Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота № 4

з дисципліни «Теорія алгоритмів»

на тему:

**" Метод швидкого сортування "**

Виконала:

студентка гр. ІС-02

Павлущенко Ольга

Викладач:

Новікова П. А.

Київ – 2020

**Завдання**

Реалізувати наступні три модифікації алгоритму швидкого сортування (Quick Sort) та порівняти їх швидкодію. Швидкість алгоритмів порівнюється на основі підрахунку кількості порівнянь елементів масиву під час роботи алгоритмів.

**Алгоритм 1.** Звичайний алгоритм швидкого сортування

**Алгоритм 2. Швидке сортування з 3-медіаною в якості опорного елемента**

**Програмний код**

def main():

input\_file = input("Enter the file name: ") #введення назви файла для відкриття

f = open(input\_file)

arr = f.readlines()

del arr[0]

arr = [[int(n) for n in x.split()]for x in arr] #масив чисел

n = len(arr)

print(arr)

comparisons1 = QuickSort(arr,0,n-1)

comparisons2 = MedianQuickSort(arr, 0, n-1)

print (arr)

file=open("is02\_OlhaPavlushchenko\_04\_output.txt", "w")

file.write(str(comparisons1) + ' ' + str(comparisons2)) #виведення в файл

file.close()

def QuickSort(array, p, r):

if p<r:

q,count1 = Partition(array,p,r) #розділення

count2 = QuickSort(array, p, q-1) #рекурсивно вихиваємо для лівого підмасиву

count3 = QuickSort(array, q+1, r) #рекурсивно вихиваємо для правого підмасиву

else:

return 0

return count1 + count2 + count3

def Partition(array, p, r):

count = 0

x = array[r] #опорний елемент

i = p-1

for j in range(p, r):

count += 1 #кількість порівнянь

if array[j] <= x: #якщо елемент менший за опорний

i += 1

array[i], array[j] = array[j], array[i] #перестановка елементів, що в лівому підмасиві елементи менші за опорний

array[i+1],array[r] = array[r],array[i+1] #перестановка крайнього лівого елемента, який більший за опорний, та опорний

return i+1, count

def MedianQuickSort(array, p, r):

count = 0

n=len(array) #довжнина масиву

j = 0

if p<r:

if n >= 3: #якщо розмір масиву більший рівний 3

q, count1 = MedianPartition(array, p, r)

count2 = MedianQuickSort(array, p, q-1)

count3 = MedianQuickSort(array, q+1, r)

else: #якщо розмір менший 3, порівнюємо елементи між собою

if array[p] > array[r]:

array[p], array[r] = array[r], array[p]

return 1

else:

return 0

return count1+count2+count3

def MedianPartition(array, p, r): #визначення медіани

m = int((p+r)/2) #середній елемент(індекс)

if array[m] <= array[p] <= array[r] or array[r] <= array[p] <=array[m]: #пошук середнього за значенням елементу

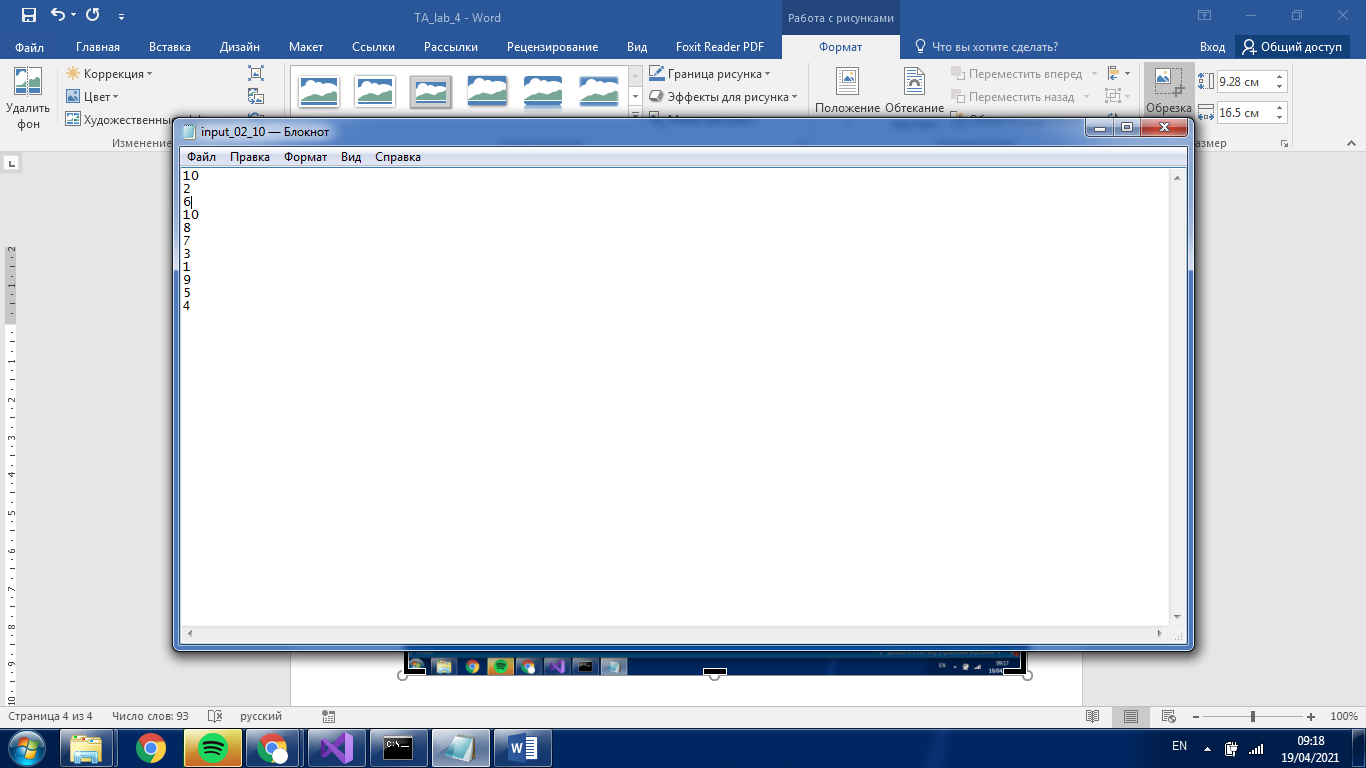
array[m],array[p] = array[p],array[m]

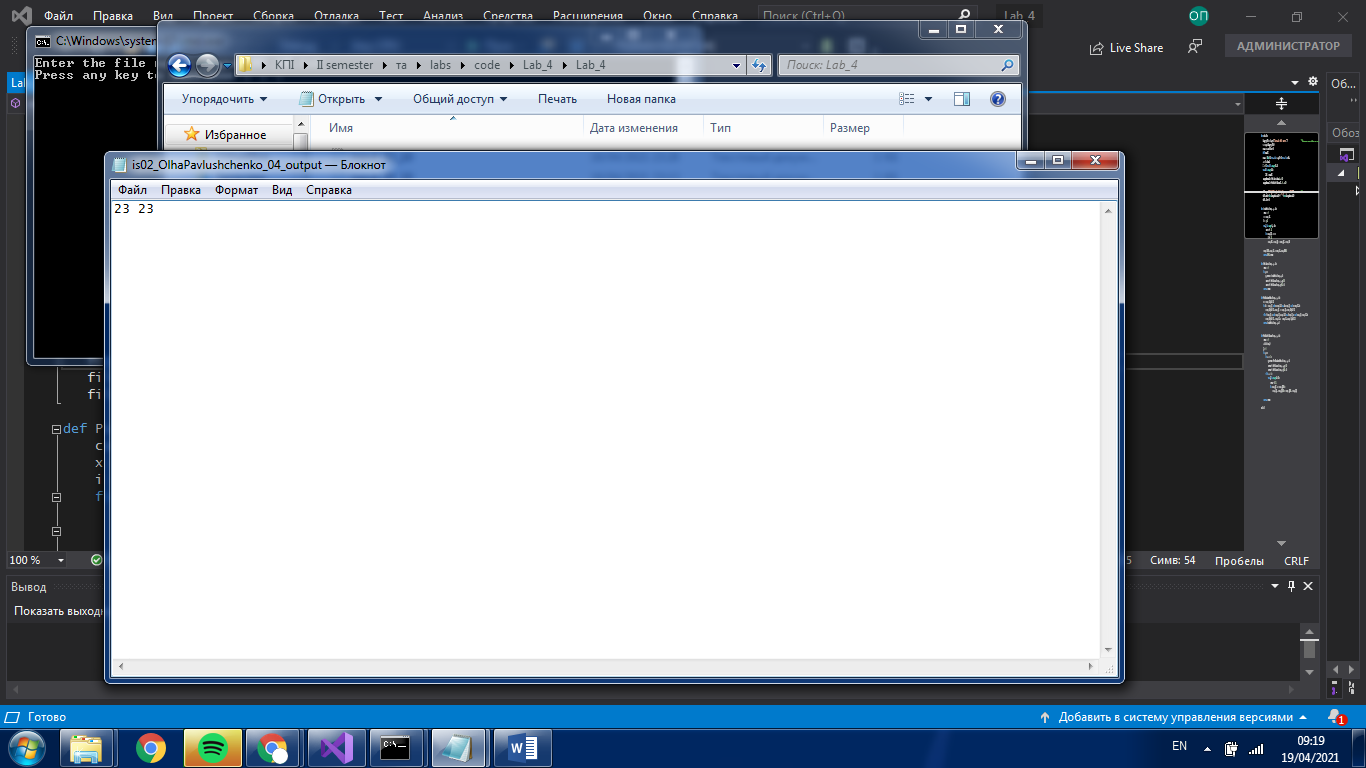
elif array[p] <= array[m] <= array[r] or array[r] <= array[m] <= array[p]:

array[m], array[r] = array[r],array[m]

return Partition(array, p,r)

main()

Результати



**Висновок**

У ході лабораторної роботи було реалізовано 2 модифікації алгоритму QuickSort та порівнювалась їхня швидкість роботи. Вибір медіани з 3-х елементів: першого, середнього та останнього – знижує ймовірність виникнення гіршого випадку, коли опорний елемент –найбільше чи найменше значення (можливо ще, коли масив є вже відсортованим). В такому випадку ефективність *O*(*n*2). При розбитті масиву на дві приблизно однакові частини складність алгоритму складає  *O*(nlogn). В найкращому випадку складність алгоритму складає  *O*(nlogn), коли масив розбивається на дві однакові частини{\displaystyle O(n\cdot \log \_{2}n)}

