тасування, а на *k*-ом етапі виконується впорядковування пар.

Варто врахувати, що циклічний зсув також можна здійснювати вниз.

Як приклад розглянемо тасування послідовності з 8-ми елементів: 6418 5327. Її процедура показана на рис. 3.1.

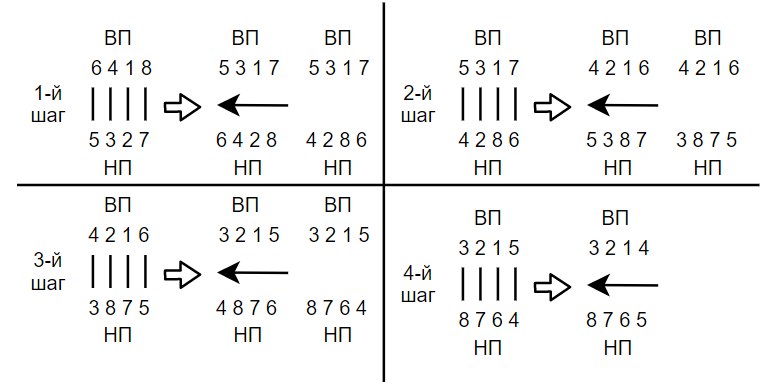


Рисунок 3.1 – Процедура тасування для *n*=8

Розгляд процедури тасування в даному прикладі, а також для прикладу при n = 16 (не приведений в силу громіздкості), показує, що конкретний елемент потрапляє в потрібну половину послідовності не відразу; так, елемент 4 (n = 8) потрапляє в ВП після 4-х переміщень. Аналіз показує, що число переміщень конкретного елемента залежить від того, в яку половину послідовності він потрапить в кінцевому рахунку: «легкі» елементи (по аналогії з бульбашковим сортуванням) можуть переміщатися парне число раз (2, 4, ..., n / 2), а «важкі» - непарне число раз (1, 3, ..., n / 2-1). Варто відзначити, шо в певних випадках переміщень може не бути.

При повному циклічному зсуві здійснюється принцип «кожен з кожним», при якому елемент кожної позиції ВП порівнюється по черзі за n / 2 кроків з елементами на всіх позиціях НП.

Таким чином, можна вивести **теорему**: «В результаті паралельного тасування виконується умова: будь-який елемент верхньої половини послідовності буде менше будь-якого елементу нижньої половини послідовності».

Доказ теореми (реалізованості умови) наведений нижче.

*Необхідність* повного циклічного зсуву. При реалізації принципу «кожен з кожним» будь-який елемент потрапляє в потрібну половину послідовності в гіршому випадку за *n* / 2 переміщень. Це і означає необхідність *n* / 2 кроків тасування для гарантованого виконання умови, наведеного в теоремі.

*Достатність* тасування – додаткові дії не потрібні. Для деякого кроку тасування в результаті одночасного попарного порівняння елементів ВП і НП на всіх позиціях ВП виявляться менші за величиною елементи пари. Однак це не гарантує виконання зазначеного вище умови: в результаті циклічного зсуву елементи ВП можуть виявитися в НП. При повному ж зсуві за *n* / 2 кроків тасування реалізується принцип «кожен з кожним» і навіть в гіршому випадку (*n* / 2 переміщень елемента) елемент виявиться в ВП; інші елементи з меншими значеннями виявляться в ВП навіть швидше. Таким чином, додаткових переміщень елементів не буде потрібно, тобто тасування досить для виконання зазначеного умови.

**Зауваження**. Виконання даної умови в загальному випадку не впорядковує елементи половин послідовності. З цього випливає необхідність подальшої тасування, але вже двох послідовностей половинного розміру.

На рисунку 3.2 представлена структура тасувальної мережі (ТМ), що виконує повне сортування. ТМ будується рекурсивно; при цьому на кожному етапі, починаючи з 1-го, попередні частини вихідної послідовності діляться навпіл і для кожної половини проводиться паралельне тасування.

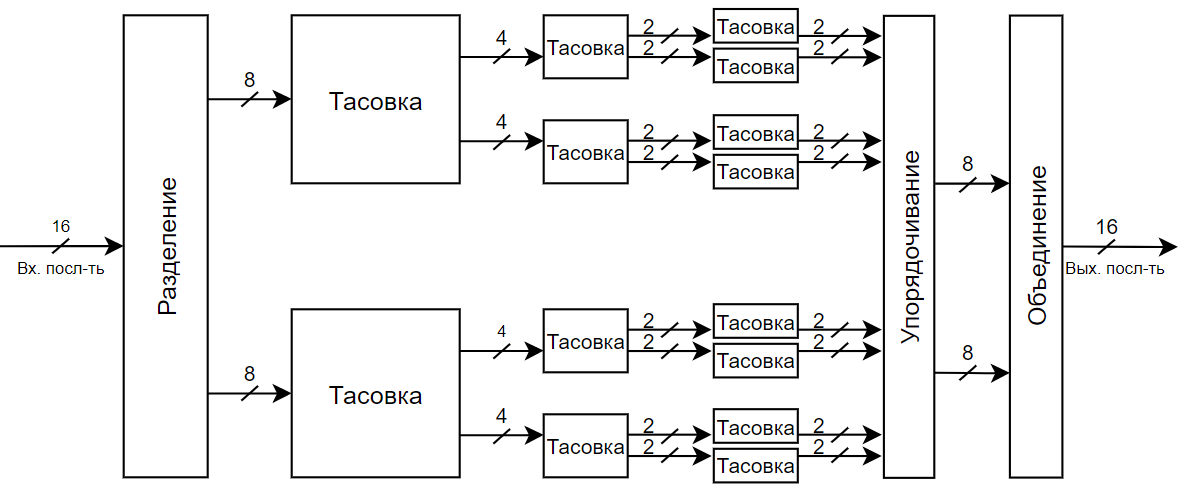


Рисунок 3.2 – Структура тасувальної мережі

Розподіл навпіл здійснюється до тих пір, поки отримані частини вихідної послідовності не складе 2 елементи. Тоді тасування вироджується в просте упорядкування цих пар елементів; таких пар буде n / 2. Після цього всі результати об'єднуються в відсортовану послідовність.

Оцінимо параметри запропонованої тасувальної мережі і сортувальної мережі.

Час обробки заданої послідовності можна оцінити таким виразом:

Т= (*n/2* + *n/4* + … + *n/2 m-1*)*t*, (1)

де t – час спрацьовування компаратору.

Далі винесемо в формулі *n* за дужки. Таким чином, сума в дужках при великих значеннях *m* буде прагнути до 1, отже, Т буде прагнути до *nt*, тобто Т = О (*n*). При цьому часом зсуву частини послідовності в порівнянні з часом тасування нехтуємо.

**3.3 Загальні теоретичні відомості щодо регістрів та тригерів**

Перед тим, як описувати складність обладнання тасувальної мережі, необхідно надати деякі теоретичні відомості стосовно регістрів та тригерів, з яких вони складаються [5].

Регістрами називаються послідовні та паралельні з’єднання тригерів, що виконують функції прийому, зберігання і передачі інформації. Інформація в регістрі зберігається у вигляді двійкового коду, тобто представлена ​​комбінацією сигналів логічного нуля і логічної одиниці. Кожному розряду коду, записаного в регістр, відповідає свій розряд регістра, виконаний, на основі тригерів різних типів.

Тригер – це електронний пристрій, який здатний тривалий час знаходиться в одному із двох станів та міняти їх в результаті впливу зовнішнього сигнала.

Відмінною особливістю тригера як функціонального пристрою є властивість запам'ятовування двійкової інформації. Під пам'яттю тригера мають на увазі здатність залишатися в одному з двох станів і після припинення дії сигналу, що переключає стан тригеру. Прийнявши один зі станів за «1», а інше за «0», можна вважати, що тригер зберігає (пам'ятає) один розряд числа, записаного в двійковому коді.

Синхронні тригери реагують на інформаційні сигнали лише у випадку наявності спеціального сигналу на вході синхронізації С. А самі такі сигнали називають синхронними.

Відзначимо також, що для підвищення швидкості обробки інформації необхідно застосовувати так звані двоступеневі тригери, керовані різними сигналами синхронізації, зокрема, протифазні при одному синхроімпульсі. Це забезпечує роздільне здійснення двох операцій: перезапис інформації з 1-го ступеня у 2-ий, що дозволяє зчитувати інформацію з виходу 2-го ступеня тригера, і одночасний запис у першу сходинку нової інформації.

Відзначимо також, що аналогічними властивостями володіє і весь регістр.

**3.4 Розробка тасувальної мережі та параметрів**

На основі теоретичних даних, описаних у попередньому розділі, далі буде розглянута структура тасувальної мережі з точки зору розміщення в пам’яті ЕВМ. В даному методі використовується обладнання, складається з компараторів і зсувних регістрів (подвійна пам'ять на D-тригерах). D-тригери були обрані завдяки їх властивості зберігати стан входу та передавати його на виході, що необхідно під час виконання операцій порівнянь.

Враховуючи, що за алгоритмом основна послідовність в процесі ділиться на верхню підпослідовність та нижню підпослідовність, це обумовлює наявність саме двох окремих регістрів Також самі обидва регістри являються паралельними, так як кожен з них окремо зберігає масиви даних, рівних половині послідовності, що розглядається на певний момент час.

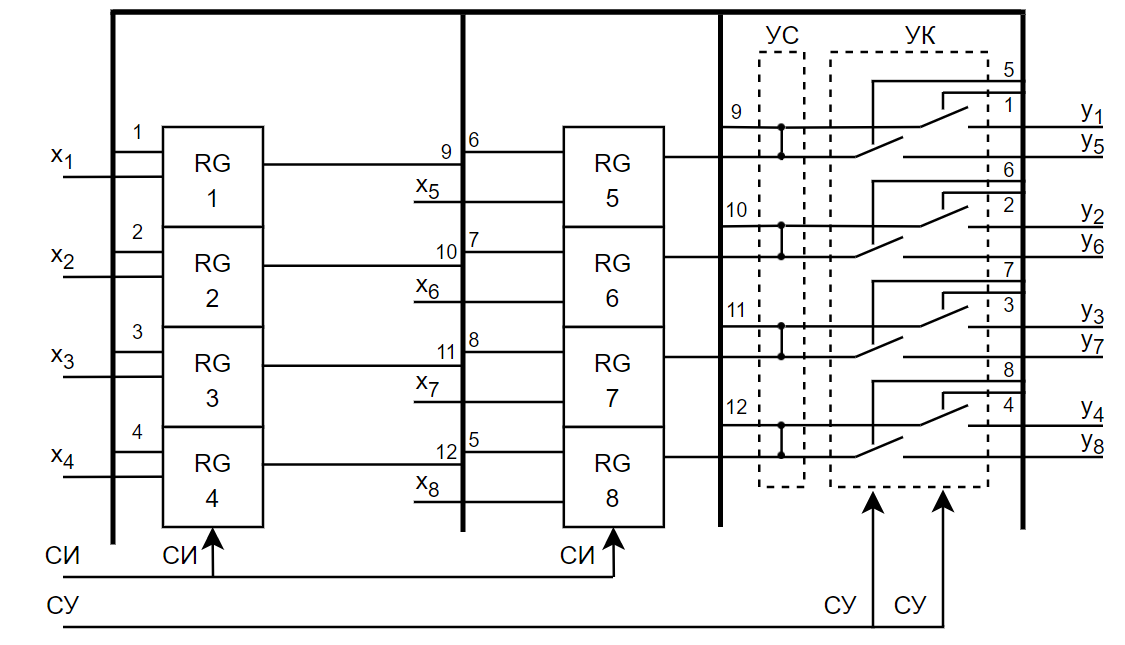


Рисунок 3.3 – Загальна схема тасувальної мережі

Рисунок вище побудований для бітонічної послідовності, що складається з 16-ти елементів. Через шину на вхід кожного регістру в обидвох регістрів подається числовий елемент сортувальної послідовності.. Таким чином, для кожного із регістрів подається окрема частина послідовності. Наприклад, RG 1 зберігає першу комірку пам’яті верхньої половини послідовності, RG 2 зберігає другу комірку пам’яті верхньої половини послідовності, RG 3 зберігає третю комірку пам’яті верхньої половини послідовності і так далі до тих пір, поки кожен елемент послідовності не буде зберігатися у відповідному тригері.

Далі, дані з виходів тригерів передаються на шину та потрапляють до пристрою порівняння, який на схемі позначений як УС. Після порівняння та перестановок у випадку необхідності, дані подаються далі на пристрій комутації. На схемі це позначено як УК. В цьому пристрої визначається, куди необхідно передати дані. Якщо кількості здвигів біло недостатньо ( їх кількість має бути рівною до кількості тригерів у лівому регістрі), то дані передаються на шину і знову потрапляють до регістрів з уже виконаним здвигом за допомогою сихнроімпульсу. У протилежному випадку, подається сигнал керування та, за рахунок цього, відбувається перебудова ліній даних. В результаті, дані подаються далі, на вихід із цієї мережі.

Також варто описати основну перевагу тасувальної мережі перед іншими розглянутими раніше сортувальними мережами – використання того ж самого обладнання під час всієї роботи методу.

Як видно з малюнку 3.3, є регістр, що складається з 4-ох тригерів, вздовж якого відбувається зсув даних. Позначимо точки комутації за індексом змінної, причому для виходу регістрів введемо штрих.

Тоді, в результаті цього, входи і виходи регістрів, що розглядаються як елементи пам'яті для одного даного, пов'язані так (вихід - вхід): 6 ' - 5; 7 ' - 6; 8 ' - 7; 5 ' - 8.

При завершенні етапу тасування та розділені кожної підпослідовності на дві додаткові, вони зберігаються до того ж самого обладнання, але виконується перебудова зв’язку між ними. Таким чином, відбулась наступна зміна: 5' - 6; 6' – 5 (для однієї частини) та 7' - 8; 8' - 7 (для другої частини). Видно, що 5' йде не на 8, а на 6, тобто після 5' слід ставити перемикач з 8 на 6 Аналогічно, 7' йде не на 6, а на 8, тобто і тут треба поставити перемикач, але з 6 на 8.

Ідея скорочення обладнання полягає в заміщенні обладнання даного рівня рекурсії обладнанням наступного рівня рекурсії (при переході до наступного рівню рекурсії). Тобто, використовуємо ті ж регістри, але міняються зв'язки.

Дане скорочення досягається шляхом з’єднанням позицій верхньої половини (ВП) з позиціями в нижньої половини (НП). Таким чином, виконується наступна формула:

*j= i + 2k,* (2)

де i – позиція в ВП,

j – позиція в НП,

k – номер каскаду.

Початкове k дорівнює 1 (при k = 0 каскад є виродженим). Це означає, що обробляються 21 = 2 пари, тобто елементи на позиціях 1-3 та 2-4. Для каскаду, наприклад, 3-го рівня (k = 3), j = i + 23, тобто номера позицій в НП зміщені на 8 щодо відповідних позицій в ВП.

Отже, за рахунок цього досягається ефективність роботи з пам’яттю.

Крім того, існує більш загальний випадок роботи тасувальної мережі.

Основна відмінність – на схемі відсутній чіткий розмір вхідної послідовності. Це може бути будь-який числовий масив, розмір якого кратний ступені двох.

Вхідна послідовність також ділиться на дві половини, кожна з яких за допомогою синхроімпульсу передається до ВП та НП. Далі дані подаються до пристрою порівняння (УС), відбувається обмін, після чого результат подається на вихід цього пристрою. Після цього за допомогою СУ дані або передаються назад до ВП та НП, або передаються на вихід пристрою, де, за необхідністю, рекурсивно подаються до такого ж пристрою, з меншим числом тригерів у регістрах.

Детальний розбір вказаний нижче, на рисунку 3.4.

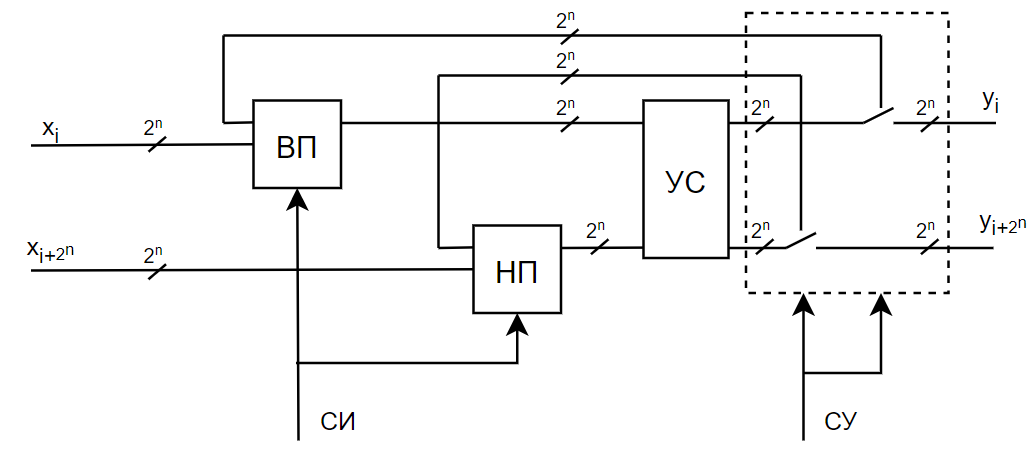


Рисунок 3.4 – Схема тасувальної мережі для числа послідовності, кратного 2n

Як видно, відбувається обробка даних без чіткого вказання довжини послідовності чисел, наприклад, 4 чи 8, як на малюнку 3.3. Натомість, обробка відбувається лише для послідовностей довжиною 2 у ступені n (так як тасувальна мережа підтримує обробка тільки такої довжини чисел). Причому, на вхід потрапляють дві однакові довжини даних, які мають певні індекси для кожного елементи послідовності. Але для верхньої половини це xi, а для нижньої половини – це xi+2^n.

**3.5 Висновки до розділу**

У цьому розділі було надано загальний опис щодо розроблюваного методу тасувальної мережі. Був описаний алгоритм дій при кожному етапі роботи, а також розглянуто на реальному прикладі обробки послідовності. Також була виведена теорема і розглянуті параметри і складність тасувальної мережі. Були детально описані та прокоментовані схеми тасувальної мережі.

**4 ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ**

**4.1 Функціональні вимоги**

Нижче, на рисунку 4.1, вказані функціональні вимоги для створюваної програмної системи. Вони описані за допомогою мови моделювання UML, а саме – за допомогою спеціальної діаграми варіантів використання та сценаріїв, що відображає відносини між акторами (ролі користувачів) і прецедентами (виконувані системою дії), і є складовою частиною моделі прецедентів, що дозволяє описати систему на концептуальному рівні [4].

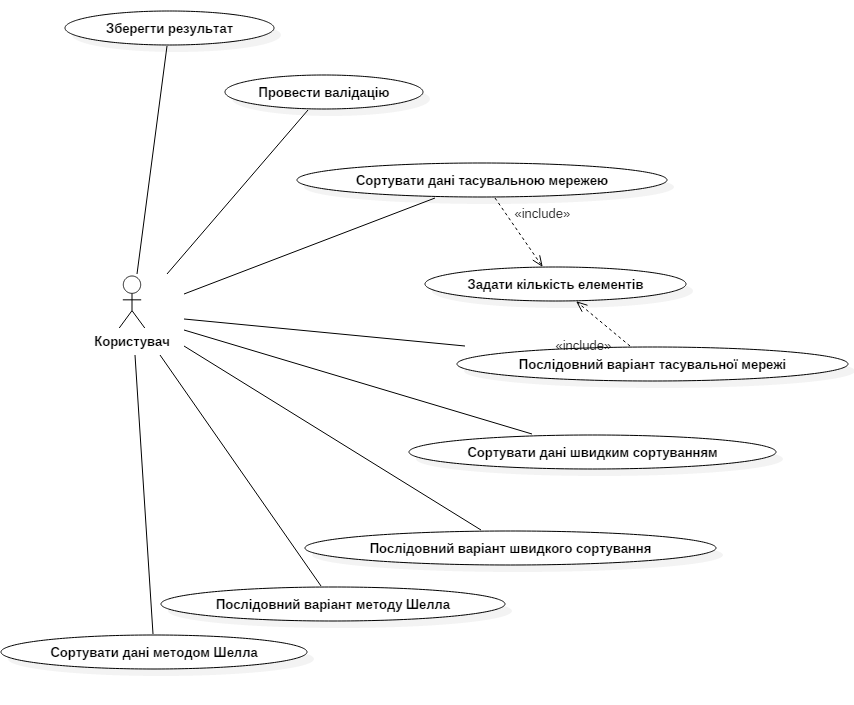


Рисунок 4.1 – Діаграма варіантів використання

Опис акторів наведено нижче.

Користувач – користувач програмою, що отримує можливість використовувати реалізовані методи сортування для вирішення своїх задач, робочих або навчальних.

Опис варіантів використання наведено нижче.

ВВ «Зберегти результат» відповідає за збереження вхідної та вихідної послідовності в окремий тестовий файл.

Мінімальні гарантії: дані не збережені, система продовжую працювати.

Гарантії успіху: дані успішно збережено.

Основний успішний сценарій:

1. Користувач натискає кнопку, яка відповідає за збереження даних, вводить ім’я для файлу та вибирає місце його збереження.

2. Система зберігає файл та сповіщає користувача про успіх операції.

Альтернативні сценарії:

2.1 Вихідні дані відсутні, система повідомляє про помилку.

2.2 Користувач не ввів або ввів некоректне ім’я, система відміняє операцію та повідомляє про помилку.

ВВ «Провести валідацію» відповідає за перевірку коректності вхідної послідовності.

Мінімальні гарантії: дані не коректні, система продовжую працювати.

Гарантії успіху: дані успішно перевірені та скоректовані.

Основний успішний сценарій:

1. Користувач натискає кнопку, яка відповідає за перевірку даних.

2. Система виконує перевірку та коректування вхідних даних у випадку необхідності.

Альтернативні сценарії:

2.1 Вхідні дані відсутні, система повідомляє про помилку.

ВВ «Сортувати дані тасувальною мережею» відповідає за сортування вхідних даних за допомогою розробленого алгоритму тасувальної мережі.

Мінімальні гарантії: дані не коректні або відсутні, система продовжує працювати.

Гарантії успіху: дані успішно відсортовані та виведені на екран. Показано час, затрачений на сортування та кількість елементів.

Основний успішний сценарій:

1. Користувач переходить до вкладки, де знаходяться елементи керування алгоритмом тасувальної мережі, обирає паралельний режим обробки даних, вводить послідовність даних до спеціального поля та натискає на кнопку, щоб почати операцію.

2. Система виконує сортування даних та виводить результат на екран в спеціальну форму, а також затрачений час та кількість елементів.

Альтернативні сценарії:

2.1 Користувач не обрав режим сортування даних, система нагадує користувачу обрати режим.

2.2 Користувач не ввів данні, система повідомляє про пусту вхідну послідовність.

2.3 Користувач ввів некоректні дані, система повідомляє про помилку обробки послідовності.

ВВ «Послідовний варіант тасувальної мережі» відповідає за сортування вхідних даних за допомогою послідовного режиму розробленого алгоритму тасувальної мережі.

Мінімальні гарантії: дані не коректні або відсутні, система продовжує працювати.

Гарантії успіху: дані успішно відсортовані та виведені на екран. Показано час, затрачений на сортування та кількість елементів.

Основний успішний сценарій:

1. Користувач переходить до вкладки, де знаходяться елементи керування алгоритмом тасувальної мережі, обирає послідовний режим обробки даних, вводить послідовність даних до спеціального поля та натискає на кнопку, щоб почати операцію.

2. Система виконує сортування даних та виводить результат на екран в спеціальну форму, а також затрачений час та кількість елементів.

Альтернативні сценарії:

2.1 Користувач не обрав режим сортування даних, система нагадує користувачу обрати режим.

2.2 Користувач не ввів данні, система повідомляє про пусту вхідну послідовність.

2.3 Користувач ввів некоректні дані, система повідомляє про помилку обробки послідовності.

ВВ «Задати кількість елементів» відповідає за вказання кількості елементів, які буде оброблювати розроблений алгоритм тасувальної мережі.

Мінімальні гарантії: кількість не вказані, система продовжую працювати.

Гарантії успіху: кількість елементів успішно обрана.

Основний успішний сценарій:

1. Користувач переходить до вкладки, де знаходяться елементи керування алгоритмом тасувальної мережі та обирає кількість елементів.

2. Система змінює допустиму кількість для вхідної послідовності даного алгоритму.

Альтернативні сценарії:

2.1 Користувач не обрав кількість елементів, система нагадує зробити це.

ВВ «Сортувати дані швидким сортуванням» відповідає за сортування вхідних даних за допомогою швидкого сортування.

Мінімальні гарантії: дані не коректні або відсутні, система продовжує працювати.

Гарантії успіху: дані успішно відсортовані та виведені на екран. Показано час, затрачений на сортування та кількість елементів.

Основний успішний сценарій:

1. Користувач переходить до вкладки, де знаходяться елементи керування алгоритмом швидкого сортування, обирає паралельний режим обробки даних, вводить послідовність даних до спеціального поля та натискає на кнопку, щоб почати операцію.

2. Система виконує сортування даних та виводить результат на екран в спеціальну форму, а також затрачений час та кількість елементів.

Альтернативні сценарії:

2.1 Користувач не обрав режим сортування даних, система нагадує користувачу обрати режим.

2.2 Користувач не ввів данні, система повідомляє про пусту вхідну послідовність.

2.3 Користувач ввів некоректні дані, система повідомляє про помилку обробки послідовності.

ВВ «Послідовний варіант швидкого сортування» відповідає за сортування вхідних даних за допомогою послідовного режиму швидкого сортування.

Мінімальні гарантії: дані не коректні або відсутні, система продовжує працювати.

Гарантії успіху: дані успішно відсортовані та виведені на екран. Показано час, затрачений на сортування та кількість елементів.

Основний успішний сценарій:

1. Користувач переходить до вкладки, де знаходяться елементи керування алгоритмом швидкого сортування, обирає послідовний режим обробки даних, вводить послідовність даних до спеціального поля та натискає на кнопку, щоб почати операцію.

2. Система виконує сортування даних та виводить результат на екран в спеціальну форму, а також затрачений час та кількість елементів.

Альтернативні сценарії:

2.1 Користувач не обрав режим сортування даних, система нагадує користувачу обрати режим.

2.2 Користувач не ввів данні, система повідомляє про пусту вхідну послідовність.

2.3 Користувач ввів некоректні дані, система повідомляє про помилку обробки послідовності.

ВВ «Сортувати дані методом Шелла» відповідає за сортування вхідних даних за допомогою методу Шелла.

Мінімальні гарантії: дані не коректні або відсутні, система продовжує працювати.

Гарантії успіху: дані успішно відсортовані та виведені на екран. Показано час, затрачений на сортування та кількість елементів.

Основний успішний сценарій:

1. Користувач переходить до вкладки, де знаходяться елементи керування алгоритмом методу Шелла, обирає паралельний режим обробки даних, вводить послідовність даних до спеціального поля та натискає на кнопку, щоб почати операцію.

2. Система виконує сортування даних та виводить результат на екран в спеціальну форму, а також затрачений час та кількість елементів.

Альтернативні сценарії:

2.1 Користувач не обрав режим сортування даних, система нагадує користувачу обрати режим.

2.2 Користувач не ввів данні, система повідомляє про пусту вхідну послідовність.

2.3 Користувач ввів некоректні дані, система повідомляє про помилку обробки послідовності.

ВВ «Послідовний варіант методу Шелла» відповідає за сортування вхідних даних за допомогою послідовного режиму методу Шелла.

Мінімальні гарантії: дані не коректні або відсутні, система продовжує працювати.

Гарантії успіху: дані успішно відсортовані та виведені на екран. Показано час, затрачений на сортування та кількість елементів.

Основний успішний сценарій:

1. Користувач переходить до вкладки, де знаходяться елементи керування алгоритмом методу Шелла, обирає послідовний режим обробки даних, вводить послідовність даних до спеціального поля та натискає на кнопку, щоб почати операцію.

2. Система виконує сортування даних та виводить результат на екран в спеціальну форму, а також затрачений час та кількість елементів.

Альтернативні сценарії:

2.1 Користувач не обрав режим сортування даних, система нагадує користувачу обрати режим.

2.2 Користувач не ввів данні, система повідомляє про пусту вхідну послідовність.

2.3 Користувач ввів некоректні дані, система повідомляє про помилку обробки послідовності.

Далі треба встановити пріоритети для кожного ВВ, використовуючи принцип MoSCoW.

MoSCoW - це метод для завдання пріоритету історій в інкрементних і ітераційних методах. Метод MoSCoW забезпечує спосіб досягнення загального розуміння відносної важливості поставляється історії. Всі історії в журналі є цінними, але часто не всі з них можуть бути поставлені в один і той же час.

Таблиця 4.2 – Пріоритети функцій системи

|  |  |
| --- | --- |
| Функція системи | Пріоритет |
| Зберегти результат | C |
| Провести валідацію | S |
| Сортувати дані тасувальною мережею | M |
| Задати кількість елементів | C |
| Послідовний варіант тасувальної мережі | M |
| Сортувати дані швидким сортуванням | M |
| Послідовний варіант швидкого сортування | M |
| Сортувати дані методом Шелла | M |
| Послідовний варіант методу Шелла | M |

Як видно, усі функції, окрім «Зберегти результат», «Задати кількість елементів», «Задати кількість елементів» повинні бути реалізованими.

**4.2 Оцінка тривалості розробки**

В цьому підрозділі буде визначена тривалість розробки за методом UCP (Use Case Points). Він складається з п’яти етапів та дозволяє визначити тривалість розробки в годинах на основі діаграми варіантів використання та сценаріїв.

Перший етап – «Оцінка акторів» – дає оцінку складності інтерфейсів системи. Розрізняють три типи акторів: простий, середній та складний. Простий тип представляє собою зовнішню систему з точно визначеним програмним інтерфейсом. Вага – 1. Середній тип представляє собою зовнішню систему, яка взаємодія по протоколу, подібному до TCP/IP. Вага – 2. Складний тип являє особо, що взаємодії з програмою за допомогою графічного інтерфейсу. Вага – 3.

Актор «Користувач» взаємодіє з системою за допомогою графічного інтерфейсу, тому його тип можна визначити як складний. Отже, вага актора – 3.

Після цього, треба обчислити нескоректовану оцінку акторів (UAW), тобто підрахувати кількість акторів кожного типу та помножити на відповідні вагові категорії.

|  |  |
| --- | --- |
|  | (4.1) |

Другий етап – «Оцінка варіантів використання» – оцінює масштаб системи. Визначимо тип варіантів використання за кількістю неподільних операцій в ньому.

ВВ «Зберегти результат» має 2 транзакції (ініціація варіанту використання, збереження даних).

ВВ «Провести валідацію» має 2 транзакції (ініціація варіанту використання, збереження результату).

ВВ «Сортувати дані тасувальною мережею» має 2 транзакції (ініціація варіанту використання, виведення результату).

ВВ «Послідовний варіант тасувальної мережі» має 2 транзакції (ініціація варіанту використання, виведення результату).

ВВ «Задати кількість елементів» має 2 транзакції (ініціація варіанту використання, збереження результату).

ВВ «Сортувати дані швидким сортуванням» має 2 транзакції (ініціація варіанту використання, виведення результату).

ВВ «Послідовний варіант швидкого сортування» має 2 транзакції (ініціація варіанту використання, виведення результату).

ВВ «Сортувати дані методом Шелла» має 2 транзакції (ініціація варіанту використання, виведення результату).

ВВ «Послідовний варіант методу Шелла» має 2 транзакції (ініціація варіанту використання, виведення результату).

В табл. 4.3 представлено оцінку складності варіантів використання.

Таблиця 4.4 – Оцінка складності варіантів використання

|  |  |  |  |
| --- | --- | --- | --- |
| Варіант використання | Кількість транзакцій | Складність | Вага |
| Зберегти результат | 2 | Простий | 5 |
| Провести валідацію | 2 | Простий | 5 |
| Сортувати дані тасувальною мережею | 2 | Простий | 5 |
| Послідовний варіант тасувальної мережі | 2 | Простий | 5 |
| Задати кількість елементів | 2 | Простий | 5 |
| Сортувати дані швидким сортуванням | 2 | Простий | 5 |
| Послідовний варіант швидкого сортування | 2 | Простий | 5 |
| Сортувати дані методом Шелла | 2 | Простий | 5 |
| Послідовний варіант методу Шелла | 2 | Простий | 5 |

Для обчислення нескоректованої оцінки варіантів використання (UUCW) необхідно кількість варіантів використання кожного типу помножити на відповідні вагові коефіцієнти.

|  |  |
| --- | --- |
|  | (4.2) |

Виконується перша ітерація розробки, тому не потрібно враховувати можливість розділення результатів роботи в команді, так як показник UCP обчислюється як сума показників UAW та UUCW (формула 4.3). Ця оцінка показує складність системи, але без врахування технічних та зовнішніх факторів.

|  |  |
| --- | --- |
|  | (4.3) |

Третій етап – «Оцінка технічних факторів» – необхідний для оцінки складності архітектури системи. Оцінка виконується за шкалою від 0 до 5, де 0 означає відсутність впливу, 3 – середній вплив, 5 – сильний вплив на розробку. Оцінку технічних факторів вказано в табл. 4.4.

Таблиця 4.4 – Оцінка технічних факторів системи

|  |  |  |  |
| --- | --- | --- | --- |
| Фактор | Опис | Вага | Оцінка |
| T1 | Розподіленість системи | 2 | 0 |
| T2 | Час відгуку | 1 | 4 |
| T3 | Ефективність кінцевого користувача | 1 | 3 |
| T4 | Складність обробки | 1 | 1 |
| T5 | Фокус на повторному використанні коду | 1 | 4 |
| T6 | Простота інсталяції | 0,5 | 2 |

Продовження таблиці 4.4

|  |  |  |  |
| --- | --- | --- | --- |
| Фактор | Опис | Вага | Оцінка |
| T7 | Простота використання | 0,5 | 4 |
| T8 | Портативність | 2 | 4 |
| T9 | Простота змін | 1 | 3 |
| T10 | Паралельні обчислення | 1 | 5 |
| T11 | Засоби захисту | 1 | 0 |
| T12 | Доступ до третьої сторони | 1 | 0 |
| T13 | Потреби в спеціальному навчанні | 1 | 1 |

Сума добутків вагових коефіцієнтів та оцінок для кожного з технічних факторів визначає показник TFactor.

|  |  |
| --- | --- |
|  | (4.4) |

Оцінка технічного фактору обчислюється за формулою 4.5.

|  |  |
| --- | --- |
|  | (4.5) |

Четвертий етап – «Оцінка зовнішніх факторів» – дає коефіцієнт для організаційних ризиків при розробці. Оцінка виконується за шкалою від 0 до 5, де 0 означає відсутність впливу, 3 – середній вплив, 5 – сильний вплив на розробку. Оцінку зовнішніх факторів представлено в табл. 4.5.

Таблиця 4.5 – Оцінка зовнішніх факторів системи

|  |  |  |  |
| --- | --- | --- | --- |
| Фактор | Опис | Вага | Оцінка |
| F1 | Знайомство з процесом розробки | 1,5 | 2 |
| F2 | Досвід подібних проектів | 0,5 | 2 |
| F3 | Досвід об’єктно-орієнтованої розробки | 1 | 3 |
| F4 | Досвідченість провідного аналітика | 0,5 | 2 |
| F5 | Мотивація | 1 | 4 |
| F6 | Стабільність вимог | 2 | 4 |
| F7 | Часткова зайнятість працівників | -1 | 3 |
| F8 | Складність мови програмування | -1 | 3 |

Сума добутків вагових коефіцієнтів та оцінок для кожного з зовнішніх факторів визначає показник EFactor.

|  |  |
| --- | --- |
|  | *(4.6)* |

Оцінка зовнішнього фактору обчислюється за формулою 4.7.

|  |  |
| --- | --- |
|  | (4.7) |

П’ятий етап – підраховуються результуючі оцінки.

Скоректовані UCP обчислюються за формулою 4.8.

|  |  |
| --- | --- |
|  | (4.8) |

Для визначення тривалості розробки необхідно з’ясувати, яка кількість робочих годин відповідає одному UCP. Для цього підраховується кількість факторів з множини F1 – F8, оцінки яких за абсолютним значенням перевищують 3. Таких факторів – 1. Результат менше 3, тому приймається, що одному UCP відповідає 20 робочих годин. Підрахуємо тривалість розробки ТUCP:

|  |  |
| --- | --- |
|  | (4.9) |

Отже тривалість розробки дорівнює 768 робочих годин.

**4.3 Розробка плану робіт проекту**

Тепер треба побудувати структурну декомпозицію робіт проекту (WBS - Work Breakdown Structure). Для цього необхідно виділити задачі, які мають бути виконані в проекті та відобразити їх ієрархію. Структурна декомпозиція робіт проекту представлена в таблиці 4.6.

Таблиця 4.6 – Структурна декомпозиція робіт

|  |  |
| --- | --- |
| Назва задачі | WBS |
| Проектування | 1 |
| Проектування архітектури | 1.1 |
| Проектування структури та організації класів | 1.2 |
| Проектування графічного інтерфейсу | 1.3 |
| Реалізація | 2 |
| ВВ «Зберегти результат» | 2.1 |
| ВВ «Провести валідацію» | 2.2 |
| ВВ «Сортувати дані тасувальної мережі» | 2.3 |

Продовження таблиці 4.6

|  |  |
| --- | --- |
| Назва задачі | WBS |
| ВВ «Послідовний варіант тасувальної мережі» | 2.4 |
| ВВ «Задати кількість елементів» | 2.5 |
| ВВ «Сортувати дані швидким сортуванням» | 2.6 |
| ВВ «Послідовний варіант швидкого сортування» | 2.7 |
| ВВ «Сортувати дані методом Шелла» | 2.8 |
| ВВ «Послідовний варіант методу Шелла» | 2.9 |
| Тестування | 3 |
| ВВ «Зберегти результат» | 3.1 |
| ВВ «Провести валідацію» | 3.2 |
| ВВ «Сортувати дані тасувальною мережею» | 3.3 |
| ВВ «Послідовний варіант тасувальної мережі» | 3.4 |
| ВВ «Задати кількість елементів» | 3.5 |
| ВВ «Сортувати дані швидким сортуванням» | 3.6 |
| ВВ «Послідовний варіант швидкого сортування» | 3.7 |
| ВВ «Сортувати дані методом Шелла» | 3.8 |
| ВВ «Послідовний варіант методу Шелла» | 3.9 |
| Інтеграційне тестування | 3.10 |

Для побудови плану проекту визначимо тривалості кінцевих задач та їх залежності один від одної.

Тривалість всього проекту за методом UCP становить 866 годин. Опираючись на стандартний розподіл часу за WBS та власний досвід розподілимо доступні 866 годин між задачами (табл. 4.7).

Таблиця 4.7 – Визначення тривалості окремих задач та залежностей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WBS | Назва задачі | Розподіл часу | Час (год) | Попередники |
| 1 | Проектування | 25% | 217 |  |
| 1.1 | Проектування архітектури | 10% | 86 | 1 |
| 1.2 | Проектування структури та організації класів | 12% | 103 | 1.1 |
| 1.3 | Проектування графічного інтерфейсу | 3% | 28 | 1.2 |
| 2 | Реалізація | 65% | 563 | 1.3 |
| 2.1 | ВВ «Зберегти результат» | 4% | 35 | 2 |
| 2.2 | ВВ «Провести валідацію» | 4% | 33 | 2.1 |
| 2.3 | ВВ «Сортувати дані тасувальною мережею» | 10% | 86 | 2.2 |
| 2.4 | ВВ «Послідовний варіант тасувальної мережі» | 8% | 71 | 2.3 |
| 2.5 | ВВ «Задати кількість елементів» | 3% | 26 | 2.4 |
| 2.6 | ВВ «Сортувати дані швидким сортуванням» | 10% | 86 | 2.5 |
| 2.7 | ВВ «Послідовний варіант швидкого сортування» | 8% | 70 | 2.6 |
| 2.8 | ВВ «Сортувати дані методом Шелла» | 10% | 86 | 2.7 |
| 2.9 | ВВ «Послідовний варіант методу Шелла» | 8 | 70 | 2.8 |
| 3 | Тестування | 10% | 86 | 2.9 |
| 3.1 | ВВ «Зберегти результат» | 1% | 8 | 3 |

Продовження таблиці 4.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WBS | Назва задачі | Розподіл часу | Час (год) | Попередники |
| 3.2 | ВВ «Провести валідацію» | 1% | 8 | 3.1 |
| 3.3 | ВВ «Сортувати дані тасувальною мережею» | 1% | 8 | 3.2 |
| 3.4 | ВВ «Послідовний варіант тасувальної мережі» | 1% | 8 | 3.3 |
| 3.5 | ВВ «Задати кількість елементів» | 1% | 8 | 3.4 |
| 3.6 | ВВ «Сортувати дані швидким сортуванням» | 1% | 8 | 3.5 |
| 3.7 | ВВ «Послідовний варіант швидкого сортування» | 1% | 8 | 3.6 |
| 3.8 | ВВ «Сортувати дані методом Шелла» | 1% | 8 | 3.7 |
| 3.9 | ВВ «Послідовний варіант методу Шелла» | 1% | 8 | 3.8 |
| 3.10 | Інтеграційне тестування | 1% | 14 | 3.9 |

Проектування структури та організації класів може бути виконано тільки після проектування архітектури. Реалізація усіх функцій системи може початися після проектування. Тестування виконується над уже реалізованими функціями системи.

Нижче вказана діаграма Ганта, побудована на основі вхідних даних с таблиці 4.7.

Згідно з цією діаграмою, критичним шляхом буде весь період тестування, який займає близько 86 годин.

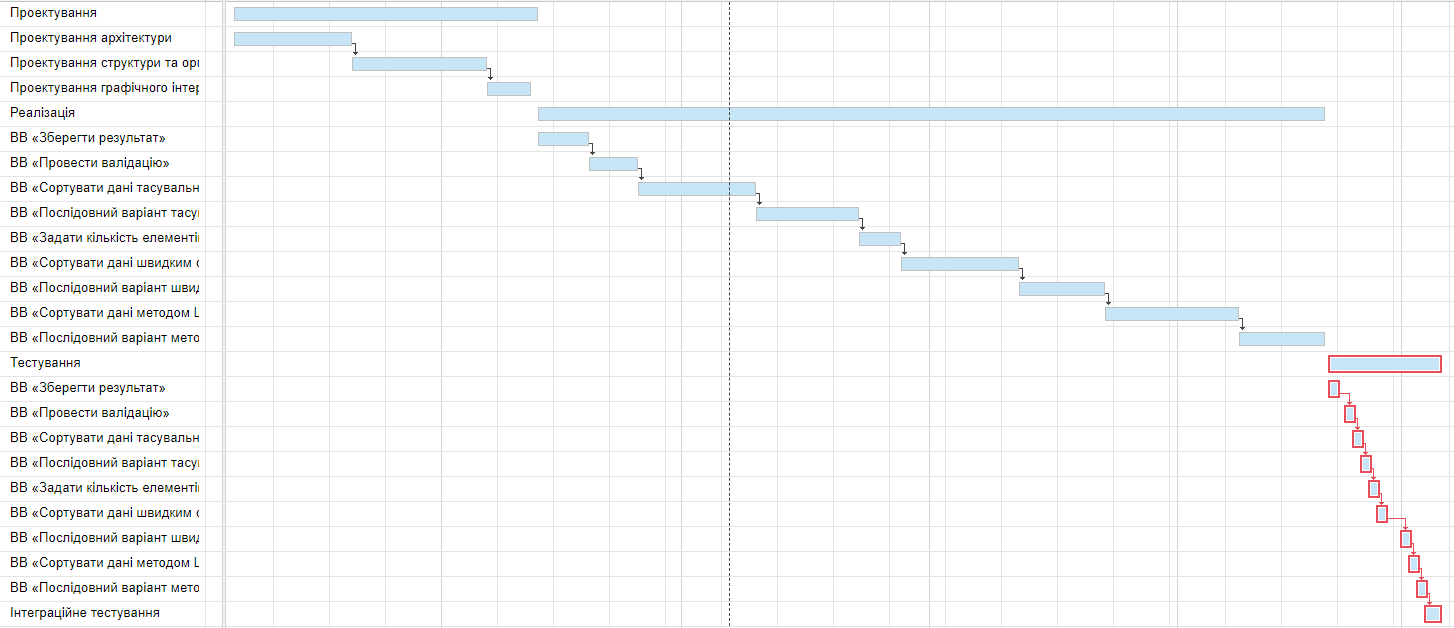


Рисунок 4.2 – Діаграма Ганта

Отже, згідно з діаграмою було обчислено тривалість задач, які необхідно розробити, а також був знайдений критичний шлях.

**4.4 Проектування архітектури програмної системи**

Так як більшість архітектур програмних систем мають схожі набори інтересів, то визначивши контекст системи до неї, можна застосувати певні архітектурні стилі та шаблони проектування. Використання архітектурних стилів полегшує проектування системи, за рахунок вирішення стилем певних проблем та задач проектування, які повторюються від системи до системи.

Для проектування системи використаємо архітектурний стиль Model-View-Controller (MVC). Стиль MVC дозволяє розділити обробку та зберігання даних (Model) від інтерфейсу користувача, який в свою чергу розділяється на відображення інформації (View) та реакцію на дії користувача (Controller). Використання даного шаблону полегшує подальші зміни чи розширення системи.

Користувач завжди взаємодіє з компонентом Controller, який у свою чергу може звертатися до компонента View для зміни графічного інтерфейсу або до компоненту Model, генеруючи йому нову подію.

Компонент Model виконує обробку даних та у випадку змін сповіщає View про необхідність обновлення.

Компонент View при необхідності запитує дані у Model та відображає їх на екрані для користувача.

Архітектура програмної системи зображена нижче, на рисунку 4.3.

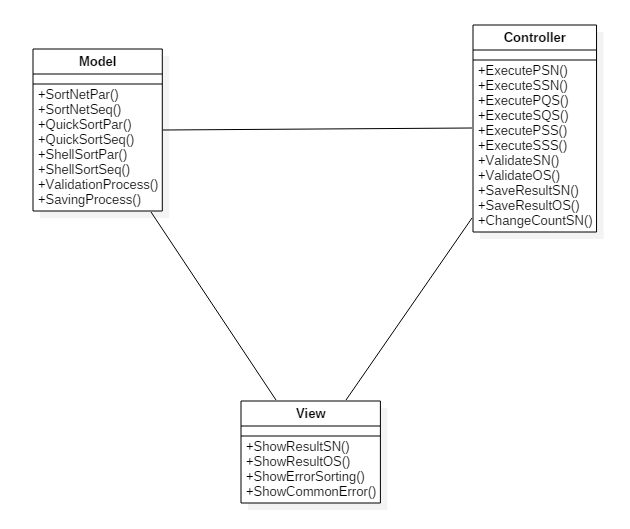


Рисунок 4.3 – Логічне подання архітектури

**4.5 Детальне логічне проектування**

Нижче зображена детальна діаграма класів, на основі якої відбудеться реалізація програмної системи.

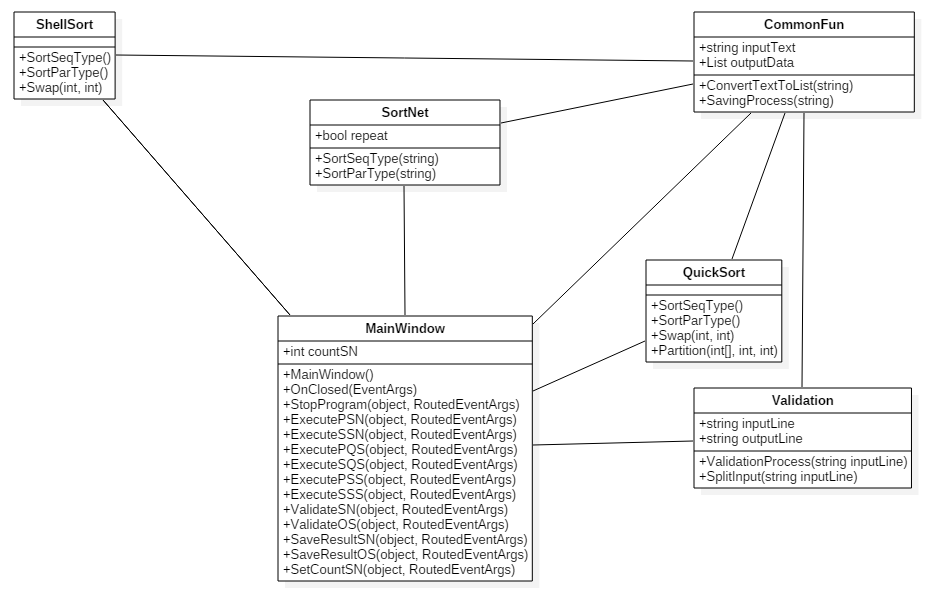


Рисунок 4.4 – Діаграма класів

Опишемо кожен клас більш детально.

MainWindow – клас, що керує основною роботою програми. Складається с наступних методів та полів.

Поле counts – поле, що відповідає за кількість елементів вхідної послідовності для тасувальної мережі.

Метод OnClosed() – функція, що активізується при певних діях користувача та автоматично закриває вікна програми.

Метод StopProgram() – ініціює функцію, яка дає можливість вручну закрити програму.

Метод ExecutePSN() – ініціює функцію, що починає процес сортування даних методом тасувальної мережі в паралельному режимі.

Метод ExecuteSSN() – ініціює функцію, що починає процес сортування даних методом тасувальної мережі в послідовному режимі.

Метод ExecutePQS() – ініціює функцію, що починає процес сортування даних методом швидкого сортування в паралельному режимі.

Метод ExecuteSQS() – ініціює функцію, що починає процес сортування даних методом швидкого сортування в послідовному режимі.

Метод ExecutePSS() – ініціює функцію, що починає процес сортування даних методом сортування Шелла в паралельному режимі.

Метод ExecuteSSS() – ініціює функцію, що починає процес сортування даних методом сортування Шелла в послідовному режимі.

Метод ValidateSN() – ініціює функцію, що починає процес валідації вхідних даних на вкладці «Тасующая сеть».

Метод ValidateOS() – ініціює функцію, що починає процес валідації вхідних даних на вкладці «Другие сортировки».

Метод SaveResultSN() – ініціює функцію, що починає процес збереження вихідних даних на вкладці «Тасующая сеть».

Метод SaveResultOS() – ініціює функцію, що починає процес збереження вихідних даних на вкладці «Другие сортировки».

Метод SetCountSN() – метод, що задає кількість вхідних елементів для тасувальної мережі.

Детальне зображення класу вказано на рисунку 4.5

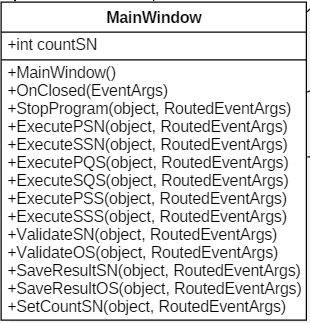


Рисунок 4.5 – Класс MainWindow

Клас SortNet – клас, який відповідає за тасування даних методом тасувальної мережі.

Детальне зображення класу вказано на рисунку 4.6.

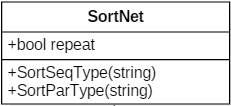


Рисунок 4.6 – Клас SortNet

Поле repeat – відповідає за умову, при якій необхідно повторити процес розділу даних на менші послідовності.

Метод SortSeqType() – функція, що сортує вхідні дані методом сортувальної мережі в послідовному режимі.

Метод SortParType() – функція, що сортує вхідні дані методом сортувальної мережі в паралельному режимі.

Клас QuickSort – клас, який відповідає за тасування даних методом швидкого сортування. Детальне зображення класу вказано на рисунку 4.7.

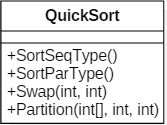


Рисунок 4.7 – Клас QuickSort

Метод SortSeqType() – функція, що сортує вхідні дані методом швидкого сортування в послідовному режимі.

Метод SortParType() – функція, що сортує вхідні дані методом швидкого сортування в паралельному режимі.

Метод Swap() – функція, що переставляє два елементи місцями.

Метод Partition() – функція, що знаходить індекс опорного елементу.

Клас ShellSort – клас, який відповідає за тасування даних методом Шелла.

Детальне зображення класу вказано на рисунку 4.8.

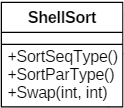


Рисунок 4.8 – Клас ShellSort

Метод SortSeqType() – функція, що сортує вхідні дані методом Шелла в послідовному режимі.

Метод SortParType() – функція, що сортує вхідні дані методом Шелла в паралельному режимі.

Метод Swap() – функція, що переставляє два елементи місцями.

Клас Validation – клас, що відповідає за валідацію вхідних даних. Детальне зображення класу вказано на рисунку 4.9.

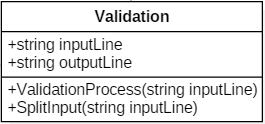


Рисунок 4.9 – Клас Validation

Поле inputLine – зберігає значення вхідних даних до валідації.

Поле outpuLine – зберігає значення вхідних даних після валідації.

Метод Validation() – виконує процес валідації вхідних даних.

Метод SplitInput() – виконує процес розділення вхідних даних на окремі елементи.

Клас CommonFun – клас, що відповідає за загальні функції для деяких інших класів, а також – за збереження даних. Детальне зображення класу вказано на рисунку 4.10.

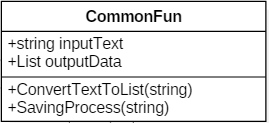


Рисунок 4.10 – Клас CommonFun

Поле inputLine – зберігає значення вхідних даних.

Поле outpuLine – зберігає значення вхідних в потрібному форматі.

Метод ConvertTextToList() – виконує процес перетворення даних.

Метод SavingProcess() – виконує процес збереження вихідних даних.

**4.6 Опис використаних програмних технологій**

Для реалізації програмної системи обрана мова програмування C# [7]. Це строго типізована об’єктно-орієнтована мова програмування з C-подібним синтаксисом, найбільш схожим до таких мов програмування як C++ та Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі у форматі XML. Завдяки цим інструментам, вона дозволяє без зайвих зусиль створювати сучасні програмні системи. Для її використання необхідна встановлена на машині платформа Microsoft .NET Framework. Мова містить багато переваг, завдяки яким вона була обрана мною для реалізації програмної системи. Основними перевагами цієї мови програмування є:

* надійність – мова не містить механізмів, що потенційно призводять до помилок при написані коду: арифметика покажчиків, неявне перетворення типів з втратою точності і т.д.;
* стандартні бібліотеки – багато рутинних задач вирішено в стандартних бібліотеках мови, що дозволяє при написані коду зосередитися на основній задачі;
* збирач сміття – при написані програмного коду не потрібно задумуватись про управління пам’яттю, адже звільнення пам’яті здійснюється автоматично;

У якості середовища розробки для написання коду на мові C# було використано інтегроване середовище розробки Visual Studio. Це сучасний інструмент, що має багато зручних функцій, які полегшують процес написання коду, контролю його версій, тестування, перевірку коректності та швидкий доступ до документації C#. Крім інших переваг цей інструмент має зручний редактор графічних форм, який надає можливість створення графічного інтерфейсу користувача різної складності.

Для реалізації графічного інтерфейсу користувача була використана технологія WPF [8]. Технологія WPF (Windows Presentation Foundation) є частиною екосистеми платформи .NET і являє собою підсистему для побудови графічних інтерфейсів.

Якщо при створенні традиційних додатків на основі WinForms за відрисовку елементів управління і графіки відповідали такі частини ОС Windows, як User32 і GDI+, то додатки WPF засновані на DirectX. У цьому полягає ключова особливість рендеринга графіки в WPF: використовуючи WPF, значна частина роботи по відображенні графіки, як найпростіших кнопок, так і складних 3D-моделей, лягає на графічний процесор на відеокарті, що також дозволяє скористатися апаратним прискоренням графіки.

Однією з важливих особливостей є використання мови декларативної розмітки інтерфейсу XAML, що заснована на XML: це надає можливість створення насиченого графічного інтерфейсу, при цьому використовуючи або декларативне оголошення інтерфейсу, або код на керованих мовах C # і VB.NET, або поєднувати і те, і інше.

Основні переваги WPF:

* використання традиційних мов .NET-платформи - C # і VB.NET для створення логіки додатка;
* можливість декларативного визначення графічного інтерфейсу за допомогою спеціальної мови розмітки XAML, заснованому на XML і представляє альтернативу програмному створення графіки та елементів управління, а також можливість комбінувати XAML і C # / VB.NET;
* незалежність від роздільної здатності екрану: оскільки в WPF всі елементи вимірюються в незалежних від пристрою одиницях, додатки на WPF легко масштабуються під різні екрани з різним дозволом;
* нові можливості, яких складно було досягти в WinForms, наприклад, створення тривимірних моделей, прив'язка даних, використання таких елементів, як стилі, шаблони, теми і інше.

У той же час WPF має певні обмеження. Незважаючи на підтримку тривимірної візуалізації, для створення додатків з великою кількістю тривимірних зображень, перш за все ігор, краще використовувати інші засоби - DirectX або спеціальні фреймворки, такі як Monogame або Unity.

**4.7 Опис програмних бібліотек**

Для створення програмних модулів використовувалася додаткова бібліотека, а саме System.Threading.Tasks – бібліотека, в якій клас Parallel забезпечує паралельну заміну даних на основі бібліотеки для спільних операцій, таких як цикли for, для кожного циклу і виконання набору операторів.

**4.8 Висновки до розділу**

В цьому розділі були розглянуті детальні функціональні вимоги до розроблювальної системи, на основі яких було вирахувано час розробки системи в цілому та час, затрачений на кожен модуль окремо. Окрім того, була побудовано загальна архітектура системи та, на її основи, діаграма класів. І, нарешті, були описані використані програмні технології та бібліотеки

**5 ТЕСТУВАННЯ, ВСТАНОВЛЕННЯ ТА ВИКОРИСТАННЯ СИСТЕМИ**

**5.1 Функціональне тестування**

Перед масовим впровадженням програмної системи, необхідно провести тестування з метою пошуку помилок або інших дефектів та відповідності заявлених до продукту вимог і реально реалізованої функціональності.

Виконаємо функціональне тестування системи, яке необхідне для підтвердження того, що написана програма відповідає заявленим на початку проекту вимогам, тобто виконаємо її верифікацію.

В табл. 5.1 описано тестування прецеденту «Зберегти результат».

Таблиця 5.1 – Тестові варіанти прецеденту «Зберегти результат»

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Натискання кнопки «Сохранить» | Система відображає вікно, де користувач має ввести назву файлу та вибрати місце збереження | Успішно |
| Введення коректних даних | Система проводить процес збереження даних в місце, задане користувачем. | Успішно |
| Введення некоректних даних | Система повідомляє користувача про помилку у вигляді діалогового вікна | Успішно |
| Вхідні дані відсутні | Система повідомляє користувача про відсутні вхідні дані у вигляді діалогового вікна | Успішно |
| Дані збережено | Система повідомляє про успішне завершення операції у вигляді діалогового вікна | Успішно |

В табл. 5.2 описано тестування прецеденту «Провести валідацію».

Таблиця 5.2 – Тестові варіанти прецеденту «Провести валідацію»

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Натискання кнопки «Валидировать» | Система починає перевірку вхідних даних у спеціальному вікні | Успішно |
| Дані перевірені | Система повідомляє про успішне завершення операції у вигляді діалогового вікна | Успішно |
| Дані відсутні | Система повідомляє користувача про відсутні вхідні дані у вигляді діалогового вікна | Успішно |

В табл. 5.3 описано тестування прецеденту «Сортувати дані тасувальною мережею».

Таблиця 5.3 – Тест прецеденту «Сортувати дані тасувальною мережею»

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Натискання кнопки «Сортировать» | Система починає сортування вхідних даних алгоритмом тасувальної мережі, в паралельному режимі | Успішно |
| Дані відсортовано | Система повідомляє про успішне завершення операції у вигляді діалогового вікна та виводить вихідні лані в спеціальному полі | Успішно |

Продовження таблиці 5.3

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Дані не відсортовано | Система повідомляє користувача про помилку у вигляді діалогового вікна, попереджуючи, що вхідна послідовність некоректна | Успішно |
| Вхідні дані відсутні | Система повідомляє користувача про відсутні вхідні дані у вигляді діалогового вікна | Успішно |

В табл. 5.4 описано тестування прецеденту «Послідовний варіант тасувальної мережі».

Таблиця 5.4 – Тестові варіанти прецеденту «Послідовний варіант тасувальної мережі»

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Натискання кнопки «Сортировать» | Система починає сортування вхідних даних алгоритмом тасувальної мережі, в послідовному режимі | Успішно |
| Дані відсортовано | Система повідомляє про успішне завершення операції у вигляді діалогового вікна та виводить вихідні лані в спеціальному полі | Успішно |

Продовження таблиці 5.4

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Дані не відсортовано | Система повідомляє користувача про помилку у вигляді діалогового вікна, попереджуючи, що вхідна послідовність некоректна | Успішно |
| Вхідні дані відсутні | Система повідомляє користувача про відсутні вхідні дані у вигляді діалогового вікна | Успішно |

В табл. 5.5 описано тестування прецеденту «Задати кількість елементів».

Таблиця 5.5 – Тестові варіанти прецеденту «Задати кількість елементів»

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Вибір кількості елементів | Система задає кількість елементів вхідної послідовності для | Успішно |
| Кількість елементів не обрано | Система повідомляє про успішне завершення операції у вигляді діалогового вікна та виводить вихідні лані в спеціальному полі | Успішно |

В табл. 5.6 описано тестування прецеденту «Сортувати дані швидким сортуванням».

Таблиця 5.6 – Тестові варіанти прецеденту «Сортувати дані швидким сортуванням»

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Натискання кнопки «Сортировать» | Система починає сортування вхідних даних алгоритмом швидкого сортування, в паралельному режимі | Успішно |
| Дані відсортовано | Система повідомляє про успішне завершення операції у вигляді діалогового вікна та виводить вихідні лані в спеціальному полі | Успішно |
| Дані не відсортовано | Система повідомляє користувача про помилку у вигляді діалогового вікна, попереджуючи, що вхідна послідовність некоректна | Успішно |
| Вхідні дані відсутні | Система повідомляє користувача про відсутні вхідні дані у вигляді діалогового вікна | Успішно |

В табл. 5.7 описано тестування прецеденту «Послідовний варіант швидкого сортування».

Таблиця 5.7 – Тест прецеденту «Послідовний варіант швидкого сортування»

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Натискання кнопки «Сортировать» | Система починає сортування вхідних даних алгоритмом швидкого сортування, в послідовному режимі | Успішно |

Продовження таблиці 5.7

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Дані відсортовано | Система повідомляє про успішне завершення операції у вигляді діалогового вікна та виводить вихідні лані в спеціальному полі | Успішно |
| Дані не відсортовано | Система повідомляє користувача про помилку у вигляді діалогового вікна, попереджуючи, що вхідна послідовність некоректна | Успішно |
| Вхідні дані відсутні | Система повідомляє користувача про відсутні вхідні дані у вигляді діалогового вікна | Успішно |

В табл. 5.8 описано тестування прецеденту «Сортувати дані методом Шелла».

Таблиця 5.8 – Тест прецеденту «Послідовний варіант швидкого сортування»

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Натискання кнопки «Сортировать» | Система починає сортування вхідних даних алгоритмом сортування Шелла, в паралельному режимі | Успішно |
| Дані відсортовано | Система повідомляє про успішне завершення операції у вигляді діалогового вікна та виводить вихідні лані в спеціальному полі | Успішно |

Продовження таблиці 5.8

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Дані не відсортовано | Система повідомляє користувача про помилку у вигляді діалогового вікна, попереджуючи, що вхідна послідовність некоректна | Успішно |
| Вхідні дані відсутні | Система повідомляє користувача про відсутні вхідні дані у вигляді діалогового вікна | Успішно |

В табл. 5.9 описано тестування прецеденту «Послідовний варіант методу Шелла».

Таблиця 5.9 – Тест прецеденту «Послідовний варіант швидкого сортування»

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Натискання кнопки «Сортировать» | Система починає сортування вхідних даних алгоритмом сортування Шелла, в послідовному режимі | Успішно |
| Дані відсортовано | Система повідомляє про успішне завершення операції у вигляді діалогового вікна та виводить вихідні лані в спеціальному полі | Успішно |
| Дані не відсортовано | Система повідомляє користувача про помилку у вигляді діалогового вікна, попереджуючи, що вхідна послідовність некоректна | Успішно |

Продовження таблиці 5.9

|  |  |  |
| --- | --- | --- |
| Дія | Очікуваний результат | Результат тестування |
| Вхідні дані відсутні | Система повідомляє користувача про відсутні вхідні дані у вигляді діалогового вікна | Успішно |

**5.2 Модульне тестування**

Модульне тестування – процес в програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми, набори з одного або більше програмних модулів разом з відповідними керуючими даними, процедурами використання і обробки.

Нижче буде розглянуті та протестовані деякі функції за допомогою модульного тестування, а саме – білого ящика. Тестування методом білого – метод тестування програмного забезпечення, який передбачає, що внутрішня структура системи відомі особам, що тестує. Вибираються вхідні значення, грунтуючись на знанні коду, який буде їх обробляти. Точно так само відомо, яким повинен бути результат цієї обробки. Знання всіх особливостей програми, що тестується і її реалізації - обов'язкові для цієї техніки.

Також розрізняють тип тестування «чорним ящиком». Тестування методом «чорного ящика», також відоме як тестування, засноване на специфікації або тестування поведінки - техніка тестування, заснована на роботі виключно з зовнішніми інтерфейсами тестованої системи. Тобто особо, що тестує програму, не бачить внутрішньою реалізації, а використовує тільки розробленій графічний інтерфейс.

В табл. 5.10 описано тестування прецеденту «Провести валідацію».

Таблиця 5.10 – Набір тестових сценаріїв для тестування прецеденту «Провести валідацію»

|  |  |  |  |
| --- | --- | --- | --- |
| № тесту | Вхідні дані | Очікуваний результат | Результат тестування |
| 1 | inputTextBoxQS.Text = “1 4 3 text 12 dd 6” | inputTextBoxQS.Text = “1 4 3 12 6”  MessageBox.Show(“Успешно); | Успішно |
| 2 | inputTextBoxQS.Text = “8 12asc3 75 zct12 1 dd 6” | inputTextBoxQS.Text = “8 75 1  6”  MessageBox.Show(“Успешно); | Успішно |
| 3 | inputTextBoxQS.Text = “” | MessageBox.Show(“Данные осутсвуют. Введите их и попробуйте еще раз”); | Успішно |

В табл. 5.11 описано тестування прецеденту «Сортувати дані тасувальною мережею».

Таблиця 5.11 – Набір тестових сценаріїв для тестування прецеденту «Сортувати дані тасувальною мережею»

|  |  |  |  |
| --- | --- | --- | --- |
| № тесту | Вхідні дані | Очікуваний результат | Результат тестування |
| 1 | inputTextBoxTN.Text = “1 4 3 8 7 5 6 2” | outputTextBoxTN.Text = “1 2 3 4 5 6 7 8”  executionTimeTN = “Время выполнения - 0 мс”  countElementsTN = “Количество элементов - 8”  MessageBox.Show(“Успешно!); | Успішно |

Продовження таблиці 5.11

|  |  |  |  |
| --- | --- | --- | --- |
| № тесту | Вхідні дані | Очікуваний результат | Результат тестування |
| 2 | inputTextBoxTN.Text = “1 4 3 8 7 5 6” | outputTextBoxTN.Text = “”  MessageBox.Show(“Неверное количество элементов!”); | Успішно |
| 3 | inputTextBoxTN.Text = “” | outputTextBoxTN.Text = “”  MessageBox.Show(“Входные данные отсутствуют!”); | Успішно |

В табл. 5.12 описано тестування прецеденту «Послідовний варіант тасувальної мережі».

Таблиця 5.12 – Набір тестових сценаріїв для тестування прецеденту «Послідовний варіант тасувальної мережі»

|  |  |  |  |
| --- | --- | --- | --- |
| № тесту | Вхідні дані | Очікуваний результат | Результат тестування |
| 1 | inputTextBoxTN.Text = “1 4 3 8 7 5 6 2” | outputTextBoxTN.Text = “1 2 3 4 5 6 7 8”  executionTimeTN = “Время выполнения - 0 мс”  countElementsTN = “Количество элементов - 8”  MessageBox.Show(“Успешно!); | Успішно |
| 2 | inputTextBoxTN.Text = “1 4 3 8 7 5 6” | outputTextBoxTN.Text = “”  MessageBox.Show(“Неверное количество элементов!”); | Успішно |

Продовження таблиці 5.12

|  |  |  |  |
| --- | --- | --- | --- |
| № тесту | Вхідні дані | Очікуваний результат | Результат тестування |
| 3 | inputTextBoxTN.Text = “” | outputTextBoxTN.Text = “”  MessageBox.Show(“Входные данные отсутствуют!”); | Успішно |

В табл. 5.13 описано тестування прецеденту «Сортувати дані швидким сортуванням».

Таблиця 5.13 – Набір тестових сценаріїв для тестування прецеденту «Сортувати дані швидким сортуванням»

|  |  |  |  |
| --- | --- | --- | --- |
| № тесту | Вхідні дані | Очікуваний результат | Результат тестування |
| 1 | inputTextBoxQS.Text = “7 4 1 2 6 8 9 17 16 11 0” | outputTextBoxQS.Text = “0 1 2 4 6 7 8 9 11 16 17”  executionTimeQS = “Время выполнения - 0 мс”  countElementsQS = “Количество элементов - 11”  MessageBox.Show(“Успешно!); | Успішно |
| 2 | inputTextBoxQS.Text = “” | outputTextBoxQS.Text = “”  MessageBox.Show(“Входные данные отсутствуют!”); | Успішно |

В табл. 5.14 описано тестування прецеденту «Послідовний варіант швидкого сортування».

Таблиця 5.14 – Набір тестових сценаріїв для тестування прецеденту «Послідовний варіант швидкого сортування»

|  |  |  |  |
| --- | --- | --- | --- |
| № тесту | Вхідні дані | Очікуваний результат | Результат тестування |
| 1 | inputTextBoxQS.Text = “7 4 1 2 6 8 9 17 16 11 0” | outputTextBoxQS.Text = “0 1 2 4 6 7 8 9 11 16 17”  executionTimeQS = “Время выполнения - 0 мс”  countElementsQS = “Количество элементов - 11”  MessageBox.Show(“Успешно!); | Успішно |
| 2 | inputTextBoxQS.Text = “” | outputTextBoxQS.Text = “”  MessageBox.Show(“Входные данные отсутствуют!”); | Успішно |

Таким чином, модульне тестування дозволяє ретельно перевірити кожну виконувану умову окремої задачі та, в результаті цього, перевити функцію в цілому, як при правильних варіантах, так і помилкових.

**5.3 Розгортання програмного продукту**

Розгортання програмного забезпечення – це усі дії, що роблять програмну систему готовою до використання. Даний процес є частиною життєвого циклу програмного забезпечення. В цілому процес розгортання складається з декількох взаємопов'язаних дій з можливими переходами між ними.

Нижче будуть описані інструкції, що допоможуть користувачеві встановити та почати використати програмну систему.

**5.4 Інструкція до встановлення**

Для роботи системи необхідна встановлена платформа .Net Framework версії 4.5 або вище. Завантажити платформу можна з офіційного сайту розробника. Посилання на завантаження даної платформи: https://www.microsoft.com/ru-RU/download/details.aspx?id=55167.

Для встановлення системи необхідно перенести виконуваний файл на робочий комп’ютер. Додаткових дій по налаштуванню виконувати непотрібно.

**5.5 Інструкція до використання**

Для початку роботи з системою необхідно запустити виконуваний файл, після чого відкриється головне вікно програми. В ньому присутні дві вкладки: «Тасующая сеть» та «Другие сортировки». Розглянемо їх більш детально.

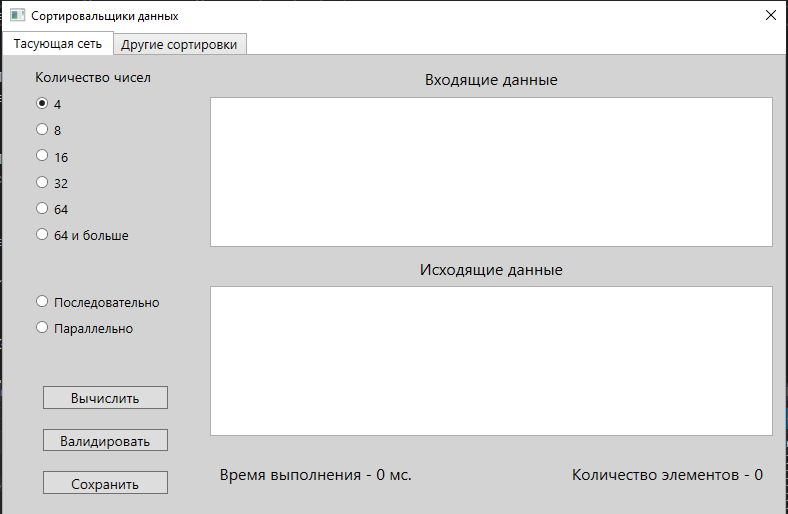


Рисунок 5.1 – Графічний інтерфейс вкладки «Тасующая сеть»

Як видно з малюнку 5.1 вище, вкладка «Тасующая сеть» включає в себе наступні елементи:

* кількість вхідних даних;
* тип сортування;
* поле для вхідних даних;
* поле для вихідних даних;
* кнопки для керування програмою;

Для початку сортування необхідно вибрати кількість елементів, яку прийме тасувальна мережа, обрати режим сортування (послідовний чи паралельний), ввести коректну вхідну послідовність та натиснути кнопку «Вычислить». В результаті, програма виконає необхідні обчислення та виведе результат до поля вихідної послідовності. Приклад виконання вказаний на рисунку 5.2.

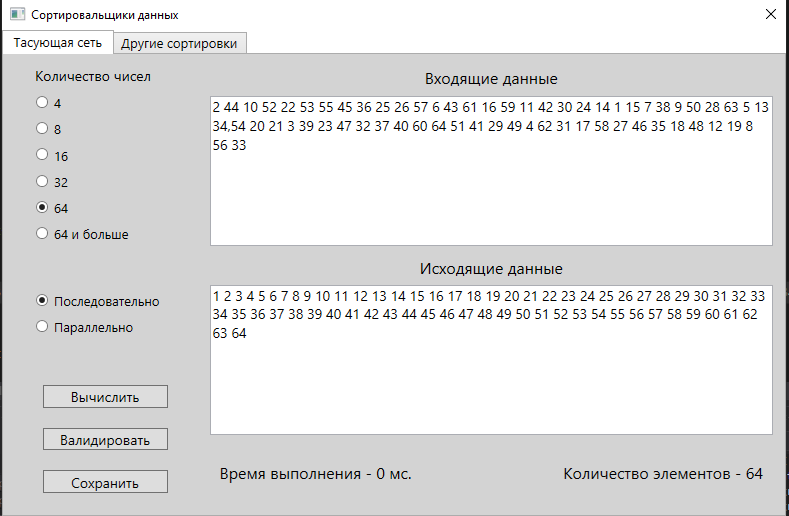


Рисунок 5.2 – Роботи з графічним інтерфейсом вкладки «Тасующая сеть»

Далі треба розглянути вкладку «Другие сортировки». На цій вкладці знаходяться елементи керування сортування методами швидкого сортування та сортуваням Шелла.

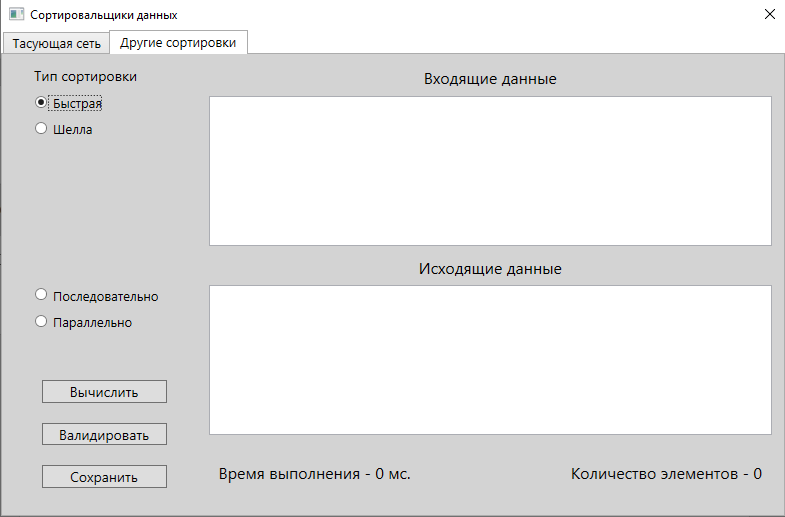


Рисунок 5.3 – Графічний інтерфейс вкладки «Другие сортировки»

Як видно з малюнку 5.3 вище, вкладка «Другие сортировки» подібна до першої вкладки, так як включає в себе наступні елементи:

* тип сортування;
* поле для вхідних даних;
* поле для вихідних даних;
* кнопки для керування програмою;

Для початку сортування необхідно вибрати режим сортування (послідовний чи паралельний), ввести коректну вхідну послідовність та натиснути кнопку «Вычислить». В результаті, програма виконає необхідні обчислення та виведе результат до поля вихідної послідовності. Приклад виконання вказаний на рисунку 5.4.

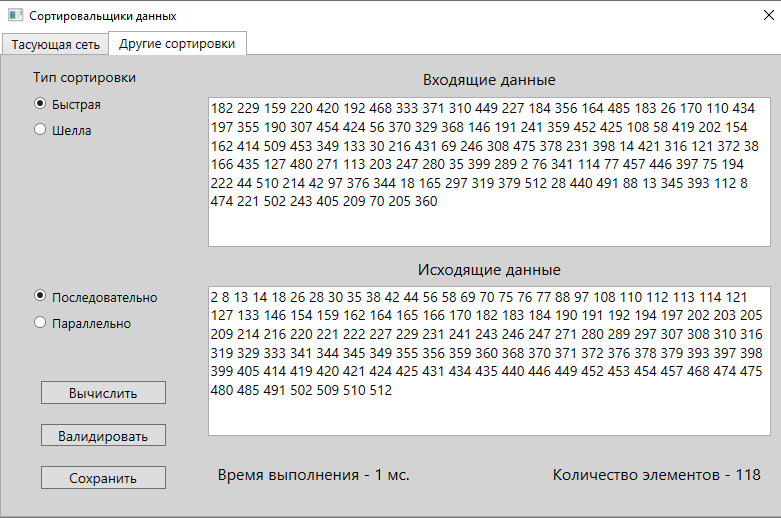


Рисунок 5.4 – Роботи з графічним інтерфейсом вкладки «Другие сортировки»

Крім цього, треба продемонструвати роботи функції валідації. Вона працює однаково для будь-якої вкладки чи методу сортування.

Для її коректної роботи необхідно в поле вхідних даних ввести ці самі дані та натиснути кнопку «Валидировать».

Приклад такої обробки вказаний нижче, на рисунках 5.5 та 5.6.

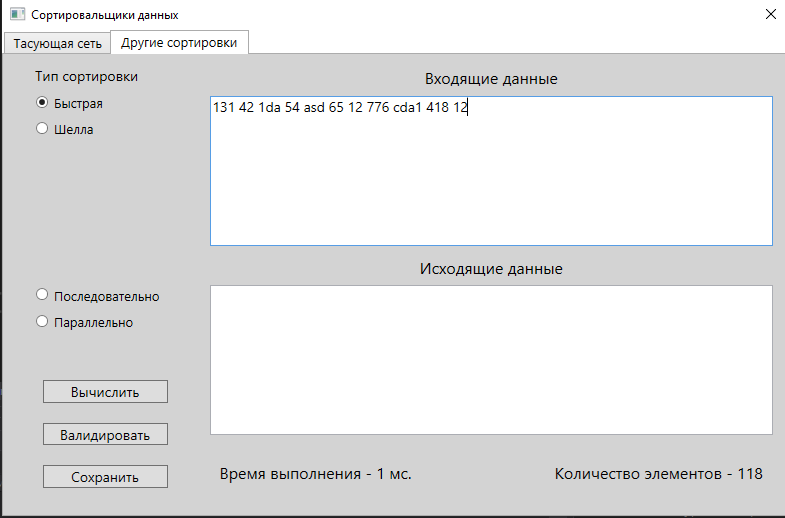


Рисунок 5.5 – Вхідні дані до валідації

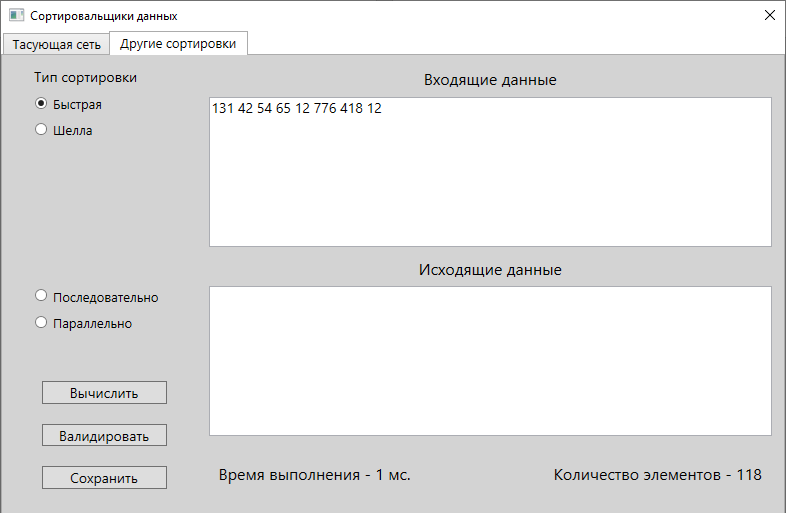


Рисунок 5.6 – Вхідні дані після валідації

Таким чином, можна відмітити, що функція точно валідує вхідні дані, незважаючи на тип сортування.

**5.6 Висновки до розділу**

В даному розділі було проведено тестування, а саме – функціональне та модульне. Окрім того, для зручності користувача біли створенні інструкції зі встановлення та використання.

**6 ЕКСПЕРИМЕНТ ДЛЯ ВИЗНАЧЕННЮ ПАРАМЕТРІВ ПРОГРАМИ**

Завершальним штрихом реалізації алгоритму та програмного забезпечення, заснованого на ньому, є проведення експерименту для визначення ефективності новоствореного методу. Для цього необхідно порівняти часові характеристики при сортування однакових послідовностей різної довжини.

Будуть розглянуті паралельні обчислення даних при різних довжинах вхідних послідовностей, зокрема, при 256 елементів, при 2048 елементів та при 4096 елементів.

Спершу буде порівняна послідовність довжиною в 256 елементів.

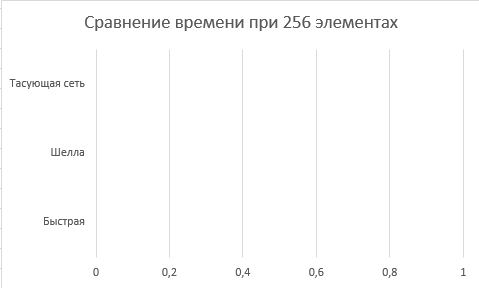


Рисунок 6.1 – Порівняння часу при 256 елементів

Як видно, дані обчислення являються незначними для будь-якого методу сортування, тому час, затрачений на них – менший, ніж 0 мс

Далі – при 2048 елементів.

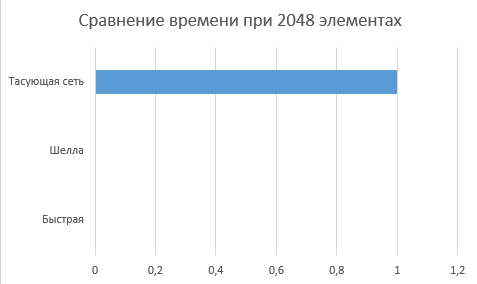


Рисунок 6.2 – Порівняння часу при 2048 елементів

Для обчислення даного типу послідовності тасувальній мережі знадобилось більше часу, ніж двом іншим, але не набагато більше.

Далі – при 4096 елементів.

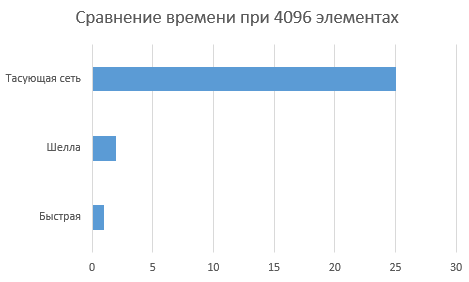


Рисунок 6.3 – Порівняння часу при 4096 елементів

Отже, таким чином, в порівнянні з іншими методами, тасувальна мережа не являється ефективною, так як існують інші методи, що більше підходять для виконання обчислень з точки зору затраченого часу. Але, як описувалось в розділі 3, даний метод виконує більш ефективну роботу з точки зору цифрового обладнання, тому можна вважати, що в даному сенсі він має більші переваги, ніж його конкуренти.

**7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ**

У будь-якій професії є свої певні вимоги стосовно безпеки експлуатації тих чи інших пристроїв, які використовуються в роботі. У зв'язку з цим була створена і розвивається наука про охорону праці і життєдіяльності людини.

Стан діяльності, за якого з визначеною ймовірністю виключено прояв небезпек або ж відсутня надзвичайна небезпека, визначають як «безпеку». ДСТУ 2293-99 визначає термін «безпека» як стан захищеності особи та суспільства від ризику зазнати шкоди.

Дипломна робота розглядає питання сортування числових послідовностей, виконання яких здійснюється працівниками галузі інженерії програмного забезпечення. Працівники цієї галузі при роботі з комп'ютером наражаються ряду небезпечних і шкідливих виробничих факторів: електромагнітних полів (діапазон радіочастот: ВЧ, УВЧ і СВЧ), інфрачервоного і іонізуючого випромінювань, шуму і вібрації, статичної електрики і ін.

Охорона праці – це регульована предметна область, для якої потрібно визначити види заходів та відповідні умови їх проведення, необхідні для успішної реалізації і завершення кваліфікаційної роботи магістра зі спеціальності інженерія програмного забезпечення.

Організація заходів із забезпечення безпечних умов праці та поведінки у надзвичайних ситуаціях засновується на наступних нормативних документах:

* ДСТУ 2293-99. Охорона праці. Терміни та визначення;
* НАПБ А.01.001-2015. Правила пожежної безпеки в Україні;
* ДСН 3.3.6.037-99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку;
* НПАОП 0.00-1.28-10. Правила охорони праці при експлуатації ЕОМ;
* ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень;
* НАПБ Б.03.001-2204. Типові норми належності вогнегасників;
* ДБН В.2.5-28-2006. Природне і штучне освітлення.

Вивчення найважливіших заходів з охорони праці та безпеки у надзвичайних ситуаціях для працівників галузі інженерія програмного забезпечення наведено у Додатку Б та висвітлює наступні питання:

* організація та управління охороною праці на фірмі ТОВ «СІД СІНЕРДЖИ ЮКРЕЙН»;
* обґрунтування заходів з покращення умов охорони праці;
* розрахунок захисного заземлення;
* надзвичайні ситуації та шляхи їх запобігання.

В ході роботи проаналізовано робоче місце інженера-програміста, вимоги до освітлення, мікроклімату, іонізуюче випромінення при умовах роботи на моніторі високої точності.

Спираючись на проаналізовані дані, було розраховано захисне заземлення та визначений час перебування робітників в зоні радіоактивного забруднення.

В якості рекомендацій щодо поліпшення умов праці, основне – придбання та встановлення засобів, що забезпечать повноцінне захисне заземлення виробничого обладнання, а також пристроїв, що дозволяють виключити виникнення небезпечних ситуацій при повному або частковому припиненні енергопостачання і подальшому його відновленні, а також придбання засобів індивідуального захисту для використання в разі радіоактивної загрози.

**ВИСНОВКИ**

У процесі виконання дипломної роботи був розроблений новий метод сортування даних, який був впроваджений в розроблену програмну систему. Особливістю даного алгоритму являється його рекурсивність та ефективність цифрового обладнання, використаного в схемі.

Для досягнення цього результату були проаналізовані існуючі методи сортування даних різних типів, як паралельних так і послідовних. Були порівнянні їх особливості, складність та параметри.

На основі результатів було створено технічне завдання на розробку методу і програмної системи, в якому описувалися вимоги, план виконання окремих етапів, технічно-економічні показники та інші фактори.

В результаті виконання етапу завдання був розроблений алгоритм тасувальної мережі, а також описані її порядок дій та сама схема цифрового обладнання, яка доводить ефективність нового методу.

Даний метод був впроваджений в розроблену програмну систему нарівні з іншими найбільш популярними методами сортування. Сама системи була ретельно реалізована та протестована з метою покращення якості та продуктивності програми та зменшення кількості помилок.

**СПИСОК ЛІТЕРАТУРИ**

1. Кнут Д. Искусство программирования. Том 3. Сортировка и поиск: 2-е изд. – М.: Вильямс, 2000. – 844 с.

2. Заика И.В. Обзор методов сортировки/ И.В. Заика, И.А. Тюшнякова//Научные труды SWORLD, Т. 2, № 4 (41). – Иваново: Научный мир, 2015. – C. 56-61.

3. Кормен, Т., Лейзерсон, Ч., Ривест, Р. Алгоритмы: построение и анализ = Introduction to Algorithms. – М.: МЦНМО, 2000. – 960 с.

4. Ларман, К. Применение UML и шаблонов проектирования. Введение в ОО анализ, проектирование и унифицированный процесс UP: Пер. с англ. 2-е издание. – М.: Издательский дом «Вильямс», 2002. – 624 с.

5. Угрюмов Е.П. Цифровая схемотехника: Учеб. пособие для вузов. – 2-е изд., перераб. и доп. – СПб: БХВ-Петербург, 2004. – 800 с.

6. Принципы построения интерфейсов. – http://www.business-process.ru/interface/princip\_design\_inerface.htm

7. Албахари, Д. C# 6.0. Справочник. Полное описание языка. – Издательство Вильямс, 2017. – 1440с.

8. MacDonald M. Pro WPF 4.5 in C#. Windows Presentation Foundation in .NET 4.5. – Friendsof, 2017. – 1095.