МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії
(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з дисципліни «Основи програмування 2. Модульне програмування»
(назва дисципліни)

на тему: «Пошук заданих елементів у масиві»

Студентки 1 курсу, групи ІП-21
Скрипець Ольги Олександрівни
Спеціальності 121 «Інженерія програмного забезпечення»

Керівник
асистент Вовк Є.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____
Національна оцінка _____

Члени комісії _____ ст. вик. Головченко М. М.
(підпис) (посада, вчене звання, науковий ступінь, прізвище та ініціали)

_____ к. т. н., доц. Муха І. П.
(підпис) (посада, вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2023 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-21 Семестр 2

ЗАВДАННЯ

на курсову роботу студентки

Скрипець Ольги Олександрівни

(прізвище, ім'я, по батькові)

1. Тема роботи «Пошук заданих елементів у масиві»
2. Строк здачі студентом закінченої роботи 31.05.2023
3. Вихідні дані до роботи Технічне завдання. Додаток А
4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

Вступ, постановка задачі, теоретичні відомості, опис алгоритмів, опис програмного забезпечення, результати тестування програмного забезпечення, інструкція користувача, висновок, перелік посилань

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Скріншоти тестування спроєктованого програмного забезпечення

6. Дата видачі завдання 12.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	12.02.2023	
2.	Підготовка ТЗ	24.02.2023	
3.	Пошук та вивчення літератури з питань курсової роботи	28.02.2023	
4.	Розробка сценарію роботи програми	05.03.2023	
6.	Узгодження сценарію роботи програми з керівником	11.03.2023	
5.	Розробка (вибір) алгоритму рішення задачі	18.03.2023	
6.	Узгодження алгоритму з керівником	22.03.2023	
7.	Узгодження з керівником інтерфейсу користувача	29.03.2023	
8.	Розробка програмного забезпечення	10.04.2023	
9.	Налагодження розрахункової частини програми	14.04.2023	
10.	Розробка та налагодження інтерфейсної частини програми	25.04.2023	
11.	Узгодження з керівником набору тестів для контрольного прикладу	18.05.2023	
12.	Тестування програми	20.05.2023	
13.	Підготовка пояснювальної записки	27.05.2023	
14.	Здача курсової роботи на перевірку	31.05.2023	
15.	Захист курсової роботи	05.06.2023	

Студент _____ Скрипець О. О.

(підпис)

Керівник _____ Вовк Є. А.

(підпис) (прізвище, ім'я, по батькові)

"12" березня 2023 р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 44 сторінок, 12 рисунки, 16 таблиць, 4 посилання.

Мета роботи: розробка якісного програмного забезпечення, яке здійснює пошук елементів у масиві.

Вивчено методи пошуку елементів у масиві: послідовний метод, Фібоначчі, інтерполяційний та метод пошуку за допомогою Хеш-функції.

Виконана програмна реалізація алгоритму послідовного пошуку, Фібоначчі, інтерполяційного та методу Хеш-функції

У першому розділі описано постановку задачі.

У другому розділі описано теоретичні відомості про кожний з описаних вище алгоритмів.

У третьому розділі описані алгоритми за допомогою псевдокоду (послідовний метод пошуку, Фібоначчі, інтерполяційний та Хеш-функції).

У четвертому розділі описане програмне забезпечення за допомогою діаграми класів та таблиці з використаннями методів, функцій та класів.

У п’ятому розділі описано процес тестування кінцевого програмного забезпечення.

У шостому розділі наведена інструкція користувача.

У сьомому розділі описано тестування методів послідовного пошуку, Фібоначчі, інтерполяційного та пошуку за допомогою Хеш-функції.

КЛЮЧОВІ СЛОВА: МАСИВ, ПОШУК, МЕТОД ФІБОНАЧЧІ, ІНТЕРПОЛЯЦІЙНИЙ МЕТОД, МЕТОД ХЕШ-ФУНКЦІЇ.

ЗМІСТ

ВСТУП

Пошук елементів в масиві є однією з найпоширеніших операцій в програмуванні. Часто виникає необхідність знайти певний елемент в наборі даних для подальшого аналізу, обробки або відображення його значення користувачу.

Програма, що реалізує різні методи пошуку елементів в масиві, дозволить вибрати оптимальний алгоритм залежно від контексту і характеристик задачі. Основна мета такої програми - ефективно знаходити елементи в масиві за мінімальні часові та ресурсні витрати. При роботі з великими обсягами даних та наборами, швидкість пошуку може стати критичною, особливо якщо дані змінюються динамічно або якщо програма працює в реальному часі.

Традиційний послідовний метод пошуку, який просто проходиться по елементах масиву від початку до кінця, має складність O(n), де n - розмір масиву. Це може бути неефективно, особливо при великому розмірі масиву.

Тому, розробка та використання більш ефективних алгоритмів пошуку стає важливою задачею. У цій роботі будуть розглянуті та порівняні такі методи пошуку: метод Фібоначчі - алгоритм, який використовує властивості чисел Фібоначчі для зменшення кількості порівнянь при пошуку, інтерполяційний метод - алгоритм, який використовує інтерполяцію для наближення до шуканого значення та скорочення обсягу пошуку та метод Хеш-функції - алгоритм, який використовує хеш-функції для швидкого знаходження елемента за його ключем.

Ці методи мають різні переваги та особливості, і їх ефективність може залежати від особливостей даних та конкретного завдання. У цій роботі будуть досліджені та порівняні часові характеристики цих методів та їх складність.

Цей дослід допоможе розробникам та програмістам обрати оптимальний метод пошуку елементів в масиві залежно від конкретних вимог та обмежень їх проекту. Результати дослідження також можуть бути використані для покращення вже існуючих алгоритмів пошуку та оптимізації роботи з даними в різних областях, де швидкість пошуку має велике значення.

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде шукати задані елементи у масиві наступними методами:

а) послідовний метод (для порівняння);

б) метод Фібоначчі;

в) інтерполяційний метод;

г) метод Хеш-функції;

Вхідними даними для даної роботи є елемент, який вводить користувач.

Програмне забезпечення повинно обробляти масив розмір якого – не менше 1000 унікальних елементів, що генеруються випадковим чином.

Вихідними даними для даної роботи являється виведення на екран позиції (індексу) цього елементу в масиві або повідомлення про його відсутність.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Пошук заданих елементів у масиві можна реалізувати за допомогою чотирьох методів: послідовного, Фібоначчі, інтерполяційного та Хеш-функції.

2.1. Послідовний пошук

Метод послідовного пошуку у масиві елементів, також відомий як лінійний пошук, є простим алгоритмом пошуку певного елемента в масиві. Сутність цього методу полягає в послідовному переборі елементів масиву до знаходження шуканого елемента або до досягнення кінця масиву.

Тобто якщо ми маємо такий масив елементів:

$$\mathrm{A} = [a_0, b_1, c_2, d_3, i_4, f_5, g_6\ ]$$

Шуканим числом виступає елемент d.

Алгоритм починає з першого елементу нульового індексу масиву: a не співпадає з шуканим d. Переходимо до наступного елемента b, він також не співпадає з d, йдемо далі до c, він не є d. Отже, переходимо до наступного елементу d. На цьому кроці алгоритм знайшов співпадіння.

Тому виконалась умова завершення ітераційного процесу і метод послідовного пошуку знайшов шукане число d на четвертому кроці під індексом 3.

2.2. Метод пошуку Фібоначчі

Суть методу полягає в тому, що алгоритм використовує два індекси, щоб встановити діапазон пошуку в масиві. Значення цих індексів базуються на числах Фібоначчі, де кожне наступне число дорівнює сумі двох попередніх чисел.

Щоб виконати пошук Фібоначчі для початку потрібно знайти число Фібоначчі, яке більше або дорівнює розміру даного масиву в якому ми маємо

шукати key (шукане значення). Можна сказати, що якщо розмір масиву дорівнює n, то ми повинні знайти таке число Фібоначчі, F_k, що $F_k \geq n$.

Наступним кроком є обчислення F_{k-1}, F_{k-2}, $offset$ (зсув) і $index$ значення. Обчислюється $index$ за допомогою $offset$, n і F_{k-2}.

Далі ми маємо порівнювати key з елементом на позиції знайденого $index$ в масиві. Це порівняння дасть нам один із наступних трьох результатів. Якщо key і елемент масиву на позиції знайденого $index$ рівні, то key знаходиться в $index$ позиції в даному масиві. Якщо key менший за елемент масиву на позиції знайденого $index$, тоді ми шукаємо ключ у лівому піддереві до F_{k-2}. Якщо задане key більше, ніж елемент масиву на позиції знайденого $index$, тоді ми шукаємо у правому піддереві до F_{k-1}. Якщо key не знайдено, повторюються кроки ще раз, це може відбутися, коли, наприклад $F_{k-2} \geq 0$, у нас є число Фібоначі, яке перевищує довжину масиву n.

Таким чином, ми бачимо, що після кожної ітерації розмір масиву n зменшується на $\frac{2}{3}$ або $\frac{1}{3}$.

2.3. Інтерполяційний метод

Метод інтерполяційного пошуку використовується для швидкого пошуку елемента у впорядкованому масиві, особливо тоді, коли дані розподілені рівномірно. Основна ідея полягає у тому, що замість рівномірного зменшення діапазону пошуку ми оцінюємо можливе положення шуканого елемента на основі його значення та границь масиву.

Спочатку ми маємо оцінити положення шуканого елементу, використовуючи інтерполяційну формулу. За допомогою неї визначаємо приблизне положення шуканого елемента:

$$pos = low + \frac{(value - array[low]) * (high - low)}{array[high] - array[low]}$$

де $value$ - шукане значення, array - впорядкований масив, low - початкова нижня границя діапазону пошуку (індекс), high - початкова верхня границя діапазону пошуку (індекс).

Після цього ми маємо порівнювати шуканий елемент з елементом на знайденій позиції: Якщо $array[pos]$ дорівнює $value$, то шуканий елемент знайдений і повертається його позиція. Якщо $array[pos]$ менше $value$, то шуканий елемент може знаходитися у правій частині масиву. Тому ми змінюємо нижню границю. Для цього назначаємо $low = pos + 1$ і ще раз, використовуючи інтерполяційну формулу, визначаємо приблизне положення шуканого елементу. Якщо $array[pos]$ більше $value$, то шуканий елемент може знаходитися у лівій частині масиву. Тому ми змінюємо нижню границю. Для цього назначаємо $low = pos - 1$ і ще раз, використовуючи інтерполяційну формулу, визначаємо приблизне положення шуканого елементу.

Далі відбувається повторення попередніх кроків, до моменту, поки не буде знайдено шуканий елемент або діапазон пошуку стане порожнім ($low > high$). Якщо діапазон стає порожнім, шуканий елемент вважається відсутнім у масиві.

Таким чином, метод інтерполяційного пошуку використовує інтерполяційну формулу для оцінки положення шуканого елемента та зміщення границь діапазону пошуку на основі порівнянь з елементами масиву. Це дозволяє здійснювати швидкий пошук, зменшуючи діапазон пошуку з кожною ітерацією.

2.4. Метод Хеш-функції

Метод пошуку за допомогою хеш-функції використовується для ефективного пошуку елементів у великому об'ємі даних за допомогою хеш-таблиці. Основна ідея полягає в тому, що елементи масиву (ключі) перетворюються у хеш-коди за допомогою хеш-функції, і ці хеш-коди використовуються для швидкого доступу до елементів.

Спочатку відбувається створення хеш-таблиці: яка зазвичай є масивом фіксованого розміру. Кількість блоків залежить від конкретної реалізації.

Потім відбувається хешування ключів. Кожен ключ, який потрібно зберегти або знайти, віддається хеш-функції. Хеш-функція приймає ключ і повертає відповідний індекс у масиві. Цей індекс використовується для збереження або пошуку ключа у хеш-таблиці.

При збереженні елемента в хеш-таблицю, хеш-функція використовується для визначення індексу, за яким елемент буде збережений у масиві. Елемент може бути збережений безпосередньо у цьому індексі, або можуть використовуватись додаткові структури даних, такі як списки або дерева, для управління колізіями, коли два або більше ключів мають однакове хеш-значення.

При пошуку елемента за ключем, хеш-функція застосовується до ключа, щоб визначити відповідний індекс у масиві. Потім відбувається перевірка цього індексу на наявність елемента. Якщо елемент знайдений, повертається відповідний індекс. В іншому випадку, якщо елемент не знайдений або виникає колізія, виконується додатковий пошук у структурі даних, яка управляє колізіями.

3 ОПИС АЛГОРИТМІВ

Перелік всіх основних змінних та їхнє пиризначення наведено в таблиці 3.1.

Таблиця 3.1 – Основні змінні та їхні призначення

Змінна	Призначення
$arraySize$	Розмірність масиву
$array[arraySize]$	Масив елементів
isButton1Clicked	Прапорель для перевірки натиску на кнопку
searchValue	Шуканий елемент, введений користувачем
resultSequential	Індекс знайденого елементу за послідовним пошуком
resultFibonacci	Індекс знайденого елементу методом Фібоначі
resultInterpolation	Індекс знайденого елементу методом інтерполяції
resultHash	Інекс знайденого елементу методом Хеш-функції
hashBucket	Кількість блоків у Хеш-таблиці
hashTable	Хеш-таблиця

3.1. Загальний алгоритм

1) ПОЧАТОК
2) Натиск на кнопку генерації масиву
3) Генерація рандомного масиву:
    3.1) Цикл генерації кожного елементу
4) Сортування масиву алгоритмом вставкою:
    4.1) Цикл проходу по всіх елементах масиву:
        4.1.1) Встановлюємо змінну key рівною значенню елемента масиву arr[i]
        4.1.2) Ініціалізуємо змінну j зі значенням i - 1

4.1.3) Входимо в цикл while з умовою j >= 0 && arr[j] > key, де перевіряється, чи j не менше 0 і значення елемента arr[j] більше key

4.1.3.1) Обмінюємо значення елементів arr[j] і arr[j + 1], а потім зменшуємо значення j на 1.

4.1.4) Встановлюємо arr[j + 1] рівним значенню key

5) Вивести масив в поле для виведення

5.1) Цикл проходу по всіх елементах масиву

6) ЯКЩО кнопка не була натиснута видати повідомлення про помилку

7) Отримання часла, яке ввів користувач

8) Валідація введеного числа *searchValue*:

8.1) ЯКЩО довжина введення дорівнює 0, ТО видати повідомлення про помилку

8.2) Цикл перевірки чи рядок складається лише з цифр

8.2.1) ЯКЩО так, ТО продовжити

ІНАКШЕ видати повідомлення про помилку

8.3) Перевірити чи число знаходиться у допустимому значенні від 0 до 10000

8.3.1) ЯКЩО так, продовжити

ІНАКШЕ видати повідомлення про помилку

9) ЯКЩО валідація успішна, ТО продовжити.

9.1) Перевірка наявності введеного числа в масиві:

9.1.1) Цикл проходу по кожному елементу масиву

9.1.1.1) ЯКЩО поточний елемент дорівнює *searchValue,* ТО вихід із циклу

ІНАКШЕ видати повідомлення про помилку та присвоїти *searchValue* нуль

9.2) ЯКЩО *searchValue* дорівнює нуль, ТО очистити поле для введення

ІНАКШЕ виконати наступні дії:

9.2.1) Виконати пошук *searchValue* послідовним методом (підрозділ 1.2.)

9.2.2) Виконати пошук *searchValue* методом Фібоначчі (підрозділ 1.3.)

9.2.3) Виконати пошук *searchValue* інтерполяційним методом (підрозділ 1.4.)

9.2.4) Виконати пошук *searchValue* методом Хеш-Функції (підрозділ 1.5.)

9.2.5) Вивести результат пошуку послідовним методом на екран

9.2.6) Вивести результат пошуку методом Фібоначі на екран

9.2.7) Вивести результат пошуку інтерполяційним методом на екран

9.2.8) Вивести результат пошуку методом Хеш-Функції на екран

9.2.9) Виконати запис результатів у файл

10) КІНЕЦЬ

3.1. Алгоритм послідовного пошуку

1) ПОЧАТОК

2) Цикл проходу по всіх елементах масиву, починаючи з першого

2.1) Перевіряємо поточний елемент на збіг зі шуканим значенням.

2.2) ЯКЩО знайдений збіг, ТО повертаємо індекс поточного елемента і закінчуємо пошук.

3) ЯКЩО пройдені всі елементи і збіг не знайдений, повертаємо спеціальне значення, що позначає відсутність шуканого елемента.

4) КІНЕЦЬ

3.2. Метод пошуку Фібоначчі

1) ПОЧАТОК
2) Ініціалізуємо змінні fib2, fib1 та fib для обчислення чисел Фібоначчі.
3) В циклі обчислюємо числа Фібоначчі, поки fib не перевищує розмір масиву (arraySize)
   3.1) fib2 отримує значення fib1, fib1 отримує значення fib, fib обчислюється як сума fib1 та fib2
4) Для збереження зміщення індексу у масиві ініціалізуємо змінну offset з початковим значенням -1
5) За допомогою циклу, що працює, поки fib більше 1, ми виконуємо пошук шуканого значення у масиві:
   5.1) Визначаємо змінну i як мінімум між (offset + fib2) та (arraySize - 1) Вона використовується для порівняння шуканого значення з елементами масиву
   5.2) ЯКЩО значення елементу масиву array[i] менше шуканого значення value, ТО зсуваємо fib, fib1, fib2 та offset для продовження пошуку у правій частині масиву. Таким чином fib отримує значення fib1, fib1 отримує значення fib2, fib2 отримує значення fib - fib1, offset отримує значення i
   5.3) ЯКЩО значення елементу масиву array[i] більше шуканого значення value, ТО зсуваємо fib, fib1 та fib2 для продовження пошуку у лівій частині масиву. Таким чином fib отримує значення fib2, fib1 отримує значення fib1 - fib2, fib2 отримує значення fib - fib1
   5.4) ІНАКШЕ повертаємо індекс i, оскільки знайдено шуканий елемент у масиві.
6) ЯКЩО залишилося одне число Фібоначчі fib1 = 1 і наступний елемент масиву (array[offset + 1]) співпадає зі значенням value, повертаємо його індекс, щоб врахувати особливий випадок

ІНАКШЕ повертаємо -1, щоб вказати, що шуканий елемент відсутній у масиві.

7) КІНЕЦЬ

3.3. Інтерполяційний метод пошуку

1) ПОЧАТОК
2) Ініціалізація змінних: low - початкова нижня границя діапазону пошуку, high - початкова верхня границя діапазону пошуку
3) Для оцінки положення шуканого елемента використовуємо інтерполяційну формулу $pos = low + \frac{(value - array[low]) * (high - low)}{array[high] - array[low]}$
4) Виконується цикл while, який працює, поки low не перевищує high, а шукане значення value знаходиться в межах елементів array[low] і array[high]
   4.1) ЯКЩО елемент на знайденій позиції = value, ТО шуканий елемент знайдений і повертається його позиція
   4.2) ЯКЩО елемент на знайденій позиції менше value, ТО шуканий елемент може знаходитися у правій частині масиву. Змінюємо нижню границю low = pos + 1
5) ІНАКШЕ шуканий елемент може знаходитися у лівій частині масиву. Змінюємо верхню границю high = pos – 1
6) ЯКЩО діапазон стає порожнім, ТО шуканий елемент відсутній у масиві.
7) КІНЕЦЬ

3.4. Метод Хеш-функції

1) ПОЧАТОК
2) Створюється хеш-таблиця з 10 блоків
3) Цикл додавання елементів до таблиці

3.1) Проходження по кожному елементу масиву

3.2) Додавання ключа до Хеш-таблиці

3.3) Пошук числа в Хеш-таблиці

3.3.1) Присвоєння індексу значення Хеш-функції, яка key%hashBucket значення індекса

3.3.2) Запускається цикл визначається індекс блоку, до якого може бути збережений шуканий елемент

3.3.3) Виконується ітерація по списку цього блоку і порівнюється значення key з елементом у масиві arr за допомогою ітератора it.

3.3.3.1) ЯКЩО знайдений елемент збігається з ключем, ТО повертається відповідний індекс,

3.3.3.2) ЯКЩО елемент не знайдений, повертається -1.

4) КІНЕЦЬ

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Діаграма класів програмного забезпечення

Діаграма класу знаходиться на рисунку 4.1, в яку входить клас інтерфейсу MyForm, клас Search, який відповідає за пошук індексу елементу в масиві та клас Hashing для створення та обробки хеш-таблиці.

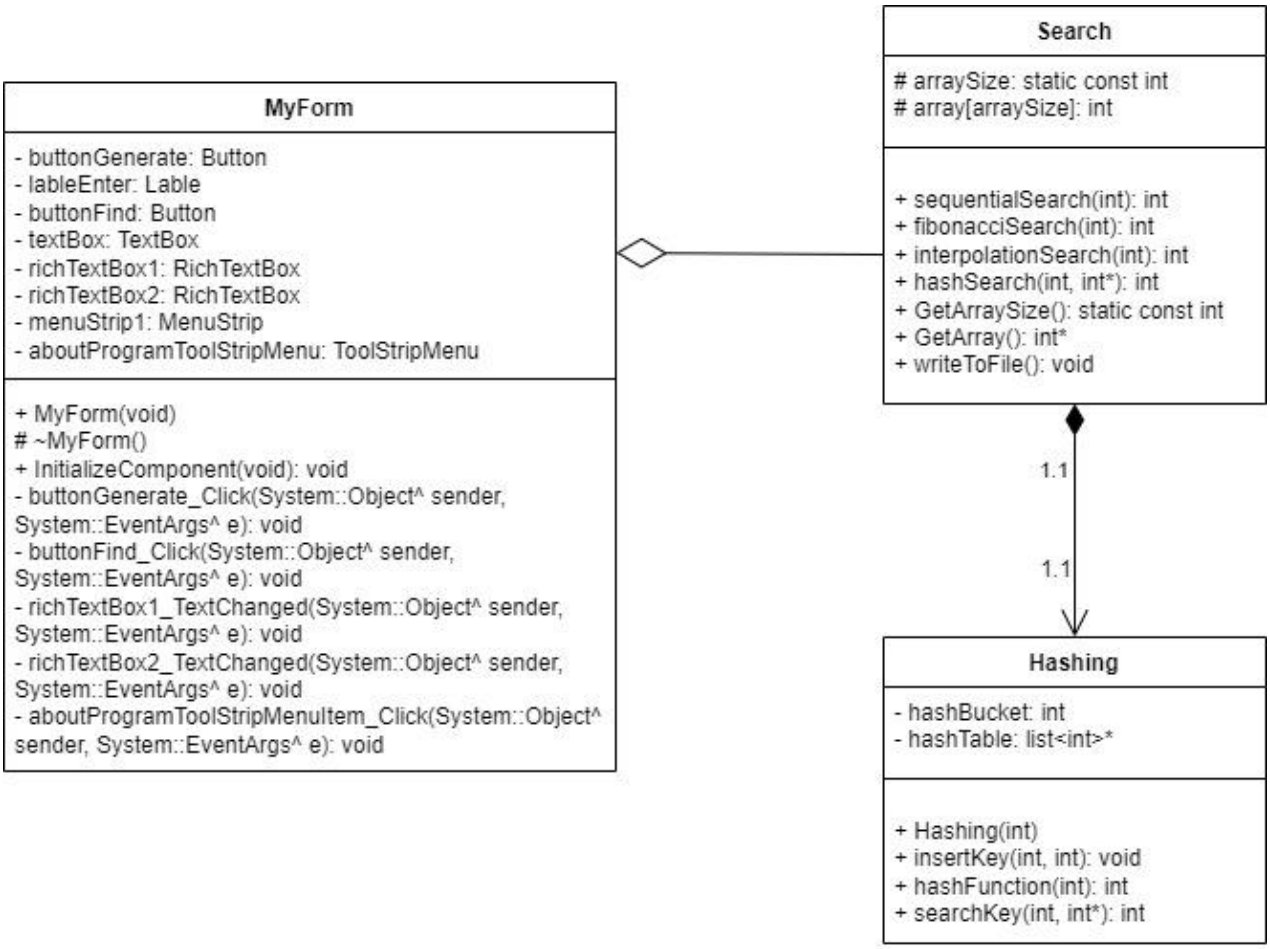

Рисунок 4.1 – Діаграма класів

4.2. Опис методів частин програмного забезпечення

4.2.1. Стандартні методи

У таблиці наведені 4.1 функції, методи та класи зі стандартної бібліотеки C++.

Таблиця 4.1– Стандартні методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
1	System::Object	ToString()	Повернення рядкового представлення об’єкта	int	String^	System.Windows.Forms
2	System::Windows::Forms::MessageBox	Show()	Виведення повідомлення, яке має бути відображене в діалоговому вікні.	String^	DialogResult	System.Windows.Forms
3	System::Convert	ToInt32()	Конвертація рядкового значення у ціле число типу int	String^	int	System.Windows.Forms
4	-	srand()	Ініціалізація генератора випадкових чисел, щоб забезпечити початкове значення для генерації псевдовипадкових чисел у програмі.	-	-	<cstdlib>
5	std::time_t	time()	повертає поточний час в секундах, який пройшов з 1 січня 1970 року	-	time_t	<ctime>

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
6	-	rand()	Генерація випадкових чисел	-	int	<cstdlib>
7	System::Char	IsDigit()	визначає, чи є заданий символ цифрою.	char/int	bool	<cctype>
8	System::String	Length	Повернення довжини рядка.	-	int	<string>
9	std::ofstream	ofstream	створення об'єкту файлового потоку для запису даних у файл.	string&	Об'єкт ofstream	<fstream>
10	std::fstream	is_open()	перевірка, чи був відкритий файловий потік успішно	-	bool	<fstream>
11	std::fstream	close()	Закриття файлового потоку	-	-	<fstream>
12	std::list	push_back()	Додати новий елемент у кінець списку.	-	-	<list>
13	std::list	begin()	Повернення ітератора, що вказує на перший елемент в списку	-	int	<list>

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
14	std::list	end ()	Повернення ітератора, що вказує на останній елемент в списку	-	int	<list>

4.2.2. Користувацькі методи

У таблиці 4.2 наведено всі користувацькі методи, функції та класи.

Таблиця 4.2 – Користувацькі методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
1	Search	sequentialSearch	Послідовний пошук	int	int	Search.h
2	Search	fibonacciSearch	Пошук методом Фібоначчі	int	int	Search.h
3	Search	interpolationSearch	Інтерполяційний пошук	int	int	Search.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
5	Search	hashSearch	Пошук за допомогою Хеш-функції	int	int, int*	Search.h
6	Search	GetArraySize	Отримання захищеного атрибуту стороннім класом	const int	const int	Search.h
7	Search	GetArray	Отримання захищеного атрибуту стороннім класом	int	Int*	Search.h
8	Search	writeToFile	Запис результатів в файл	const string, int	void	Search.h
9	MyForm	button1_Click	Опрацювання натиску користувачем на кнопку	-	void	MyForm.h
10	MyForm	buttonFind_Click	Опрацювання натиску користувачем на кнопку	-	void	MyForm.h
11	MyForm	проПрограму ToolStripMenuItem_Click	Опрацювання виводу повідомлення про програму	-	void	MyForm.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
12	MyForm	richTextBox1_TextChanged	Опрацювання зміни можливості вводу даних	-	void	MyForm.h
13	MyForm	richTextBox2_TextChanged	Опрацювання зміни можливості вводу даних	-	void	MyForm.h
16	Hashing	Hashing	Конструктор	int	int	Search.h
17	Hashing	insertKey	Додавання елементів до Хеш-таблиці	int, int	void	Search.h
18	Hashing	hashFunction	Втконує просту Хеш-функцію	int	int	Search.h
19	Hashing	searchKey	Виконує пошук числа в хеш-таблиці	int, int*	int	Search.h

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1. План тестування.

Тестування програмного забезпечення є важливим етапом розробки високоякісного програмного продукту. Його ціль полягає в перевірці відповідності програми вимогам, які встановлені у технічному завданні, забезпеченні впевненості у якості програмного забезпечення та виявленні потенційних проблем або неочікуваних станів, які могли бути пропущені розробником, але виявляться під час тестування.

Тестування програмного забезпечення включає перевірку основного функціоналу програми та реакцій на виняткові ситуації, які можуть виникнути в процесі її використання. Для цього ми використовуємо розроблений план тестування, який включає в себе різні тестові сценарії, дані та очікувані результати.

а) Тестування натиску на кнопку пошуку за відсутності генерації масиву.

б) Тестування правильності генерації масиву.

в) Тестування правильності введення значень.

   1) Тестування при відсутності введення та натиску на кнопку.
   2) Тестування при введенні некоректних символів.
   3) Тестування при введенні завеликого або замалого числа.

г) Тестування введення значення, якого немає у масиві.

д) Тестування коректності роботи алгоритмів.

   1) Тестування коректності роботи алгоритму послідовного пошуку.
   2) Тестування коректності роботи алгоритму пошуку методом Фібоначчі.
   3) Тестування коректності роботи алгоритму пошуку інтерполяційним методом.
   4) Тестування коректності роботи алгоритму пошуку методом Хеш-

функції.

е) Тестування коректності виведення результатів на екран.

ж) Тестування коректності запису результатів у файл.

5.2. Приклади тестування

Для забезпечення перевірки відповідності між очікуваними та фактичними результатами тестування, використаємо таблиці тестування.

Таблиці тестування 5.1 – 5.11 дозволяють систематизувати тестові сценарії, введені дані, очікувані результати та фактичні результати. Вони надають зручний спосіб відстежувати прогрес тестування та виявлення невідповідностей між очікуваними та отриманими результатами.

Таблиця 5.1 – Приклад роботи програми після натиску на кнопку за відсутності згенерованого масиву

Мета тесту	Перевірити натиск на кнопку пошуку за відсутності згенерованого масиву
Початковий стан програми	Відкрите вікно програми
Вхідні дані	-
Схема проведення тесту	Натиск на кнопку «знайти»
Очікуваний результат	Повідомлення про помилку, прохання спочатку згенерувати масив
Стан програми після проведення випробувань	Видано помилку «Спочатку згенеруйте масив!»

Таблиця 5.2 – Приклад правильності генерації масиву

Мета тесту	Перевірити правильність генерації масиву
Початковий стан програми	Відкрите вікно програми
Вхідні дані	-

Продовження таблиці 5.2

Схема проведення тесту	Натиск на кнопку «Згенерувати масив елементів»
Очікуваний результат	Виведення згенерованого масиву унікальних чисел на екран
Стан програми після проведення випробувань	Виведено коректний масив унікальних чисел

Таблиця 5.3 – Приклад роботи програми після натиску на кнопку при згенерованому масиві, але за відсутності введеного елементу

Мета тесту	Перевірити натиск на кнопку пошуку при згенерованому масиві, але за відсутності введеного елементу для пошуку
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Згенерований масив
Схема проведення тесту	Натиск на кнопку «знайти» за відсутності введеного елементу
Очікуваний результат	Повідомлення про помилку, прохання ввести число
Стан програми після проведення випробувань	Видано помилку «Уведіть число!»

Таблиця 5.4 – Приклад роботи програми при введенні некоректних символів

Мета тесту	Перевірити введення некоректних символів
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Згенерований масив

Продовження таблиці 5.4

Схема проведення тесту	Введення некоректних символів, натиск на кнопку «знайти»
Очікуваний результат	Повідомлення про помилку, прохання ввести число, використовуючи лише цифри
Стан програми після проведення випробувань	Видано помилку «Напишіть одне число, використовуючи лише цифри!»

Таблиця 5.5 – Приклад роботи програми при завеликого або замалого числа

Мета тесту	Перевірити введення завеликого або замалого числа
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Згенерований масив
Схема проведення тесту	Введення завеликого або замалого числа, натиск на кнопку «знайти»
Очікуваний результат	Повідомлення про помилку, прохання ввести число в діапазоні від 0 до 10000
Стан програми після проведення випробувань	Видано помилку «Число має бути від 0 до 10000!»

Таблиця 5.6 – Приклад роботи програми при введенні числа, якого немає в масиві

Мета тесту	Перевірити введення числа, якого немає в масиві
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Згенерований масив

Продовження таблиці 5.6

Схема проведення тесту	Введення числа, якого немає у виведеному масиві, натиск на кнопку «знайти»
Очікуваний результат	Повідомлення про те, що введене значення не знайдено в масиві
Стан програми після проведення випробувань	Видано результат «Введене значення не знайдено в масиві. Будь ласка, спробуйте ще раз.»

Таблиця 5.7 – Перевірка коректності роботи алгоритму послідовного пошуку

Мета тесту	Перевірити коректність роботи алгоритму послідовного пошуку
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Згенерований масив
Схема проведення тесту	Введення числа, яке є у виведеному масиві, натиск на кнопку «знайти»
Очікуваний результат	Виведення коректного індексу шуканого елементу на екран
Стан програми після проведення випробувань	Виведено коректний індекс шуканого елементу на екран

Таблиця 5.8 – Перевірка коректності роботи алгоритму Фібоначчі.

Мета тесту	Перевірити коректність роботи алгоритму пошуку методом Фібоначчі
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Згенерований масив

Продовження таблиці 5.8

Схема проведення тесту	Введення числа, яке є у виведеному масиві, натиск на кнопку «знайти»
Очікуваний результат	Виведення коректного індексу шуканого елементу на екран
Стан програми після проведення випробувань	Виведено коректний індекс шуканого елементу на екран

Таблиця 5.9 – Перевірка коректності роботи алгоритму інтерполяційним методом

Мета тесту	Перевірити коректність роботи алгоритму пошуку інтерполяційним методом
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Згенерований масив
Схема проведення тесту	Введення числа, яке є у виведеному масиві, натиск на кнопку «знайти»
Очікуваний результат	Виведення коректного індексу шуканого елементу на екран
Стан програми після проведення випробувань	Виведено коректний індекс шуканого елементу на екран

Таблиця 5.10 – Перевірка коректності роботи алгоритму методом Хеш-функції

Мета тесту	Перевірити коректність роботи алгоритму пошуку методом Хеш-Функції
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Згенерований масив

Продовження таблиці 5.10

Схема проведення тесту	Введення числа, яке є у виведеному масиві, натиск на кнопку «знайти»
Очікуваний результат	Виведення коректного індексу шуканого елементу на екран
Стан програми після проведення випробувань	Виведено коректний індекс шуканого елементу на екран

Таблиця 5.11 – Перевірка коректності запису результатів у файл

Мета тесту	Перевірити коректність запису результатів у файл
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Згенерований масив, виведено результати на екран
Схема проведення тесту	Натиск на кнопку «знайти»
Очікуваний результат	Виведення повідомлення про те, що результати збереено у файл, при відкритті файла можемо побачити результати.
Стан програми після проведення випробувань	Виведено повідомлення «Результати збережено у файлі result.txt» та при відкритті файла видно результат

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1. Робота з програмою

Після запуску виконавчого файлу з розширенням *.exe, відкривається головне вікно програми (рисунок 6.1).

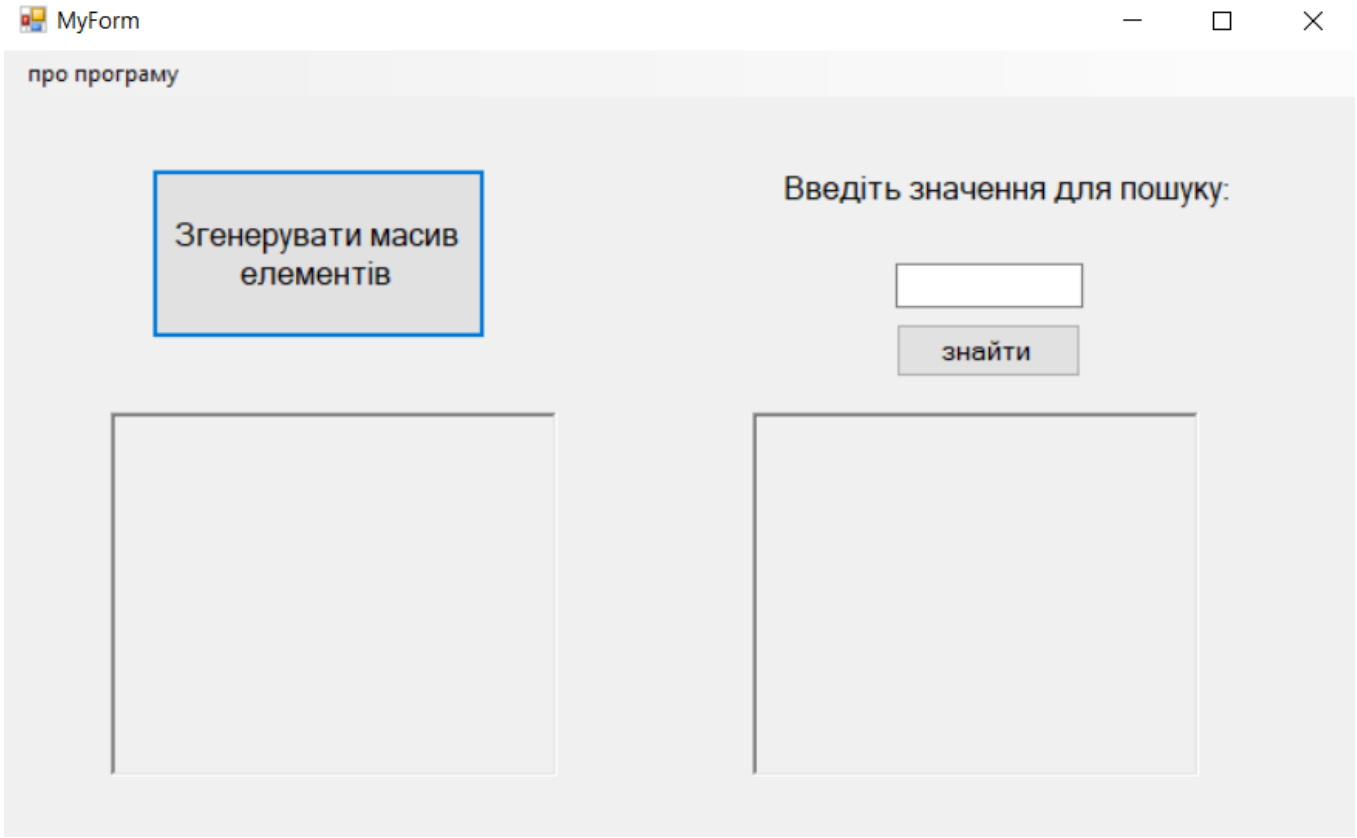

Рисунок 6.1 – Головне вікно програми

Далі потрібно натиснути «Згенерувати масив елементів» для генерації унікальних значень та виводу їх на екран (рисунок 6.2):

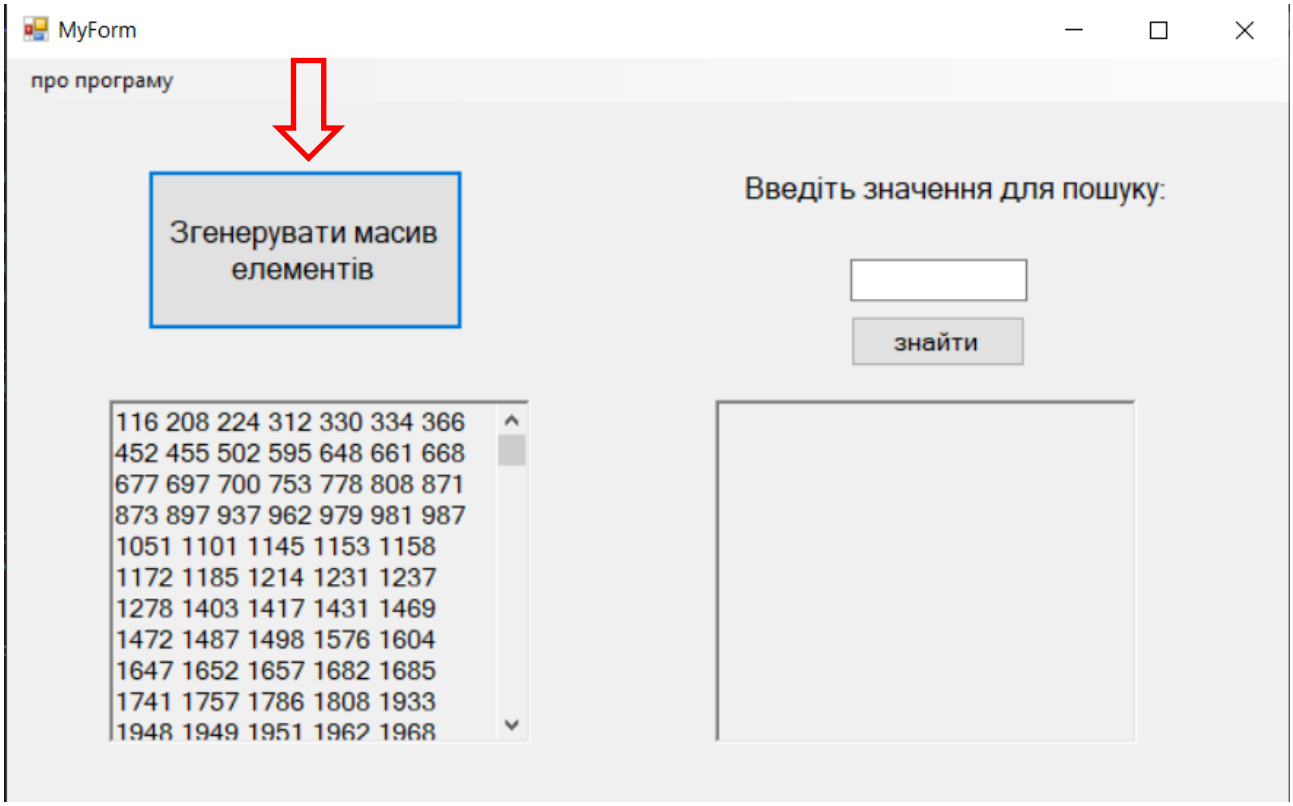

Рисунок 6.2 – Генерація масиву елементів

Потім введемо бажане значення з клавіатури, натиснемо на кнопку «знайти» і отримаємо результат (рисунок 6.3):

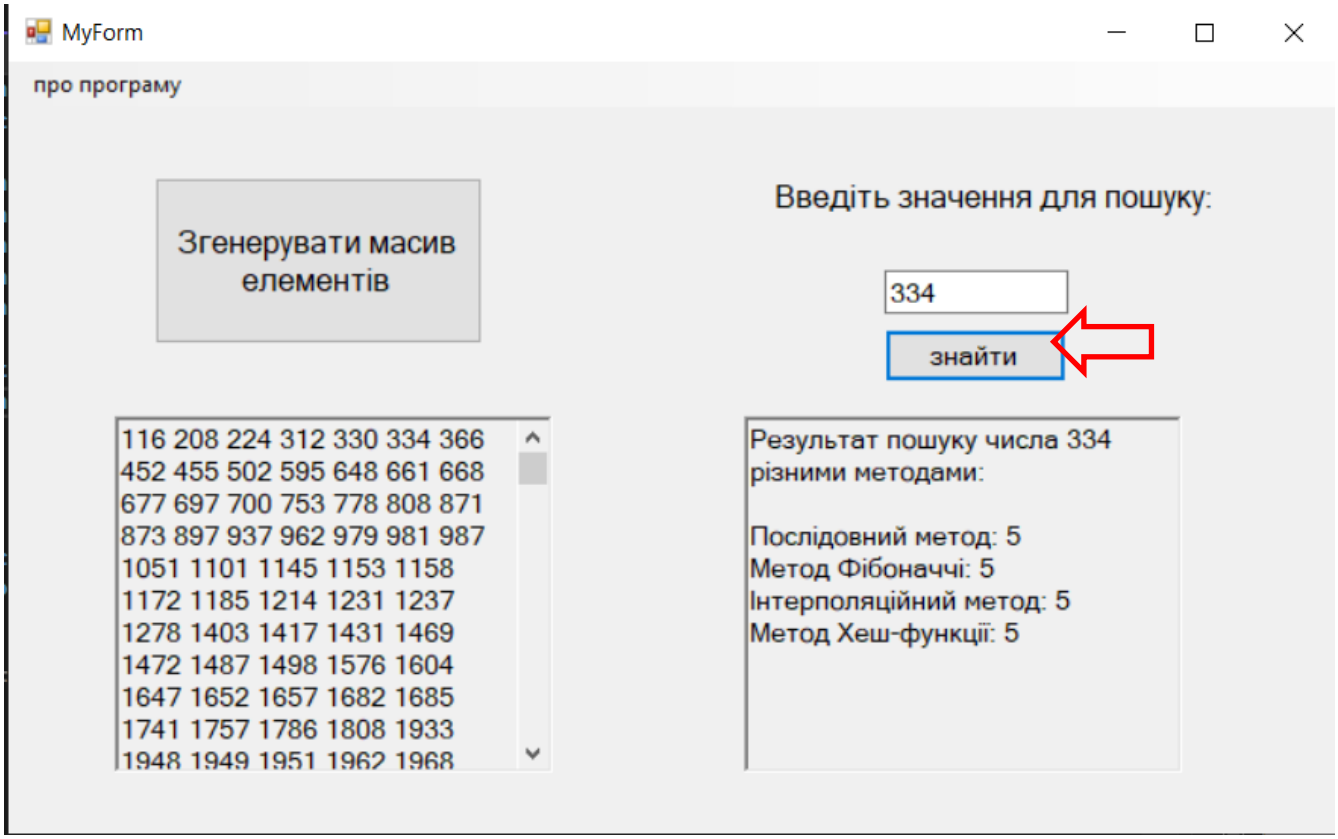

Рисунок 6.3 – Пошук елементів у масиві

Також отримаємо повідомлення про збереження результатів у файл (рисунок6.4).

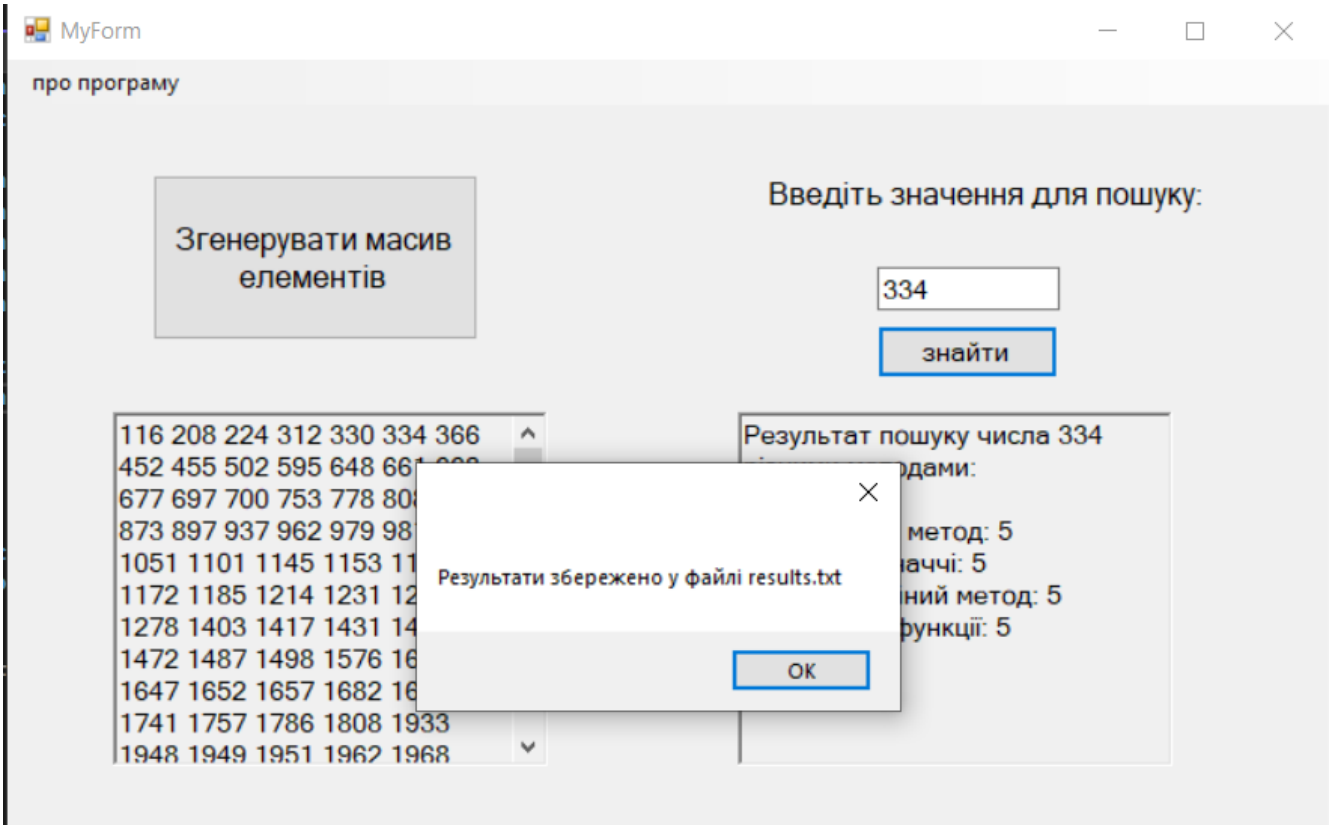

Рисунок 6.4 – Збереження результатів у файл

Такі значення ми побачимо, відкривши файл (рисунок 6.5).

```
139 152 247 268 291 314 372 379 391 398 433 480 563 580 586 597 626 639 697 735 829 886 895 1031 1050 1072 1103 1105
1136 1187 1199 1225 1377 1391 1426 1447 1451 1456 1492 1530 1539 1574 1642 1686 1692 1767 1799 1801 1828 1861 1928 1934
1970 1972 1996 2010 2019 2032 2098 2118 2162 2178 2200 2213 2246 2248 2288 2303 2334 2400 2459 2473 2481 2497 2521 2530
2536 2564 2631 2635 2644 2712 2735 2741 2751 2847 2851 2873 2878 2880 2930 2951 2976 3068 3110 3112 3120 3162 3166 3167
3169 3263 3298 3298 3317 3317 3331 3376 3396 3441 3498 3670 3671 3671 3706 3727 3734 3903 3926 3976 3990 3999 4123 4167
4203 4246 4274 4302 4316 4338 4417 4460 4489 4494 4553 4571 4598 4601 4616 4636 4647 4672 4738 4790 4798 4818 4868 4913
4929 4930 4937 4985 4987 4999 5131 5161 5269 5334 5340 5386 5406 5434 5503 5510 5528 5529 5535 5546 5564 5578 5625 5637
5695 5765 5771 5813 5826 5864 5925 5934 5960 6043 6054 6091 6097 6257 6257 6280 6293 6305 6351 6386 6508 6632 6662 6665
6731 6735 6809 6813 6835 6853 6863 6892 6901 7030 7064 7069 7105 7142 7150 7179 7214 7216 7257 7275 7300 7338 7350 7366
7376 7393 7401 7485 7544 7556 7557 7562 7614 7651 7653 7661 7725 7732 7738 7757 7758 7762 7834 7839 7852 7936 7965 7979
7989 8055 8061 8068 8075 8100 8140 8226 8227 8235 8270 8276 8279 8284 8306 8344 8354 8372 8380 8397 8413 8431 8451 8461
8483 8495 8499 8662 8724 8737 8815 8873 8889 8906 8942 8951 8964 8991 9006 9062 9086 9111 9127 9199 9200 9222 9264 9344
9369 9385 9387 9412 9422 9479 9497 9510 9543 9549 9554 9590 9610 9631 9664 9707 9709 9724 9762 9799 9810 9839 9850 9854
9910 9971 9976 10083 10105 10111 10162 10163 10183 10226 10284 10286 10294 10306 10425 10449 10458 10471 10504 10506
10563 10580 10611 10695 10861 10942 10942 10954 10964 10971 11003 11034 11053 11075 11099 11124 11125 11144 11167 11168
11172 11204 11249 11255 11261 11372 11470 11545 11548 11589 11623 11646 11655 11667 11696 11784 11819 11858 11862 11863
11866 11899 11955 11994 12047 12056 12068 12093 12128 12181 12236 12269 12334 12352 12360 12373 12418 12450 12463 12542
12625 12644 12736 12810 12813 12825 12839 12860 12861 12902 12995 12995 13052 13088 13095 13108 13142 13254 13256 13287
13296 13432 13446 13460 13516 13553 13558 13575 13592 13699 13742 13763 13840 13840 13854 13868 13869 13890 13936 13948
13976 13994 14046 14072 14093 14098 14114 14156 14171 14210 14219 14222 14226 14292 14338 14368 14403 14474 14496 14516
14580 14680 14755 14791 14823 14840 14842 14873 14903 14908 14915 14973 14978 15008 15027 15068 15149 15318 15388 15396
15406 15407 15454 15463 15469 15485 15490 15506 15507 15520 15607 15610 15643 15661 15680 15692 15727 15729 15735 15794
15801 15813 15879 15923 15947 15992 16043 16071 16135 16180 16249 16254 16269 16275 16301 16341 16350 16487 16541 16595
16600 16612 16637 16650 16700 16784 16793 16863 16875 16886 16917 16994 17101 17174 17221 17290 17304 17372 17453 17467
17486 17513 17523 17526 17570 17575 17595 17595 17621 17700 17710 17760 17768 17836 17845 17899 17936 17944 17950 18012
18049 18060 18157 18180 18225 18228 18268 18363 18376 18389 18431 18530 18535 18599 18599 18630 18656 18668 18708 18735
18841 18872 18884 18919 18972 18975 19027 19027 19056 19057 19076 19091 19095 19122 19144 19156 19197 19235 19274 19274
19357 19428 19429 19518 19597 19631 19654 19658 19675 19701 19730 19730 19754 19820 19837 19892 19941 20020 20028 20030
20055 20085 20123 20177 20196 20235 20273 20276 20283 20331 20415 20492 20569 20577 20619 20656 20671 20678 20697 20707
20752 20773 20867 20879 20889 20909 20962 20966 21030 21057 21059 21069 21161 21190 21192 21255 21266 21283 21292 21319
21337 21348 21382 21384 21393 21435 21455 21475 21569 21604 21618 21623 21646 21655 21668 21689 21699 21754 21803 21838
21901 21987 22004 22044 22100 22140 22176 22238 22308 22331 22386 22448 22449 22457 22507 22548 22585 22608 22626 22712
22833 22834 22858 22866 22866 22868 22888 22904 22912 22947 23004 23021 23025 23033 23050 23050 23147 23173 23181 23184
23205 23284 23287 23296 23333 23349 23360 23454 23489 23500 23510 23548 23554 23554 23566 23595 23598 23664 23704 23706
23713 23724 23739 23747 23770 23787 23795 23829 23840 23912 23925 23997 24110 24187 24271 24272 24288 24352 24373 24375
24418 24434 24444 24471 24529 24542 24551 24572 24598 24638 24665 24665 24687 24719 24759 24831 24839 24904 24916 24967
24977 24997 25009 25059 25181 25218 25241 25262 25271 25273 25283 25284 25298 25331 25365 25371 25372 25430 25481 25493
25516 25534 25581 25729 25819 25911 25979 26000 26064 26135 26177 26181 26202 26268 26271 26295 26338 26385 26397 26400
26414 26444 26491 26553 26558 26651 26661 26721 26810 26943 26986 27028 27033 27062 27073 27102 27114 27124 27140 27162
27188 27340 27425 27530 27598 27632 27632 27721 27770 27793 27808 27902 27913 27920 27943 28029 28067 28071 28114 28155
28166 28178 28190 28248 28278 28283 28374 28387 28446 28465 28486 28492 28728 28730 28741 28795 28806 28842 28912 28975
29186 29221 29231 29232 29248 29275 29306 29314 29329 29393 29415 29456 29472 29483 29520 29530 29535 29557 29583 29598
29693 29724 29820 29831 29840 29842 29904 29934 29938 29959 30018 30058 30098 30107 30165 30171 30188 30300 30304 30310
30382 30407 30424 30428 30466 30471 30494 30496 30506 30511 30561 30564 30580 30584 30692 30707 30765 30868 30883 30944
30970 30994 31001 31016 31067 31078 31182 31186 31187 31221 31232 31240 31258 31275 31294 31306 31324 31350 31444 31448
31475 31526 31564 31589 31651 31746 31782 31849 31851 31926 31972 31976 31986 32029 32039 32093 32098 32165 32167 32172
32195 32264 32281 32312 32359 32373 32398 32404 32404 32427 32482 32484 32493 32533 32545 32567 32598 32602 32613 32623
32651 32682 32723 32744

Результат пошуку числа 314 різними методами:

Послідовний метод: 5
Метод Фібоначчі: 5
Інтерполяційний метод: 5
Метод Хеш-функції: 5
```

Рисунок 6.5 – Вміст файлу

6.2. Формат вхідних та вихідних даних

Користувачем на вхід програми подається введення значення, за яким буде здійснений пошук в масиві, число має бути єдине, в діапазоні від 0 до 10000. На вхід алгоритму подається масив унікальних рандомно згенерованих цифр.

Результатом виконання програми є знайдений індекс введеного користувачем елементу масиву.

6.3. Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.6.

Таблиця 6.6 – Системні вимоги програмного забезпечення

	Мінімальні	Рекомендовані
Операційна система	Windows® XP/Windows Vista/Windows 7/ Windows 8/Windows 10 (з останніми обновленнями, x64)	Windows 7/ Windows 8/Windows 10 (з останніми обновленнями, x64)
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Pentium® D або AMD Athlon™ 64 X2
Оперативна пам'ять	256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/ Windows 8/Windows 10)	2 GB RAM
Відеоадаптер	Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог)	
Дисплей	800x600	1024x768 або краще
Прилади введення	Клавіатура, комп’ютерна миша	
Додаткове програмне забезпечення	Microsoft Visual Studio C++ (x86, x64)	

7 АНАЛІЗ РЕЗУЛЬТАТІВ

Головною задачею курсової роботи була реалізація програми для пошуку заданих елементів у масиві методами: послідовним, Фібоначчі, інтерполяційним, Хеш-Функції.

Критичні ситуації у роботі програми виявлені не були. Під час тестування було виявлено, що більшість помилок виникало тоді, коли користувачем вводилися не числові вхідні дані. Тому всі дані, які вводить користувач, ретельно провіряються на валідність і лише потім подаються на обробку програмі.

Достовірність коректності роботи алгоритмів можна побачити наочно: Результат виконання методів Фібоначчі, інтерполяційного, послідовного та Хеш-функції:

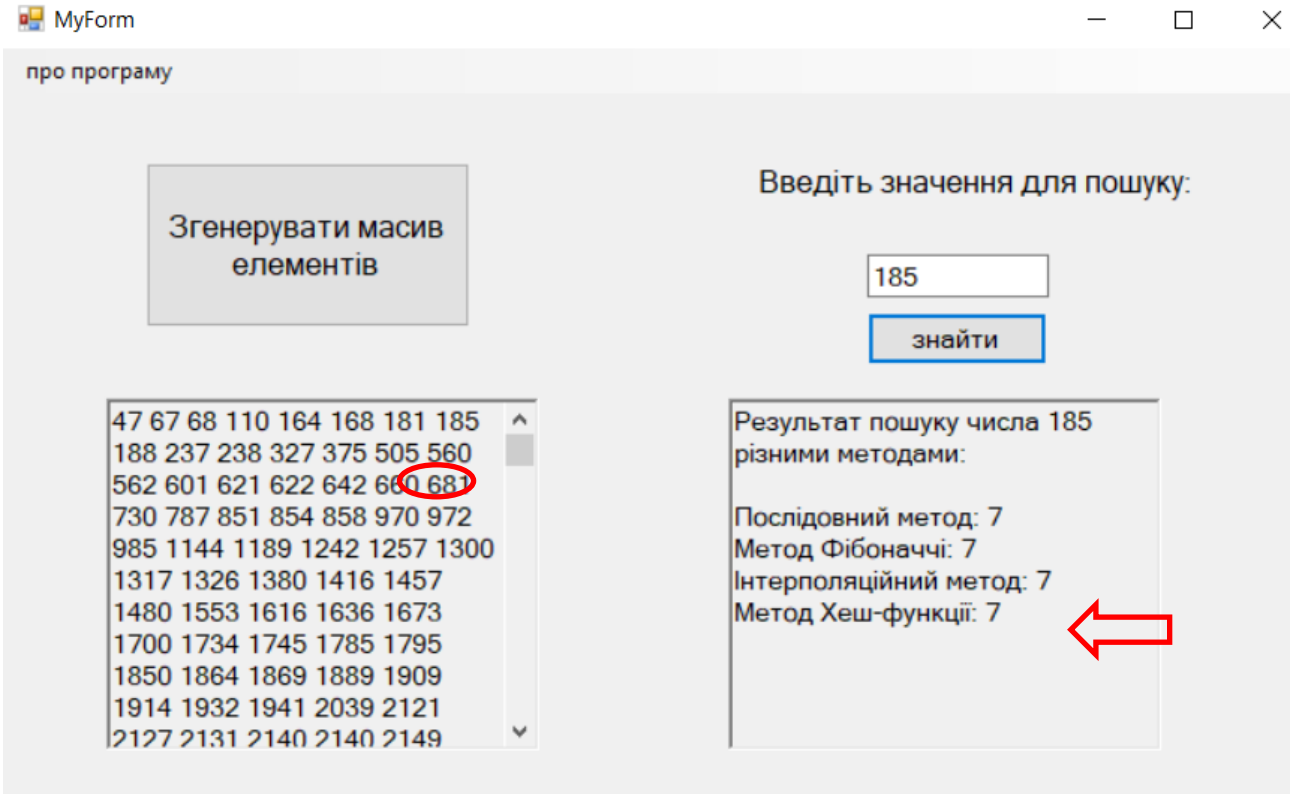

Рисунок 7.1 – Результат виконання всіх методів

Оскільки результат виконання збігається з наочним визначенням (рисунок 7.1), то дані методи працюють вірно.

Для теоретичного аналізу асимптотичної складності скористаємося стандартними методами для її визначення.

1) Послідовний (лінійний) алгоритм.

У послідовному методі пошуку елемента ми проходимо по елементам масиву послідовно, починаючи з першого і досягаючи останнього елемента, доки не знайдемо шуканий елемент або не перевіримо всі елементи масиву. У найкращому випадку, коли шуканий елемент знаходиться на першій позиції масиву, асимптотична складність послідовного алгоритму буде O(1). В цьому випадку ми знайдемо шуканий елемент безпосередньо на першій ітерації циклу і не потребуємо додаткових ітерацій.

У найгіршому випадку, коли шуканий елемент знаходиться на останній позиції масиву або взагалі відсутній в масиві, асимптотична складність послідовного алгоритму буде O(n). В цьому випадку нам доведеться пройти через всі n елементів масиву, перевіряючи кожен з них перед тим, як зрозуміти, що шуканий елемент відсутній або знаходиться на останній позиції.

В середньому випадку, припускаючи, що шуканий елемент розподілений рівномірно і його ймовірність знаходження на кожній позиції масиву однакова, асимптотична складність також буде O(n), оскільки в середньому нам доведеться пройти половину елементів масиву перед знаходженням шуканого елемента.

2) Метод Фібоначчі.

Це метод пошуку відсортованого масиву за допомогою алгоритму «розділяй та владарюй», який звужує можливі місця за допомогою чисел Фібоначчі.

У найкращому випадку, коли шуканий елемент знаходиться саме на початку або в кінці масиву, алгоритм може знайти його за декілька порівнянь. Це стає можливим, оскільки числа Фібоначчі збільшуються дуже швидко, а значення вхідного масиву зменшуються лінійно. Тому у найкращому випадку алгоритм працює з часовою складністю O(1), що означає постійну кількість операцій.

У найгіршому випадку, коли шуканий елемент знаходиться в середині масиву, алгоритм може вимагати багато порівнянь для знаходження його положення. Однак, завдяки використанню чисел Фібоначчі, асимптотична

складність методу Фібоначчі залишається досить ефективною. Середня асимптотична складність методу Фібоначчі становить O(log n), де n - розмір вхідного масиву. Це означає, що в середньому алгоритм потребує логарифмічну кількість операцій для пошуку елемента у відсортованому масиві.

3) Інтерполяційний метод

Інтерполяційний пошук є покращенням бінарного пошуку, яке використовує не тільки порівняння з елементом у середині діапазону, але і оцінку ймовірного місця розташування шуканого елемента. Цей метод передбачає, що елементи в масиві розташовані рівномірно.

У загальному випадку, якщо вважати, що масив має N елементів і передбачається рівномірний розподіл значень у масиві, то середня асимптотична складність інтерполяційного методу становить O(log log n) при впорядкованому масиві чисел.

Найкращий випадок для інтерполяційного пошуку відбувається, коли шуканий елемент знаходиться саме в середині діапазону. У цьому випадку, за допомогою оцінки ймовірного місця розташування, пошук може бути здійснений за менше число порівнянь. В такому найкращому випадку складність інтерполяційного методу становить O(1), тобто пошук може бути виконаний за постійний час.

Найгірший випадок для інтерполяційного пошуку відбувається, коли значення шуканого елемента знаходяться в крайніх позиціях масиву або значно віддалені одне від одного. У такому випадку, оцінка ймовірного місця розташування може бути неточною і пошук може потребувати багато порівнянь для знаходження шуканого елемента. В найгіршому випадку, складність інтерполяційного методу також становить O(n), де n - розмір масиву.

4) Метод Хеш-функції

Метод хеш-функції для пошуку елемента в масиві має середню асимптотичну складність O(1). Це означає, що час виконання алгоритму не залежить від розміру масиву або кількості елементів в ньому. Однак, це

стосується середнього випадку, коли відсутні колізії, тобто два різних ключа не мають однакового хешу.

У найкращому випадку, коли відсутні колізії, метод працює оптимально. Час виконання залишається стабільним незалежно від розміру масиву або кількості елементів в ньому. Таким чином, найкраща складність методу хеш-функції залишається O(1).

Проте, в найгіршому випадку, коли всі елементи масиву мають однаковий хеш або виникає багато колізій, ефективність методу хеш-функції може знизитися. У цьому випадку, складність може стати O(n), де n - розмір масиву. Причиною цього є необхідність виконати пошук серед усіх елементів масиву з однаковим хешем або вирішення колізій.

Таким чином, в середньому метод хеш-функції працює дуже ефективно зі складністю O(1). Однак, варто враховувати можливість найкращого і найгіршого випадків, які можуть вплинути на ефективність алгоритму пошуку елементів у масиві.

Для проведення практичного аналізу асимптотичної складності було протестовано програму на розмірності 1000-5000 елементів. Також була створена функція, яка виконує пошук кожного з елементів масиву, порівнює кількість елементарних операцій та ітерацій і таким чином знаходить найгірший випадок.

Результати тестування ефективності алгоритмів пошуку заданого значення у масиві наведено в таблиці 7.1:

Таблиця 7.1 – Тестування ефективності методів

Розмірність масиву, пошук останнього елементу	Параметри тестування	Метод			
		послідовний	Фібоначчі	інтерполяційний	Хеш-функції
1000	Кількість ітерацій	1000	29	10	5

Продовження таблиці 7.1

Розмірність масиву, пошук останнього елементу	Параметри тестування	Метод			
		послідовний	Фібоначчі	інтерполяційний	Хеш-функції
	Кількість елементарних операцій	1001	114	59	6
2500	Кількість ітерацій	2500	31	8	5
	Кількість елементарних операцій	2501	122	47	6
5000	Кількість ітерацій	5000	35	9	7
	Кількість елементарних операцій	5001	138	53	8

Візуалізацію результатів тестування таблиці 7.1 можна побачити на рисунках 7.1-7.5:

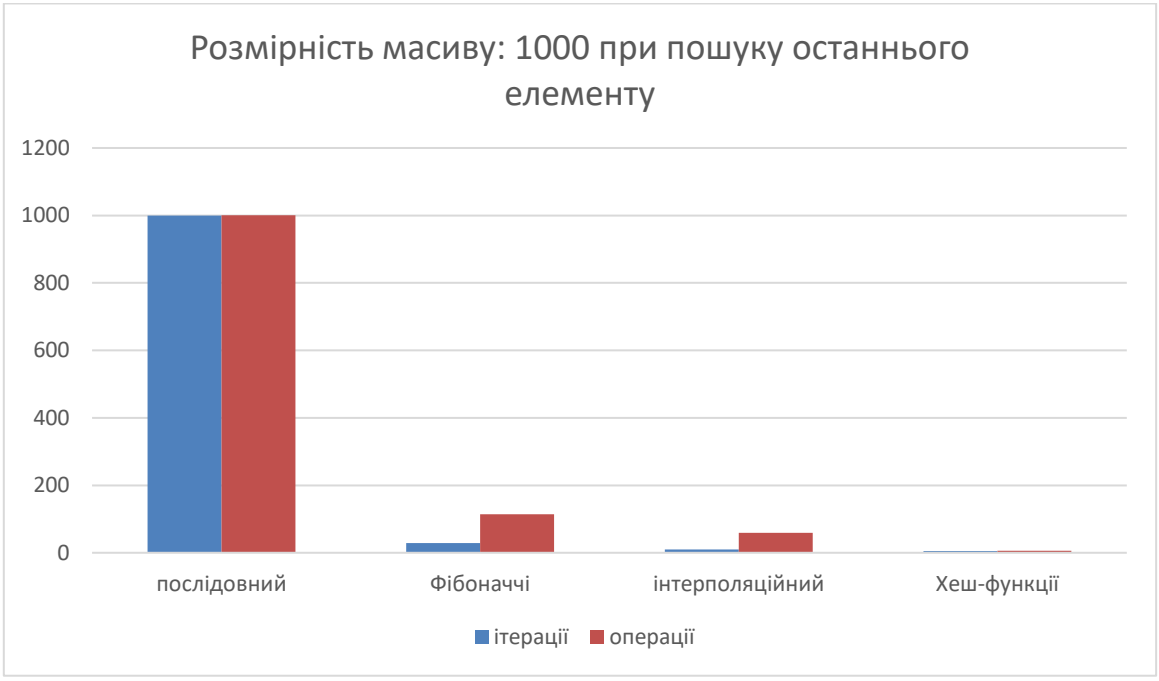

Рисунок 7.1 – Графік залежності кількості ітерацій та операцій методів за розмірністю масиву з 1000 елементів

Рисунок 7.2 – Графік залежності кількості ітерацій та операцій методів за розмірністю масиву з 2500 елементів

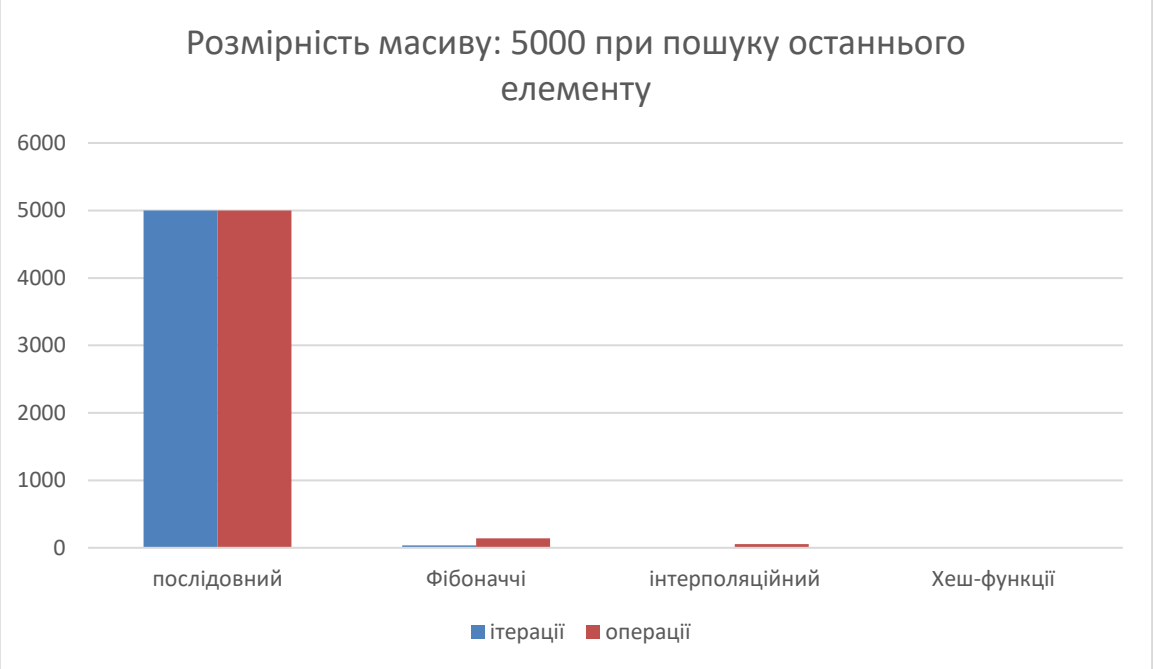

Рисунок 7.3 – Графік залежності кількості ітерацій та операцій методів за розмірністю масиву з 5000 елементів

Рисунок 7.4 – Графік залежності методів за різною розмірністю масиву

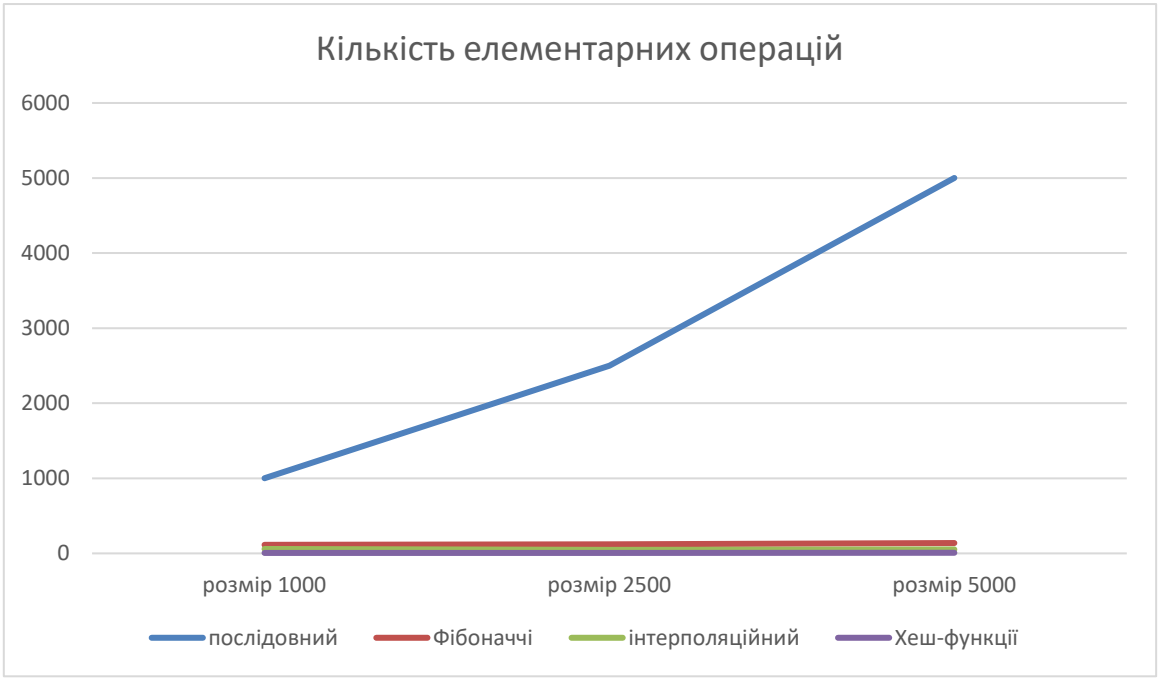

Рисунок 7.5 – Графік залежності методів за різною розмірністю масиву

В практичному аналізі асимптотичної складності розглянуті чотири методи пошуку елемента в масиві. Після проведення тестування на різних розмірах масиву були отримані такі результати:

а) Усі розглянуті методи дозволяють виконувати пошук в масиві великої або дуже великої розмірності. Це означає, що при збільшенні розміру масиву їхні асимптотичні складності не збільшуються дуже швидко, що дозволяє ефективно виконувати пошук незалежно від розміру вхідних даних.

б) Складність послідовного пошуку є лінійною (O(n)), тому зі збільшенням розміру масиву час пошуку зростає пропорційно. Метод Фібоначчі має логарифмічну складність (O(log n)), що означає, що час пошуку зростає повільніше, ніж лінійно. Інтерполяційний метод має складність O(log log n), що означає ще повільніше зростання часу пошуку зі збільшенням розміру масиву. Метод хеш-функції має постійну складність O(1) в середньому випадку, незалежно від розміру масиву, що робить його найшвидшим методом для практичного використання.

в) Найоптимальнішим методом для пошуку елементів у масиві є метод хеш-функції. Він забезпечує стабільну та швидку роботу незалежно від розміру масиву і кількості елементів в ньому.

ВИСНОВКИ

У даній курсовій роботі було розроблено програмне забезпечення для пошуку заданих елементів у масиві за допомогою різних методів. Розглянуті методи пошуку включали послідовний метод, метод Фібоначчі, інтерполяційний метод та метод Хеш-функції.

Було зазначено, що послідовний метод є традиційним, але неефективним, особливо при великому розмірі масиву. Інші методи, такі як метод Фібоначчі, інтерполяційний метод і метод Хеш-функції, були розглянуті як альтернативи для покращення швидкості пошуку.

Ці методи мають свої переваги та особливості. Метод Фібоначчі використовує властивості чисел Фібоначчі для зменшення кількості порівнянь при пошуку. Інтерполяційний метод використовує інтерполяцію для наближення до шуканого значення та скорочення обсягу пошуку. Метод Хеш-функції використовує хеш-функції для швидкого знаходження елемента за його ключем.

Метод Хеш-функції виявився найшвидшим і найстабільнішим методом пошуку елементів в масиві. Він забезпечує постійну швидкість незалежно від розміру масиву та кількості елементів, що робить його привабливим для багатьох практичних випадків.

Однак, варто враховувати обмеження методу хеш-функції, який підходить лише для пошуку конкретного значення за ключем. Якщо потрібно виконати пошук діапазону елементів або знайти найближче значення, інші методи можуть бути більш підходящими.

Також використання хеш-функцій вимагає додаткової пам'яті для збереження хеш-таблиці, що може бути проблемою при великій кількості елементів або обмеженому просторі пам'яті.

Результати дослідження часових характеристик і складності методів дозволять розробникам та програмістам обрати найоптимальніший метод пошуку елементів в масиві залежно від потреб свого проекту.

Також, результати дослідження можуть бути використані для покращення існуючих алгоритмів пошуку та оптимізації роботи з даними в різних областях, де швидкість пошуку має велике значення.

Загалом, розробка та використання ефективних методів пошуку елементів у масиві має важливе значення для покращення продуктивності програм та оптимізації роботи з даними.

ПЕРЕЛІК ПОСИЛАНЬ

1. Вступ до алгоритмів : Переклад з англійської третього видання:. / Кормен Т. Г., Лейзерсон Ч. Е., Рівест Р. Л., Стайн К. К: К. І. С, 2019. 42 с.
2. Креневич А.П. Алгоритми і структури даних. Підручник. – К.: ВПЦ "Київський Університет", 28. – 34 с.
3. Сухий О. Л. Алгоритми пошуку в інформаційних системах : методичні рекомендації / О. Л. Сухий, В. М. Міленін, В. М. Тарадайнік. – К., 2015. – 26 с.
4. Кузьменко І. М., Дацюк О. А. Базові алгоритми та структури даних : навчальний посібник. Київ :, 2022. 72 с.
5. Baeldung. Пошук Фібоначчі, стаття 2020. URL:https://www.baeldung.com/cs/fibonacci-search.
6. StudFiles. Метод інтерполяції, стаття 2018. – С. 15 URL:https://studfile.net/preview/10025806/page:15/.
7. GeefsforGeeks. Інтерполяційний пошук, стаття 2019. URL:https://www.geeksforgeeks.org/interpolation-search/#.
8. Нікіл Кумар Сінгх. Інтерполяційний пошук: стаття 2021. URL: https://www.topcoder.com/thrive/articles/interpolation-search.

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив
Керівник __Головченко М.М.____
«7» березня 2023 р.

Виконавець:
Студент ____Скрипець О.О._____
«7» березня 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання курсової роботи
на тему: «Пошук заданих елементів у масиві»
з дисципліни:
«Основи програмування»

Київ 2023

1. *Мета*: Метою курсової роботи є розробка якісного програмного забезпечення, яке здійснює пошук елементів у масиві.
2. *Дата початку роботи*: «07»березня 2023 р.
3. *Дата закінчення роботи*: «28» травня 2023 р.
4. *Вимоги до програмного забезпечення*.
   1) Функціональні вимоги:
      1. Генерація масиву:
         - Масив повинен містити принаймні 1000 унікальних елементів.
         - Елементи масиву генеруються випадковим чином.
      2. Введення шуканого значення:
         - Користувач може ввести шукане значення з клавіатури.
      3. Пошук за допомогою послідовного методу:
         - Реалізувати алгоритм пошуку, який перебирає елементи масиву послідовно, починаючи з першого елемента і закінчуючи останнім.
         - При знаходженні шуканого значення повернути позицію (індекс) цього значення у масиві або повідомлення про відсутність значення в масиві.
      4. Пошук за допомогою методу Фібоначчі:
         - Реалізувати алгоритм пошуку, який використовує метод Фібоначчі для визначення позиції цього значення у відсортованому масиві.
         - При знаходженні шуканого значення повернути позицію (індекс) цього значення у масиві або повідомлення про відсутність значення в масиві.
      5. Пошук за допомогою інтерполяційного методу:
         - Реалізувати алгоритм пошуку, який використовує інтерполяційну формулу для визначення нової позиції пошуку у відсортованому масиві.
         - При знаходженні шуканого значення повернути позицію (індекс) цього значення у масиві або повідомлення про відсутність значення в масиві.

6. Пошук за допомогою методу Хеш-функції:
   - Реалізувати алгоритм пошуку, який використовує хеш-функцію для швидкого знаходження значення у масиві.
   - Для використання методу Хеш-функції необхідно мати відомості про розподіл значень у масиві і використовувати хеш-таблицю або хеш-мапу.
   - При знаходженні шуканого значення повернути позицію (індекс) цього значення у масиві або повідомлення про відсутність значення в масиві.

2) Нефункціональні вимоги:
   - можливість запускання та використання програми в операційній системі Windows
   - програма повинна працювати швидко та ефективно, незалежно від розміру масиву та шуканого елемента.
   - програма повинна бути надійною та стабільною, не допускаючи помилок або неправильних результатів.
   - програма повинна мати зрозумілий та простий інтерфейс користувача, який дозволяє ввести шуканий елемент з клавіатури.

Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 2015 - Розробка технічної документації.

5. *Стадії та етапи розробки*:

1) Об'єктно-орієнтований аналіз предметної області задачі (до 03.03.2023 р.)

2) Об'єктно-орієнтоване проектування архітектури програмної системи (до 05.03.2023р.)
3) Розробка програмного забезпечення (до 18.03.2023 р.)
4) Тестування розробленої програми (до 20.05.2023 р.)
5) Розробка пояснювальної записки (до 27.05.2023 р.).
6) Захист курсової роботи (до 05.06.2023 р.).

6. *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду програмного забезпечення
вирішення задачі пошуку заданих елементів у масиві

(Найменування програми (документа))

CD-RW

(Вид носія даних)

21 арк, 212 Кб

(Обсяг програми (документа), арк.,

студентки групи ІП-21 І курсу
Скрипець О.О.

Файл вихідного коду *MyForm.cpp*

```
#include "MyForm.h"
#include "Search.h"
#include "AdditionalFunctions.h"


// простори імен
using namespace System;
using namespace System::Windows::Forms;



//точка входу для форми
[STAThreadAttribute]
void main(array<String^>^ args) {
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);

	KURSOVANEW::MyForm form;
	Application::Run(% form);


}

Search search;
bool isButton1Clicked = false;

//кнопка генерації масиву
System::Void KURSOVANEW::MyForm::buttonGenerate_Click(System::Object^ sender,
System::EventArgs^ e)
{
	isButton1Clicked = true;
	richTextBox1->Clear();

	generateRandomArray(search);
	insertionSort(search);

	//виведення масиву в richTextBox1
	int* arr = search.GetArray();
	for (int i = 0; i < search.GetArraySize(); ++i) {
		richTextBox1->Text += System::Convert::ToString(arr[i]);
```

```
            richTextBox1->Text += " ";
        }

        return System::Void();
}

//кнопка пошуку значень, які ввів користувач
System::Void KURSOVANEW::MyForm::buttonFind_Click(System::Object^ sender,
System::EventArgs^ e)
{

        if (!isButton1Clicked){
            MessageBox::Show("Помилка: Спочатку згенеруйте масив!");
            return;
        }

        richTextBox2->Clear();
        String^ input = textBox->Text;


        if (validateSearchValue(input)) {

            int searchValue = Convert::ToInt32(input);
            searchValue = checkValueInArray(search, searchValue);


            if (searchValue == 0) {
                textBox->Clear();

            }
            else {

                //пошук числа в масиві
                int resultSequential = search.sequentialSearch(searchValue);
                int resultFibonacci = search.fibonacciSearch(searchValue);
                int resultInterpolation = search.interpolationSearch(searchValue);
                int resultHash = search.hashSearch(searchValue, search.GetArray());

                //вивід результатів
                richTextBox2->Text += "Результат пошуку числа " + input  + "
різними методами: \n\n";
```

```
                    richTextBox2->Text += "Послідовний метод: " +
System::Convert::ToString(resultSequential) + "\n";
                    richTextBox2->Text += "Метод Фібоначчі: " +
System::Convert::ToString(resultFibonacci) + "\n";
                    richTextBox2->Text += "Інтерполяційний метод: " +
System::Convert::ToString(resultInterpolation) + "\n";
                    richTextBox2->Text += "Метод Хеш-функції: " +
System::Convert::ToString(resultHash) + "\n\n";

                    //запис у файл
                    search.writeToFile("result.txt", searchValue);
                }
        }



        return System::Void();
}

//про програму
System::Void
KURSOVANEW::MyForm::AboutProgramToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^ e)
{
        MessageBox::Show("\tЦя програма здійснює пошук заданих елементів у
масиві \n\t\tДля цього використовуються методи:\n\n -послідовний\n-Фібоначі\n
-інтерполяційний\n -Хеш-функцій\n\n Виконала Скрипець Ольга ІП-21",
"Інформація про програму");
        return System::Void();
}

//вимкнення можливість введення тексту в richTextBox1
System::Void KURSOVANEW::MyForm::richTextBox1_TextChanged(System::Object^
sender, System::EventArgs^ e)
{
        richTextBox1->ReadOnly = true;
        return System::Void();
}

//вимкнення можливість введення тексту в richTextBox2
```

```
System::Void KURSOVANEW::MyForm::richTextBox2_TextChanged(System::Object^
sender, System::EventArgs^ e)
{
	richTextBox2->ReadOnly = true;
	return System::Void();
}
```

Файл вихідного коду *MyForm.h*

```
#pragma once

namespace KURSOVANEW {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;

	public ref class MyForm : public System::Windows::Forms::Form
	{
	public:
		MyForm(void)
		{
			InitializeComponent();

		}

	protected:

		~MyForm()
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::Button^ buttonGenerate;

	protected:
	private: System::Windows::Forms::RichTextBox^ richTextBox1;
	private: System::Windows::Forms::Label^ labelEnter;
	private: System::Windows::Forms::Button^ buttonFind;
	private: System::Windows::Forms::TextBox^ textBox;
	private: System::Windows::Forms::RichTextBox^ richTextBox2;
	private: System::Windows::Forms::MenuStrip^ menuStrip1;
	private: System::Windows::Forms::ToolStripMenuItem^
AboutProgramToolStripMenuItem;
```

```
        private:

                System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

                void InitializeComponent(void)
                {
                        this->buttonGenerate = (gcnew System::Windows::Forms::Button());
                        this->richTextBox1 = (gcnew
System::Windows::Forms::RichTextBox());
                        this->labelEnter = (gcnew System::Windows::Forms::Label());
                        this->buttonFind = (gcnew System::Windows::Forms::Button());
                        this->textBox = (gcnew System::Windows::Forms::TextBox());
                        this->richTextBox2 = (gcnew
System::Windows::Forms::RichTextBox());
                        this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());
                        this->AboutProgramToolStripMenuItem = (gcnew
System::Windows::Forms::ToolStripMenuItem());
                        this->menuStrip1->SuspendLayout();
                        this->SuspendLayout();
                        //
                        // buttonGenerate
                        //
                        this->buttonGenerate->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(204)));
                        this->buttonGenerate->Location = System::Drawing::Point(103, 75);
                        this->buttonGenerate->Name = L"buttonGenerate";
                        this->buttonGenerate->Size = System::Drawing::Size(231, 108);
                        this->buttonGenerate->TabIndex = 0;
                        this->buttonGenerate->Text = L"Згенерувати масив елементів";
                        this->buttonGenerate->UseVisualStyleBackColor = true;
                        this->buttonGenerate->Click += gcnew System::EventHandler(this,
&MyForm::buttonGenerate_Click);
                        //
                        // richTextBox1
                        //
```

```
                this->richTextBox1->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 10.2F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                        static_cast<System::Byte>(204)));
                this->richTextBox1->Location = System::Drawing::Point(75, 230);
                this->richTextBox1->Name = L"richTextBox1";
                this->richTextBox1->ReadOnly = true;
                this->richTextBox1->Size = System::Drawing::Size(307, 230);
                this->richTextBox1->TabIndex = 1;
                this->richTextBox1->Text = L"";
                this->richTextBox1->TextChanged += gcnew
System::EventHandler(this, &MyForm::richTextBox1_TextChanged);
                //
                // labelEnter
                //
                this->labelEnter->AutoSize = true;
                this->labelEnter->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 12, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                        static_cast<System::Byte>(204)));
                this->labelEnter->Location = System::Drawing::Point(533, 75);
                this->labelEnter->Name = L"labelEnter";
                this->labelEnter->Size = System::Drawing::Size(292, 25);
                this->labelEnter->TabIndex = 17;
                this->labelEnter->Text = L"Введіть значення для пошуку:";
                //
                // buttonFind
                //
                this->buttonFind->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 10.2F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                        static_cast<System::Byte>(204)));
                this->buttonFind->Location = System::Drawing::Point(616, 173);
                this->buttonFind->Name = L"buttonFind";
                this->buttonFind->Size = System::Drawing::Size(128, 34);
                this->buttonFind->TabIndex = 19;
                this->buttonFind->Text = L"знайти";
                this->buttonFind->UseVisualStyleBackColor = true;
                this->buttonFind->Click += gcnew System::EventHandler(this,
&MyForm::buttonFind_Click);
                //
```

```
                    // textBox
                    //
                    this->textBox->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 10.2F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(204)));
                    this->textBox->Location = System::Drawing::Point(616, 136);
                    this->textBox->Name = L"textBox";
                    this->textBox->Size = System::Drawing::Size(128, 27);
                    this->textBox->TabIndex = 18;
                    //
                    // richTextBox2
                    //
                    this->richTextBox2->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 10.2F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(204)));
                    this->richTextBox2->Location = System::Drawing::Point(518, 230);
                    this->richTextBox2->Name = L"richTextBox2";
                    this->richTextBox2->ReadOnly = true;
                    this->richTextBox2->Size = System::Drawing::Size(307, 230);
                    this->richTextBox2->TabIndex = 20;
                    this->richTextBox2->Text = L"";
                    this->richTextBox2->TextChanged += gcnew
System::EventHandler(this, &MyForm::richTextBox2_TextChanged);
                    //
                    // menuStrip1
                    //
                    this->menuStrip1->ImageScalingSize = System::Drawing::Size(20,
20);
                    this->menuStrip1->Items->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^  >(1) { this-
>AboutProgramToolStripMenuItem });
                    this->menuStrip1->Location = System::Drawing::Point(0, 0);
                    this->menuStrip1->Name = L"menuStrip1";
                    this->menuStrip1->Size = System::Drawing::Size(936, 28);
                    this->menuStrip1->TabIndex = 21;
                    this->menuStrip1->Text = L"menuStrip1";
                    //
                    // AboutProgramToolStripMenuItem
                    //
```

```
            this->AboutProgramToolStripMenuItem->Name =
L"AboutProgramToolStripMenuItem";
            this->AboutProgramToolStripMenuItem->Size =
System::Drawing::Size(122, 24);
            this->AboutProgramToolStripMenuItem->Text = L"про програму";
            this->AboutProgramToolStripMenuItem->Click += gcnew
System::EventHandler(this, &MyForm::AboutProgramToolStripMenuItem_Click);
            //
            // MyForm
            //
            this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
            this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
            this->ClientSize = System::Drawing::Size(936, 501);
            this->Controls->Add(this->richTextBox2);
            this->Controls->Add(this->buttonFind);
            this->Controls->Add(this->textBox);
            this->Controls->Add(this->labelEnter);
            this->Controls->Add(this->richTextBox1);
            this->Controls->Add(this->buttonGenerate);
            this->Controls->Add(this->menuStrip1);
            this->MainMenuStrip = this->menuStrip1;
            this->Name = L"MyForm";
            this->Text = L"MyForm";
            this->menuStrip1->ResumeLayout(false);
            this->menuStrip1->PerformLayout();
            this->ResumeLayout(false);
            this->PerformLayout();

        }
#pragma endregion
    private: System::Void buttonGenerate_Click(System::Object^ sender,
System::EventArgs^ e);
    private: System::Void buttonFind_Click(System::Object^ sender,
System::EventArgs^ e);
    private: System::Void AboutProgramToolStripMenuItem_Click(System::Object^
sender, System::EventArgs^ e);
    private: System::Void richTextBox1_TextChanged(System::Object^ sender,
System::EventArgs^ e);
    private: System::Void richTextBox2_TextChanged(System::Object^ sender,
System::EventArgs^ e);
```

```
};


}
```

Файл вихідного коду *Search.cpp*

```
#include "Search.h"



//функція послідовного пошуку
int Search::sequentialSearch(int value) {
   for (int i = 0; i < arraySize; ++i) {
      if (array[i] == value) {
         return i;
      }
   }
   return -1;
}

//функція пошуку методом Фібоначі
int Search::fibonacciSearch(int value) {
   int fib2 = 0;
   int fib1 = 1;
   int fib = fib1 + fib2;

   while (fib < arraySize) {
      fib2 = fib1;
      fib1 = fib;
      fib = fib1 + fib2;
```

```
    }

    int offset = -1;
    while (fib > 1) {
        int i = (((offset + fib2) < (arraySize - 1)) ? (offset + fib2) : (arraySize - 1));
        if (array[i] < value) {
            fib = fib1;
            fib1 = fib2;
            fib2 = fib - fib1;
            offset = i;
        }
        else if (array[i] > value) {
            fib = fib2;
            fib1 = fib1 - fib2;
            fib2 = fib - fib1;
        }
        else {
            return i;
        }
    }

    if (fib1 == 1 && array[offset + 1] == value) {
        return offset + 1;
    }

    return -1;
}

//функція пошуку інтерполяційним методом
int Search::interpolationSearch(int value) {
    int low = 0;
    int high = arraySize - 1;

    while (low <= high && value >= array[low] && value <= array[high]) {
        //формула інтерполяції для знаходження наближеної позиції шуканого
елемента
        int position = low + ((value - array[low]) * (high - low)) / (array[high] - array[low]);

        if (array[position] == value) {
            return position;
        }
```

```
        if (array[position] < value) {
            low = position + 1;
        }
        else {
            high = position - 1;
        }
    }

    return -1;
}

//функція запису результатів до файлу
void Search::writeToFile(const std::string& filename, int searchValue) {
    std::ofstream file(filename, std::ios::app);

    if (file.is_open()) {

        int* arr = Search::GetArray();
        // Записуємо елементи масиву у файл
        for (int i = 0; i < GetArraySize(); i++) {
            file << arr[i] << " ";
        }


        // Виконуємо пошук і записуємо результати у файл
        int resultSequential = sequentialSearch(searchValue);
        int resultFibonacci = fibonacciSearch(searchValue);
        int resultInterpolation = interpolationSearch(searchValue);
        int resultHash = hashSearch(searchValue, arr);

        file << "\n\nРезультат пошуку числа " << searchValue << " різними методами:
\n" << std::endl;

        file << "Послідовний метод: " << resultSequential << std::endl;
        file << "Метод Фібоначчі: " << resultFibonacci << std::endl;
        file << "Інтерполяційний метод: " << resultInterpolation << std::endl;
        file << "Метод Хеш-функції: " << resultHash << "\n" << std::endl;
```

```
        file.close();
        MessageBox::Show("Результати збережено у файлі results.txt");


    }
    else {
        MessageBox::Show("Помилка при відкритті файлу results.txt", "Помилка");
    }

}



//функція пошуку елементу за допомогою Хеш-функції
int Search::hashSearch(int key, int* array) {
    Hashing hashTable(10); // Створення хеш-таблиці з 10 блоків
    int index = 0;
    // Додавання елементів до хеш-таблиці
    for (int i = 0; i < arraySize; i++) {
        hashTable.insertKey(array[i], i);

        // Пошук числа в хеш-таблиці
        index = hashTable.searchKey(key, array);
    }
    return index;
}


Hashing::Hashing(int b) {
    this->hashBucket = b;
    hashTable = new std::list<int>[hashBucket];
}

//функція, яка додає ключ до хеш-таблиці
void Hashing::insertKey(int key, int value) {
    int index = hashFunction(key);
    hashTable[index].push_back(value);
}

//Хеш-функція
int Hashing::hashFunction(int x) {
```

```
    return (x % hashBucket);
}

// Пошук числа в хеш-таблиці
int Hashing::searchKey(int key, int* arr) {
    int index = hashFunction(key);
    std::list<int>::iterator it;

    for (it = hashTable[index].begin(); it != hashTable[index].end(); it++) {

        if (key == arr[*it])
        {
            return *it;
        }
    }

    return -1;
}
```

Файл вихідного коду *Search.h*

```
#pragma once
#include "MyForm.h"
#include <fstream>
#include <string>
#include <list>
#include <cstdlib>
#include <iostream>
#include <msclr/marshal_cppstd.h>
#include <unordered_map>

#ifndef SEARCH_H
#define SEARCH_H


using namespace System;
using namespace System::Windows::Forms;


class Search {
protected:
   static const int arraySize = 1000;
   int array[arraySize];
public:

   int sequentialSearch(int value);
   int fibonacciSearch(int value);
   int interpolationSearch( int value);
```

```
    int hashSearch(int value, int* array);

    static const int GetArraySize() { return arraySize; }
    int* GetArray() { return array; }

    void writeToFile(const std::string& filename, int searchValue);
};


class Hashing : public Search {
    int hashBucket;
    std::list<int>* hashTable;
public:
    Hashing(int V);
    void insertKey(int key, int value);
    int hashFunction(int x);
    int searchKey(int key, int* array);
};



#endif
```

Файл вихідного коду *AdditionalFunctions.cpp*

```
#include "AdditionalFunctions.h"


// Функція для генерації масиву випадкових чисел
void generateRandomArray(Search& search) {
	srand(static_cast<unsigned int>(time(nullptr)));
	int* arr = search.GetArray();
	for (int i = 0; i < search.GetArraySize(); i++) {
		arr[i] = rand();
	}
}

// Функція для сортування масиву за алгоритмом сортування вставкою
void insertionSort(Search& search) {
	int* arr = search.GetArray();
	for (int i = 1; i < search.GetArraySize(); i++) {
		int key = arr[i];
		int j = i - 1;

		while (j >= 0 && arr[j] > key) {
			arr[j + 1] = arr[j];
			j--;
		}

		arr[j + 1] = key;
```

```
        }
}


//функція валідації введеного значення
bool validateSearchValue(String^ input)
{

        //перевірка чи рядок не порожній
        if (input->Length == 0)
        {
                MessageBox::Show("Уведіть число", "Помилка");
                return false;
        }

        //перевірка чи рядок складається лише з цифр
        for each (wchar_t c in input)
        {
                if (!System::Char::IsDigit(c))
                {
                        MessageBox::Show("Напишіть одне число, використавши лише
цифри", "Помилка");
                        return false;
                }
        }


        int searchValue = System::Convert::ToInt32(input);

        //перевірка чи число знаходиться в допустимому діапазоні
        if (searchValue < 0 || searchValue > 100000)
        {
                MessageBox::Show("Число має бути від 0 до 10000", "Помилка");
                return false;
        }

        return true;
}
```

```
//функція перевірки чи число знаходиться у масиві
int checkValueInArray(Search& search, int searchValue) {

	bool validValue = false;

	do {

		//перевірка, чи введене значення знаходиться у масиві
		int* arr = search.GetArray();
		for (int i = 0; i < search.GetArraySize(); i++) {
			if (arr[i] == searchValue) {
				validValue = true;
				break;
			}

		}
		if (!validValue) {
			MessageBox::Show("Введене значення не знайдено в масиві.
Будь ласка, спробуйте ще раз.", "Помилка");
			searchValue = 0;
			validValue = true;

		}
	} while (!validValue);

	return searchValue;
}
```

Файл вихідного коду *AdditionalFunctions.h*

```
#pragma once
#include "MyForm.h"
#include "Search.h"
#include <fstream>
#include <string>
#include <ctime>
#include <msclr/marshal_cppstd.h>


using namespace System;
using namespace System::Windows::Forms;


void generateRandomArray(Search& search);
void insertionSort(Search& search);
bool validateSearchValue(String^ input);
                    int checkValueInArray(Search& search, int searchValue);
```