МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Інститут **КНІТ**
Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи № 10
На тему: "*Бінарний пошук в упорядкованому масиві*"
З дисципліни: "Алгоритми та структури даних"

Лектор:
доцент кафедри ПЗ
Коротєєва Т.О.

Виконав:
студент групи ПЗ-22
Коваленко Д.М.

Прийняв:
асистент кафедри ПЗ
Франко А.В.

«_____» _____ 2022 р.
$\sum$ = _____.....

Львів — 2022

Тема. Бінарний пошук в упорядкованому масиві.

Мета. Навчитися застосовувати алгоритм бінарного пошуку при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму.

Лабораторне завдання

Розробити програму, яка:

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 (n) порівнянь алгоритму бінарного пошуку.

5. З переліку завдань виконайте індивідуальне завдання запропоноване викладачем.
Варіант 2: В масиві A[i], де i =20,21,. . . ,n. зберігається інформація про вартість кожної з A[i] книг. Визначити назву книжки ввівши її ціну.

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.

Хід роботи

```
use fake::{faker::lorem::en::*, Fake, Dummy};

#[derive(Debug, Clone, PartialEq, Eq, Ord, Dummy)]
pub struct Book {
    #[dummy(faker = "Word()")]
    name: String,
    #[dummy(faker = "20..100")]
    price: usize,
}

#[derive(Debug)]
pub struct Arr {
    arr: Vec<Book>,
    len: usize,
}

impl Arr {
```

```
    pub fn new(len: usize) -> Self {
        let mut arr = fake::vec![Book; len];
        arr.sort();
        Self { arr, len }
    }

    pub fn search(&self, price: usize) -> (Option<String>, usize) {
        let mut left_i = 0;
        let mut right_i = if let Some(res) = self.len.checked_sub(1) { res }
    else { return (None, 0) };
        let mut cmp_counter = 0;

        while left_i <= right_i {
            let mid = (left_i + right_i) / 2;
            print!("Left: {: <3} Mid: {: <3} Right: {: <3}\t", left_i, mid,
    right_i);
            match self.arr[mid].price {
                v if v < price => {
                    cmp_counter += 1;
                    println!("{} < {}", v, price);
                    left_i = mid + 1;
                },
                v if v > price => {
                    cmp_counter += 1;
                    println!("{} > {}", v, price);
                    right_i = if let Some(res) = mid.checked_sub(1) { res } else
     { break; };
                },
                v => {
                    println!("{} == {}", v, price);
                    return (Some(self.arr[mid].name.clone()), cmp_counter);
                }
            }
        }
        (None, cmp_counter)
    }

    pub fn print(&self) {
        for i in &self.arr {
            println!("{: <20} {}", i.name, i.price);
        }
    }
}

impl PartialOrd for Book {
    fn partial_cmp(&self, other: &Self) -> Option<std::cmp::Ordering> {
        self.price.partial_cmp(&other.price)
    }
}
```

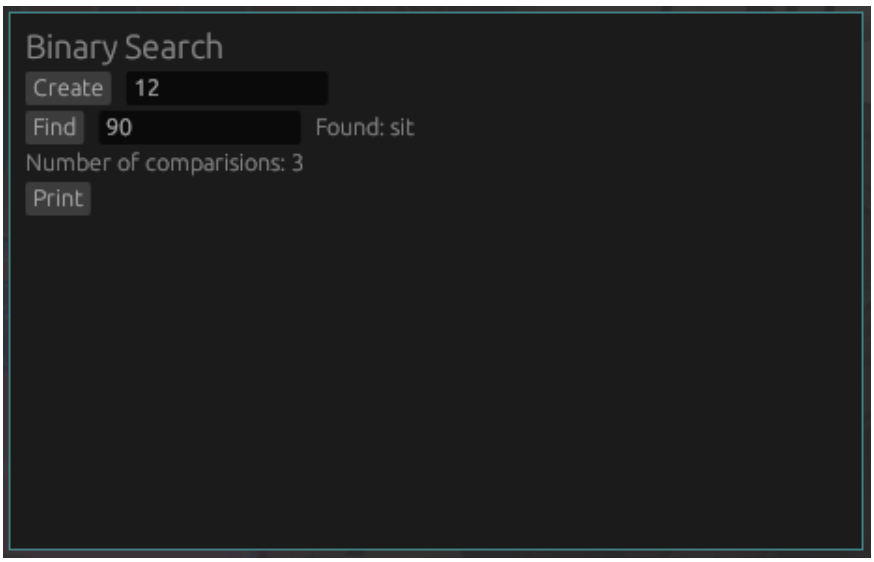

Рис. 1: Вигляд програми

```
doloremque          35
est                 37
vel                 46
eveniet             49
sapiente            65
ut                  70
minima              71
maiores             79
molestiae           87
sit                 90
rerum               91
voluptatem          99
Left: 0   Mid: 5   Right: 11    70 < 90
Left: 6   Mid: 8   Right: 11    87 < 90
Left: 9   Mid: 10  Right: 11    91 > 90
Left: 9   Mid: 9   Right: 9     90 == 90
```

Рис. 2: Покроковий вивід

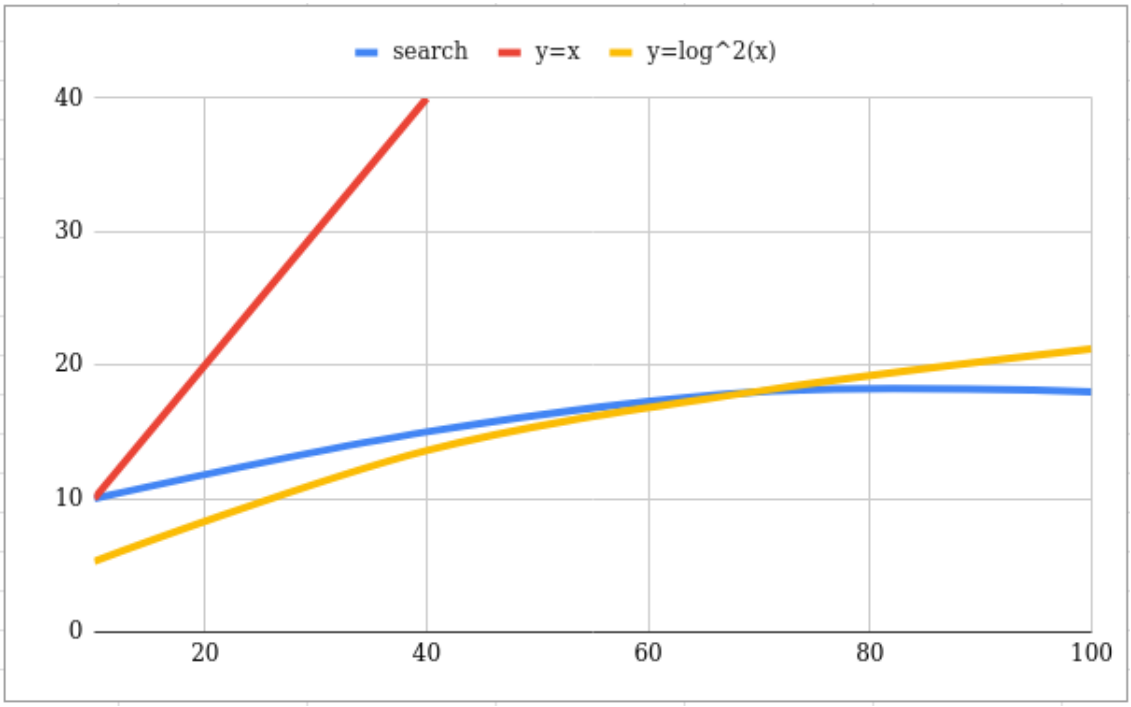

Рис. 3: Діаграма

Висновоки

Під час виконання лабораторної роботи я навчився застосовувати алгоритм бінарного пошуку при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначив складність алгоритму.