МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут комп'ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

ЗВІТ

до лабораторної роботи №10

на тему: «Бінарний пошук в упорядкованому масиві.»

з дисципліни: «Алгоритми і структури даних»

Лектор:
доц. кафедри ПЗ
Коротєєва Т. О.

Виконав:
ст. гр. ПЗ-22
Чаус О. М.

Прийняв:
асист. кафедри ПЗ
Франко А. В.

« ____ » _____ 2022 р.

∑= _____ ……:

Львів-2022

Тема роботи: Бінарний пошук в упорядкованому масиві.
Мета роботи: навчитися застосовувати алгоритм бінарного пошуку при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму.

Теоретичні відомості

Бінарний, або двійковий пошук – алгоритм пошуку елементу у відсортованому масиві. Це класичний алгоритм, ще відомий як метод дихотомії (ділення навпіл).

Якщо елементи масиву впорядковані, задача пошуку суттєво спрощується. Згадайте, наприклад, як Ви шукаєте слово у словнику. Стандартний метод пошуку в упорядкованому масиві – це метод поділу відрізка навпіл, причому відрізком є відрізок індексів 1..n. Дійсно, нехай масив A впорядкований за зростанням і m (k < m < l) – деякий індекс. Нехай Buffer = A[m]. Тоді якщо Buffer > b, далі елемент необхідно шукати на відрізку k..m-1, а якщо Buffer < b – на відрізку m+1..l.

Для того, щоб збалансувати кількість обчислень в тому і іншому випадку, індекс m необхідно обирати так, щоб довжина відрізків k..m, m..l була (приблизно) рівною. Описану стратегію пошуку називають бінарним пошуком.

b – елемент, місце якого необхідно знайти. Крок бінарного пошуку полягає у порівнянні шуканого елемента з середнім елементом Buffer = A[m] в діапазоні пошуку [k..l]. Алгоритм закінчує роботу при Buffer = b (тоді m – шуканий індекс). Якщо Buffer > b, пошук продовжується ліворуч від m, а якщо Buffer < b – праворуч від m. При l < k пошук закінчується, і елемент не знайдено.

Індивідуальне завдання

Використовуючи алгоритм бінарного пошуку, знайдіть елемент b у масиві A з кількістю елементів від 10 до 1000, розташованих за зростанням.

1. Програма повинна забезпечувати автоматичну генерацію масиву цілих чисел (кількість елементів масиву вказується користувачем) та виведення його на екран;
2. Визначте кількість порівнянь та порівняйте ефективність на декількох масивах різної розмірності заповнивши табл. 1.
3. Представте покрокове виконання алгоритму пошуку.
4. Побудуйте графік залежності кількості порівнянь від кількості елементів масиву у Excel. Побудуйте у тій же системі координат графіки функцій y=n та y=log2(n). Дослідивши графіки, зробіть оцінку кількості C(n) порівнянь алгоритму бінарного пошуку.
5. З переліку завдань виконайте індивідуальне завдання запропоноване викладачем.

Варіант 13: Дано два одновимірні масиви цілих чисел A[i] та B[i], де i =1,2,…,n. Сформувати новий масив C, який складатиметься з парних елементів з масиву A та непарних елементів з масиву B. Знайти елементи масиву C, які при діленні на 5 мають остачу 2.

Код програми

Файл **binary_search.py**

```
from rich import print as rprint

class BinarySearch:
    count = 0
    @staticmethod
    def binary_search(arr: list, low: int, high: int) -> tuple[int, int]:
        rprint(("-" * 80), "\nLower bound binary search:")
        start = BinarySearch.lower_bound(arr, low, high)
        rprint(("-" * 80), "\nUpper bound binary search:")
        end = BinarySearch.upper_bound(arr, low, high)
        end -= 1
        rprint(f"Start: {start},\nEnd: {end}.")
        rprint(arr[start:(end + 1)])
        rprint(f"Lower bound search total comparisons:
{BinarySearch.count}.")
        return (start, end)
    @staticmethod
    def lower_bound(arr: list, low: int, high: int) -> int:
        if high > low:
            mid = (high + low) // 2
            BinarySearch.print_array(arr, low, high, colored=mid, idx=low)
            BinarySearch.count += 1
            if 2 > arr[mid] % 5:
                return BinarySearch.lower_bound(arr, mid + 1, high)
            return BinarySearch.lower_bound(arr, low, mid)
        BinarySearch.print_array(arr, low, low, colored=low, idx=low)
        return low
    @staticmethod
    def upper_bound(arr: list, low: int, high: int) -> int:
        if high > low:
            mid = (high + low) // 2
            BinarySearch.print_array(arr, low, high, colored=mid, idx=low)
            if 2 >= arr[mid] % 5:
                return BinarySearch.upper_bound(arr, mid + 1, high)
            return BinarySearch.upper_bound(arr, low, mid)
        BinarySearch.print_array(arr, low, low, colored=low, idx=low)
        return low
    @staticmethod
    def sort(array: list) -> None:
        array.sort(key=lambda x: x % 5)
    @staticmethod
    def print_array(array: list, low: int, high: int, colored=-1, idx=0) ->
None:
        rprint(("    " * idx) + "[", end="")
        for i in range(low, high + 1):
            rprint(f"[white]{array[i] % 5:3}[/white]"
                   if i != colored else f"[green]{array[i] % 5:3}[green]",
end=" ")
        rprint(" ]")
```

Файл **main.py**

```python
import random
import binary_search as bs

def main() -> None:
    size = int(input("Please enter the size of the array: "))
    arr_A = []
    arr_B = []
    arr_C = []
    for _ in range(size):
        n = random.randint(0, 999)
        arr_A.append(n)
    for _ in range(size):
        n = random.randint(0, 999)
        arr_B.append(n)

    for a, b in zip(arr_A, arr_B):
        if not a % 2:
            arr_C.append(a)
        if b % 2:
            arr_C.append(b)
    print("-" * 80)
    print("Generated array A:\n" + str(arr_A))
    print("Generated array B:\n" + str(arr_B))
    print("Generated array C:\n" + str(arr_C))
    bs.BinarySearch.sort(arr_C)
    print("Sorted array:\n" + str(arr_C))
    bs.BinarySearch.binary_search(arr_C, 0, len(arr_C) - 1)


if __name__ == "__main__":
    main()
```

Зображення програми

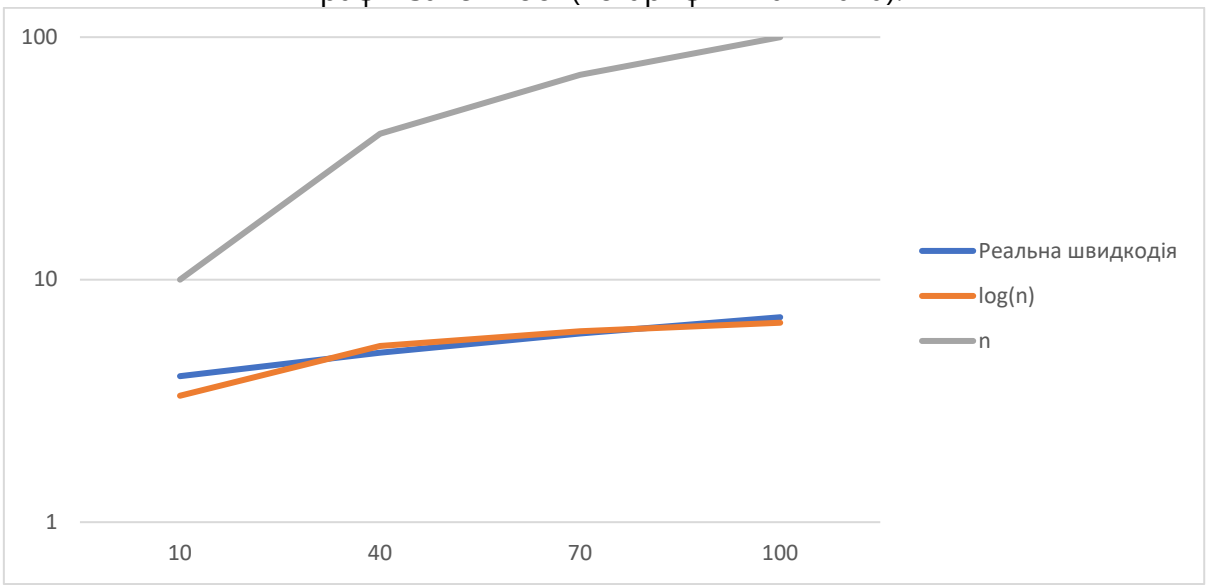
```
Enter the size of the array: 15
--------------------------------------------------------------------------------
Generated array A:
[841, 888, 562, 5, 849, 64, 113, 121, 0, 118, 284, 169, 409, 250, 791]
Generated array B:
[112, 686, 429, 745, 251, 338, 564, 918, 390, 33, 852, 510, 691, 353, 954]
Generated array C:
[888, 562, 429, 745, 251, 64, 0, 118, 33, 284, 691, 250, 353]
Sorted array:
[745, 0, 250, 251, 691, 562, 888, 118, 33, 353, 429, 64, 284]
--------------------------------------------------------------------------------
Lower bound binary search:
[  0   0   0   1   1   2   3   3   3   3   4   4   4  ]
[  0   0   0   1   1   2   3  ]
                [  1   2   3  ]
                [  1   2  ]
                    [  2  ]
--------------------------------------------------------------------------------
Upper bound binary search:
[  0   0   0   1   1   2   3   3   3   3   4   4   4  ]
[  0   0   0   1   1   2   3  ]
                [  1   2   3  ]
                        [  3  ]

Start: 5,
End: 5.
[562]
Lower bound search total comparisons: 4.
```

Таблиця кількості елементів та кількості порівнянь:

Кількість елементів	Кількість порівнянь
10	4
40	5
70	6
100	7

Графік залежності(логарифмічна шкала):



Висновок:

навчився застосовувати алгоритм бінарного пошуку при розв’язуванні задач та перевірив його ефективність на різних масивах даних. Експериментально визначив складність алгоритму, що дорівнює $O(\log_2 n)$