Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА № 1

з дисципліни «Алгоритми та методи обчислень»

на тему «Обчислювальна складність алгоритмів сортування»

ВИКОНАЛА:
студентка 2 курсу
групи ІВ-92
Бабенко В.В.
Залікова - 9201

ПЕРЕВІРИВ:
Доцент кафедри ОТ
Порєв В.М.

Київ - 2021

Хід роботи

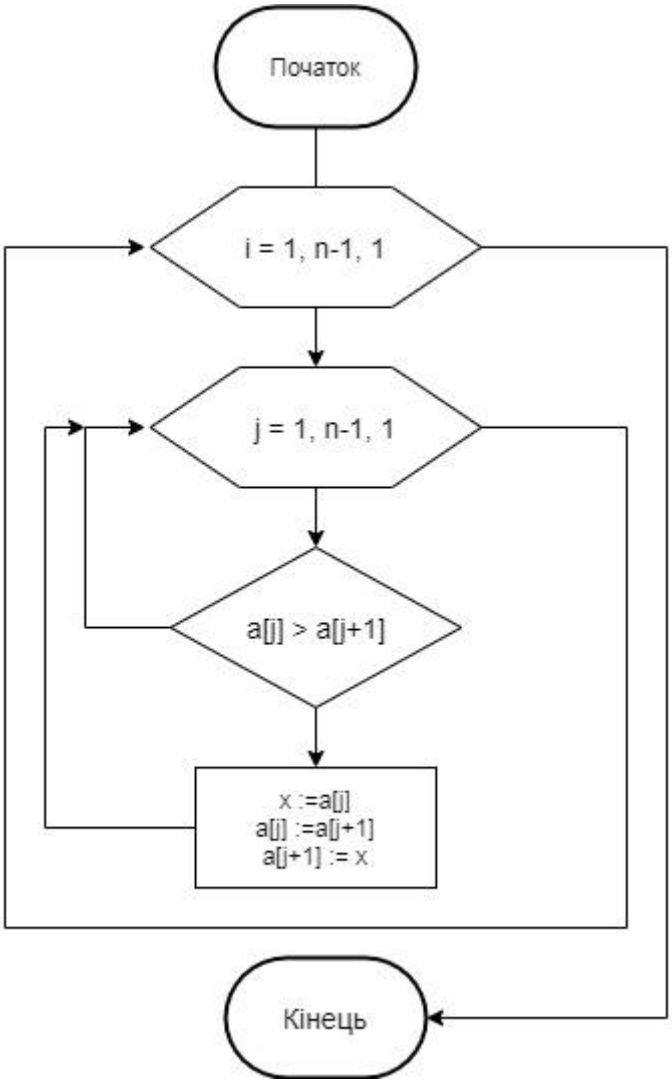
Мета: Закріплення навичок практичної оцінки алгоритмічної складності логічних алгоритмів на прикладі алгоритмів сортування.

Завдання:
Використовуючи відповідний до варіанту алгоритм сортування написати програму сортування масиву даних. Застосовуючи дану програму, дослідити часову складність алгоритму сортування та порівняти її з теоретичною алгоритмічною складністю.

Варіанти завдання

Вар.	Назва алгоритму	Алгоритмічна складність
1	Алгоритм 1.1. Бульбашкове сортування з початку до кінця	$O(^2)$

Блок-схеми алгоритмів



Роздруківка тексту програми

Алгоритм

```
def bubble(array):
    n = len(array)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if array[j] > array[j + 1]:
                buff = array[j]
                array[j] = array[j + 1]
                array[j + 1] = buff
    return array
```

Графічний інтерфейс

```
from tkinter import *
from tkinter import messagebox
import random
from time import time
import matplotlib.pyplot as plot

def bubble(array):
    n = len(array)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if array[j] > array[j + 1]:
                buff = array[j]
                array[j] = array[j + 1]
                array[j + 1] = buff
    return array


def count_sort_time(length: int, step: int = 1):
    from random import randint
    from time import time
    n_time = dict()
    for j in range(0, length + 1, step):
        r = []
        for i in range(0, j):
            new = randint(-100000000001, 100000000001)
            if new not in r:
                r.append(new)
            else:
                i -= 1
        t1 = time()
        bubble(r)
        t2 = time()
        n_time.update([(j, round(t2 - t1, 10))])
    return n_time

class Algorithm_merge:
    def __init__(self):
        self.frame = Frame(root, bg='blue', bd=20)
        self.frame.pack()
```

```
        self.label_name = Label(self.frame, text='Алгоритм сортування
бульбашкою', font=('Garamond', 20), bg='yellow')
        self.label_name.grid(row=1, column=2, columnspan=3)

        self.label_var = Label(self.frame, text='Варіант 1', font=('Garamond',
17), bg='yellow')
        self.label_var.grid(row=2, columnspan=3, column=2)

        self.entry_main = Text(self.frame, width=100, height=3, bd=2,
font=('Garamond', 12))
        self.entry_main.grid(row=4, columnspan=6)

        Label(self.frame, text='\n', font=('Garamond', 10), bg='light
blue').grid(row=5)
        Label(self.frame, text='\n', font=('Garamond', 10), bg='light
blue').grid(row=3)

        self.but1 = Button(self.frame, text='Згенерувати', width=15,
font=('Garamond', 14), bg='yellow',
                           command=self.generate)
        self.but1.grid(row=6, column=1)

        self.but3 = Button(self.frame, text='Зчитати з файлу', width=15,
font=('Garamond', 14), bg='yellow',
                           command=self.read_from_file)
        self.but3.grid(row=6, column=2)

        self.but3 = Button(self.frame, text='Сортувати', width=15,
font=('Garamond', 14), bg='yellow',
                           command=self.sort_but)
        self.but3.grid(row=6, column=3)

        self.but2 = Button(self.frame, text='Порахувати час', width=15,
font=('Garamond', 14), bg='yellow',
                           command=self.but_time)
        self.but2.grid(row=6, column=4)

        self.but4 = Button(self.frame, text='Графік', width=15,
font=('Garamond', 14), bg='yellow',
                           command=self.build_graphics)
        self.but4.grid(row=6, column=5)

        Label(self.frame, text='\n', font=('Garamond', 10), bg='light
blue').grid(row=7)

        self.entry_sort = Text(self.frame, width=100, height=3, bd=2,
font=('Garamond', 12))
        self.entry_sort.grid(row=8, columnspan=6)

        self.length = random.randint(1000, 2000)
        self.arr = []

    def generate(self):
        self.entry_main.delete(1.0, END)
        self.arr = [random.randint(-100000, 100000) for i in range(self.length)]
        self.entry_main.insert(1.0, self.arr)

    def sort_but(self):
        new_arr = []
        arr = self.entry_main.get(1.0, END)
        self.entry_sort.delete(1.0, END)
        if arr == '' or arr == '\n':
            messagebox.showerror('Помилка', 'Перевірте введені дані!')
```

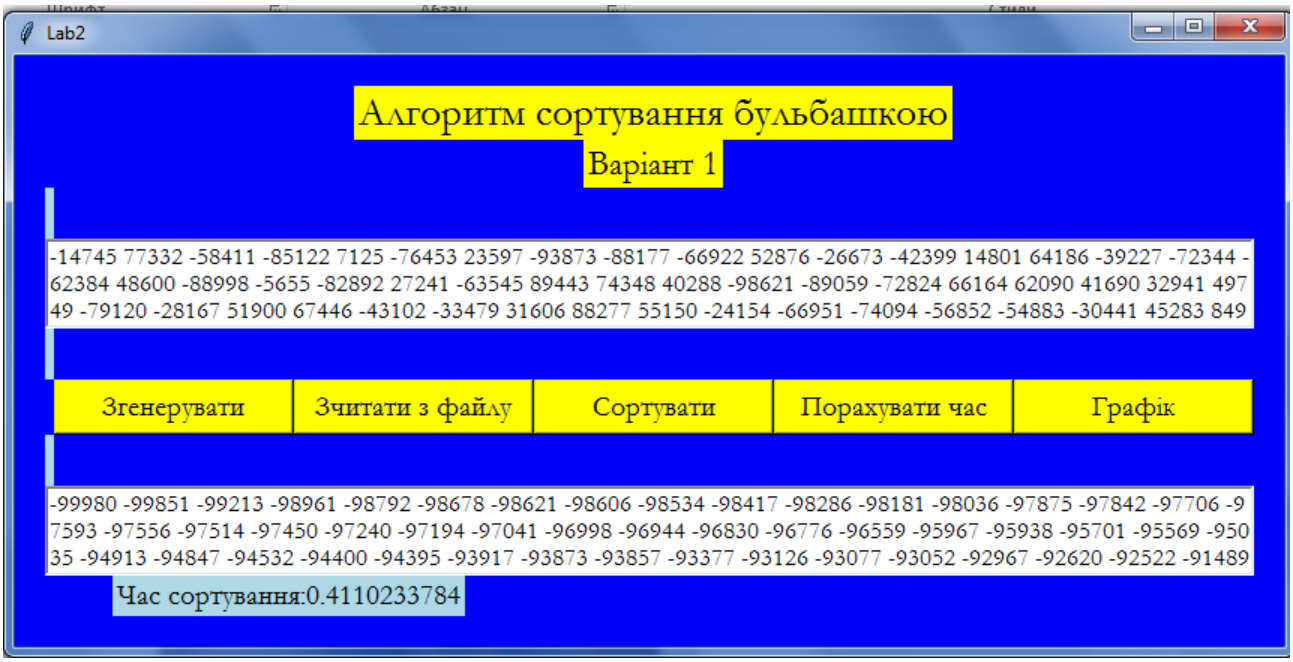
```
        else:
            arr1 = arr.split(' ')
            for i in range(len(arr1)):
                new_arr.append(int(arr1[i]))
            self.entry_sort.insert(1.0, bubble(new_arr))

    def but_time(self):
        t1 = time()
        bubble(self.arr)
        t2 = time()
        sort_time = round(t2 - t1, 10)
        answer = 'Час сортування:' + str(sort_time)
        Label(self.frame, text=answer, font=('Garamond', 14), bg='light
blue').grid(row=9, columnspan=3)

    def build_graphics(self, step: int = 1):
        from random import randint
        from time import time
        n_time = dict()
        for j in range(0, self.length + 1, step):
            r = []
            for i in range(0, j):
                new = randint(-100001, 100001)
                if new not in r:
                    r.append(new)
                else:
                    i -= 1
            t1 = time()
            bubble(r)
            t2 = time()
            n_time.update([(j, round(t2 - t1, 10))])
        d_x = []
        d_y = []
        for k in n_time:
            d_x.append(k)
            d_y.append(n_time[k])
        d_teor_x = []
        d_teor_y = []
        for i in range(self.length + 1):
            y = i * (i ** 2)
            d_teor_x.append(i + 1)
            d_teor_y.append(y)
        fig = plot.figure()
        ax = fig.add_subplot(111, label="1")
        ax2 = fig.add_subplot(111, label="2", frame_on=False)
        ax.plot(d_x, d_y, color="C0")
        ax.set_xlabel("Практично", color="C0")
        ax.tick_params(axis='x', colors="C0")
        ax.tick_params(axis='y', colors="C0")
        ax2.plot(d_teor_x, d_teor_y, color="C1")
        ax2.xaxis.tick_top()
        ax2.yaxis.tick_right()
        ax2.set_xlabel('Теоретично', color="C1")
        ax2.xaxis.set_label_position('top')
        ax2.tick_params(axis='x', colors="C1")
        ax2.tick_params(axis='y', colors="C1")
        plot.show()

    def read_from_file(self):
        with open(r'text.txt', 'r+') as f:
            text = f.read().split(' ')
            f.close()

        for i in range(text.count('')):
```

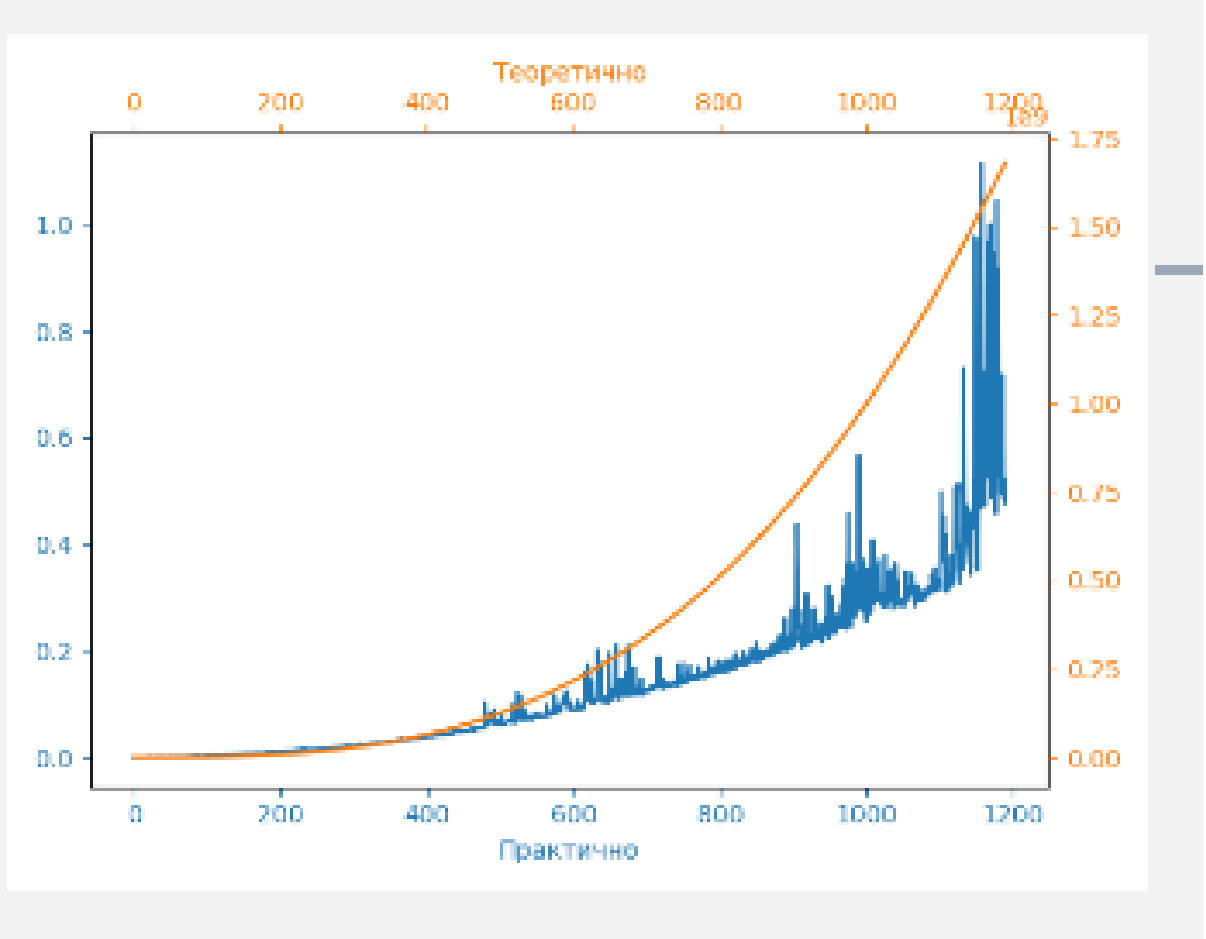
```
            try:
                text.remove('')
            except:
                pass

        for i in range(len(text)):
            text[i] = int(text[i])

        self.entry_main.delete(1.0, END)
        self.entry_main.insert(1.0, text)


root = Tk()
obj = Algorithm_merge()
root.title('Lab2')
root.mainloop()
```

ПРИМІТКА

ПРИ ВВЕДЕННІ ДАННИХ В ФАЙЛ СПОЧАТКУ ЗБЕРЕЖІТЬ ЙОГО, А ПОТІМ ПЕРЕЗАПУСТІТЬ ПРОГРАМУ ТА НАТИСНІТЬ ВІДПОВІДНУ КНОПКУ

Роздруківка результатів виконання програми





Аналіз результатів

Реалізація алгоритму сортування бульбашкою є досить таки не складною, проте під час виконання програми було помічено виконання золотого правила алгоритмів. Для опису алгоритму було побудовано відповідну блок-схему та надано графіки залежності кількості операцій від часу, у практичному та теоретичному баченні.

Висновки

У ході виконання лабораторної роботи були закріплені знання з базових понять алгоритмів, навички практичної оцінки алгоритмічної складності логічних алгоритмів на прикладі алгоритмів сортування Отримані результати виконання програми є правильними.