МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп'ютерних систем та інфокомунікацій

Кафедра комп'ютерних систем, мереж і кібербезпеки (503)

Лабораторна робота № *4*

Дослідження простих алгоритмів пошуку
(назва лабораторної роботи)

з дисципліни *Моделі та структури даних*
(шифр)

ХАІ.503.525а.03О.123-Комп'ютерна інженерія, ПЗ №9629619

Виконав студент гр. 525а *Литвиненко А.В.*
(№ групи) (П.І.Б.)
16.11.2022
(підпис, дата)

Перевірив канд. техн. наук, доцент

А. В. Шостак
(підпис, дата) (П.І.Б.)

Харків – 2022

Варіант 5
Задача 1

Частина 1. Постановка завдання

Умова:

1. Разработать проект для исследования простых алгоритмов поиска в соответствии с вариантом.
2. Разработать интерфейс проекта, *позволяющий*:
   - задавать размерность и диапазон элементов массива;
   - элемент для поиска (образец);
   - осуществлять выбор алгоритма поиска для исследования;
   - осуществлять вывод информации о результатах исследования алгоритма поиска (исходный массив, пошаговую работу алгоритма поиска (при небольшой размерности массива), показатели качества работы алгоритма поиска).
3. Создать подпрограмму, реализующую алгоритм поиска в соответствии с вариантом. В подпрограмме *предусмотреть*:
   - поиск заданного элемента массива и индекс его первого вхождения;
   - определение числа присваиваний алгоритма поиска - *Pr*;
   - определение числа сравнений алгоритма поиска - *Sr*;
   - определение суммы присваиваний и сравнений алгоритма поиска *Sum*.
4. В качестве исследуемого массива использовать одномерный массив целых чисел, равномерно распределенных в интервале [*A, B*]. Для формирования элементов одномерного массива и образца использовать датчик равномерно распределенных чисел *rand*().
5. Результатом работы алгоритма поиска является номер первого вхождения образца в исследуемый массив.

Варианты задач по лабораторной работе:

1. Линейный поиск (или линейный поиск с барьером).
2. Бинарный поиск.
3. Интерполяционный поиск.
4. *m*–блочный поиск.
5. Поиск с помощью встроенных функций C# (или *C++*).

Умова з додатка:

5.	[0, 1000]	2, 4, 5

Частина 2. Схема класу

На основі постановки завдання розроблений алгоритм, представлений на рисунку 1.

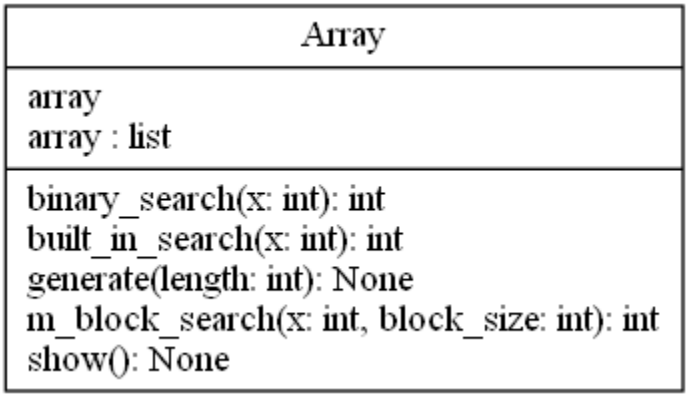


Рисунок 1 - Алгоритм перетворення

Частина 3. Заміри показників

Таблиця 1 – Показники якості бінарного пошуку

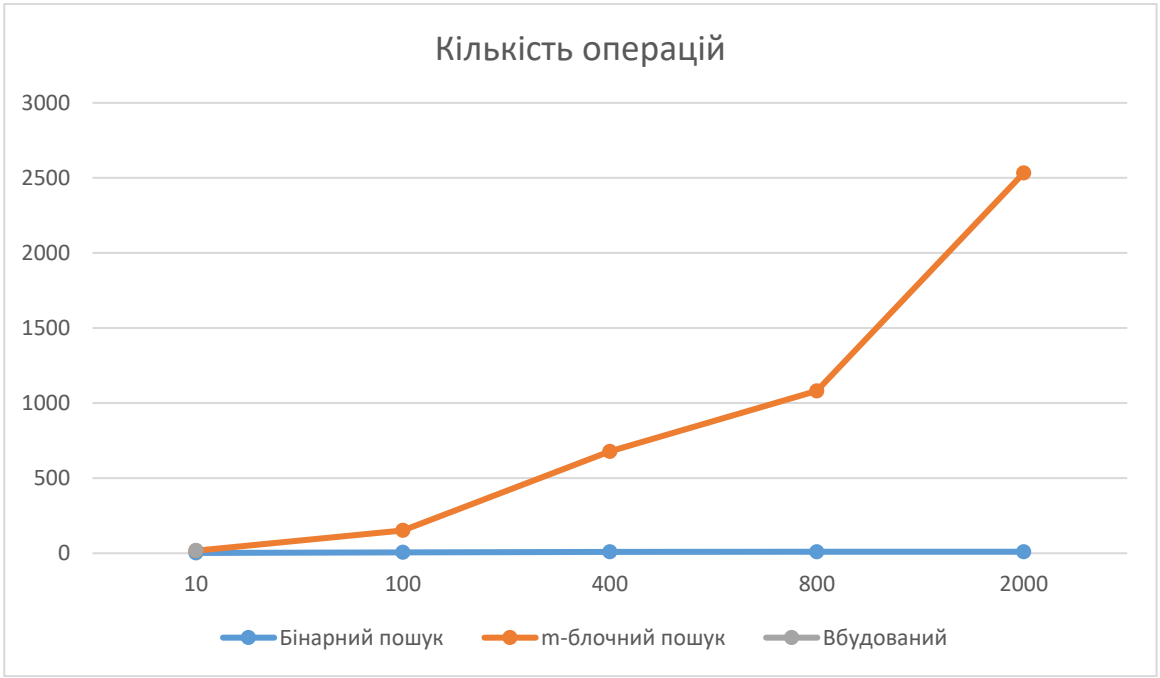
N	10	100	400	800	2000
Sr	2	6	9	10	10
Time	0.01536	0.02083	0.01092	0.02143	0.02114

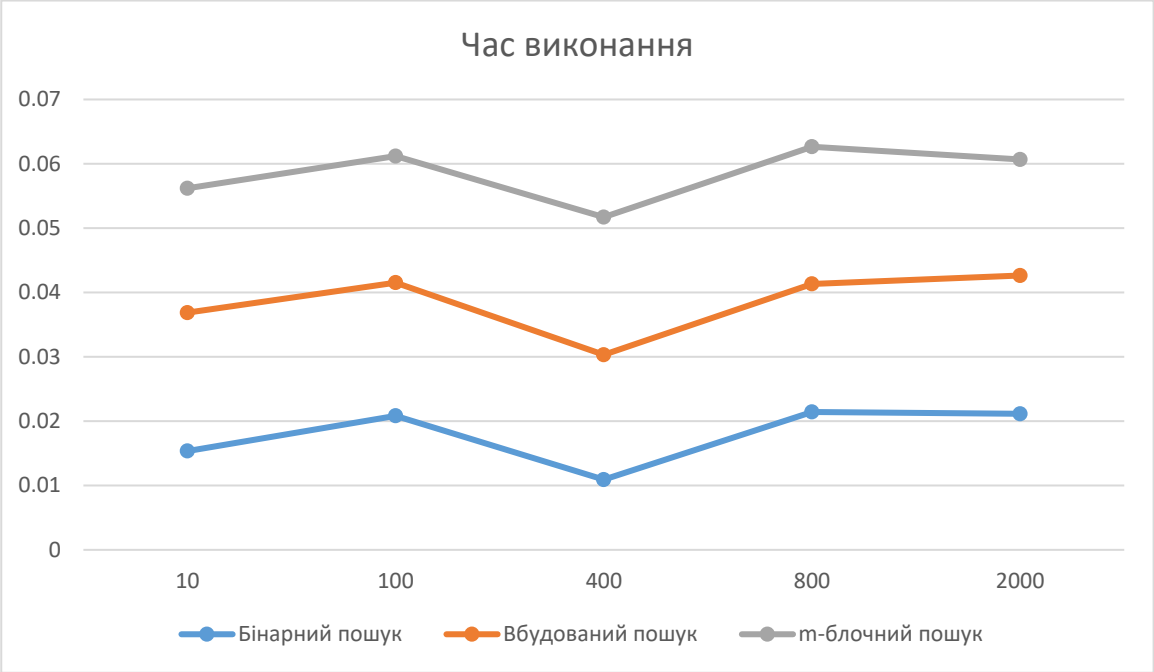
Таблиця 2 – Показники якості m-блочного пошуку

N	10		100		400		800		2000	
m	2	10	2	10	2	10	2	10	2	10
Sr	15	15	146	146	669	669	1071	1071	2523	2523
Time	0.0211	0.0219	0.0204	0.0210	0.0190	0.0199	0.0186	0.0213	0.0226	0.0204
Avg Sr	15		146		669		1071		2523	
Avg Time	0.0215		0.0207		0.0194		0.0199		0.0215	

Таблиця 3 – Показники якості вбудованого пошуку

N	10	100	400	800	2000
Sr	0	0	0	0	0
Time	0.01934	0.01967	0.02139	0.02132	0.01803





Частина 4. Текст програми

Відповідно до розробленого алгоритму в середовищі Microsoft Visual Studio була написана програма, яка наведена нижче.

```
import time
from random import randint

class Array:
    def __init__(self, arr: list):
        self._array = arr.copy()

    @property
    def array(self):
        return self._array

    @array.setter
    def array(self, value):
        self._array = value.copy()

    def __len__(self):
        return len(self._array)

    def __getitem__(self, index):
        return self._array[index]

    def show(self) -> None:
        for i in self.array:
            print('%d ' % i, end='')
        print()
```

```
def generate(self, length: int) -> None:
    MIN, MAX = 0, 1000
    new = []
    for i in range(length):
        new.append(randint(MIN,MAX))
    self.array = new

def binary_search(self, x: int) -> int:
    # array must be sorted!
    op_count = 0

    arr = sorted(self.array)
    low = 0
    high = len(arr) - 1
    mid = 0
    while low <= high:
        op_count += 1
        mid = (high + low) // 2
        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            print(op_count)
            return 1
    print(op_count)
    return 0

def m_block_search(self, x: int, block_size: int = 1) -> int:
    # array must be sorted!
    op_count = 0

    arr = sorted(self.array)
    i = len(arr) - 1
    j = 0
    s = 0
    b = []
    while 1:
        op_count += 1

        b.append(block_size)
        s += b[j]
        if(s >= i): break
```

```
        j = 0
        k = b[j] - 1
        j += 1
        while(arr[k] < x and k <= i):
            op_count += 1
            # print(j, b[j])
            k += b[j]
            j += 1
        if(k > j):
            print(op_count)
            return -1
        z = k - b[j - 1]
        while(z <= k):
            op_count += 1
            z += 1
            if(arr[z] == x):
                print(op_count)
                return 1
        print(op_count)
        return -1

    def built_in_search(self, x: int) -> int:
        print('[INFO] OPERATION COUNT CAN\'NT BE DEFINED!')
        try:
            a = self.array.index(x)
            a = 1
        except ValueError:
            a = 0
        return a

def runtime_decorator(func, blocksize: int = 1):
    arg = int(input('[VALUE] >> '))

    start_time = time.time()

    if func is Array.m_block_search:
        result = func(arg, blocksize)
    else:
        result = func(arg)

    if result in [0, -1]:
        print("Not found")
    else:
        print("Found")
```

```
    end_time = time.time()

    print('Runtime: ', round(end_time - start_time, 5))


def menu() -> None:
    a = """Menu:
    - 0. Show this menu
    - 1. Enter an array
    - 2. Generate an array
    - 3. # Binary search
    - 4. # M-block search
    - 5. # Build-in search
    - 6. Show
    - 7. Show random element and its position (faster)
    - 8. Exit"""
    print(a)

op = 0
arr = Array([])
while 1:
    match op:
        case 0:
            menu()
        case 1:
            t = map(int, input("[ARR] >> ").split(" "))
            arr = list(t)
            print('[OK]')
        case 2:
            l = int(input('[LEN] >> '))
            arr.generate(l)
            print('[OK]')
        case 3:
            runtime_decorator(arr.binary_search)
        case 4:
            m = input('[Block size] >> ')
            runtime_decorator(arr.m_block_search, m)
        case 5:
            runtime_decorator(arr.built_in_search)
        case 6:
            arr.show()
            print('[OK]')
        case 7:
```

```
            r = randint(0, len(arr)-1)
            print(r)
            print(f'Element: {arr[r]}\nIndex: {r}')
        case 8:
            print("[OK] Exiting...")
            exit()
        # Else
        case _:
            print("[ERR] An invalid operation!")
    op = int(input('>> '))
```

Частина 5. Скриншоти роботи на прикладі малого масиву
Довжина: 10
Масив: [492 435 479 595 48 545 325 477 380 240]
Відсортований масив: [48 240 325 380 435 477 479 492 545 595]

Числа для пошуку: 492, 48

```
Menu:
    - 0. Show this menu
    - 1. Enter an array
    - 2. Generate an array
    - 3. # Binary search
    - 4. # M-block search
    - 5. # Build-in search
    - 6. Show
    - 7. Show random element and its position (faster)
    - 8. Exit
>> 2
[LEN] >> 10
[OK]
>> 6
492 435 479 595 48 545 325 477 380 240
[OK]
```

Бінарний пошук

```
>> 3
[VALUE] >> 492
2
Found
Runtime:  0.01117
>>
```

```
>> 3
[VALUE] >> 48
3
Found
Runtime:  0.01241
>>
```

М-блочний пошук

```
>> 4
[Block size] >> 1
[VALUE] >> 492
17
Found
Runtime:  0.01937
>>
```

```
>> 4
[Block size] >> 2
[VALUE] >> 492
17
Found
Runtime:  0.01428
>>
```

```
>> 4
[Block size] >> 5
[VALUE] >> 492
17
Found
Runtime:  0.02097
>>
```

```
>> 4
[Block size] >> 1
[VALUE] >> 48
10
Found
Runtime:  0.01921
```

```
>> 4
[Block size] >> 2
[VALUE] >> 48
10
Found
Runtime:  0.01874
```

```
>> 4
[Block size] >> 5
[VALUE] >> 48
10
Found
Runtime:  0.01927
```

Вбудований пошук

```
>> 5
[VALUE] >> 492
[INFO] OPERATION COUNT CAN'NT BE DEFINED!
Found
Runtime:  0.01938
```

```
>> 5
[VALUE] >> 48
[INFO] OPERATION COUNT CAN'NT BE DEFINED!
Found
Runtime:  0.01939
```

Висновки

Під час цієї лабораторної роботи я вивчав різні алгоритми пошуку та реалізовував їх на практиці, прочитавши багато матеріалу я зміг зрозуміти принципи їх роботи та особливості кожного. Наприклад бінарний пошук є дуже швидким, але потребує відсортованого масиву, як і m-блочний пошук.

Моє розуміння про бінарний пошук: він бере мінімальну, максимальну позицію і вираховує їх середнє значення, якщо шукане значення не дорівнює середньому і більше за нього, то змінюється мінімальна позиція і виконується розрахунок середньої, якщо ж менше – змінюється максимальна позиція і так

до тих пір поки не знайдеться елемент або поки не позиції не почнуть перекривати одна одну.

Моє розуміння про m-блочний пошук: масив розбивається на блоки довжиною в m, масив пробігається по цим блокам і порівнює значення. Плюс цього пошуку є його швидкість, якщо правильно підібрати довжину блоку.