Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота № 3

з дисципліни «Теорія алгоритмів»

на тему:

**"Метод швидкого сортування"**

Виконав:

студент гр. ІС-02

Плостак Ілля

Викладач:

Cт. Вик. Новікова П. А.

Київ – 2021

**1. Завдання**

Реалізувати наступні дві 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

**2. Програмний код:**

[main.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab4/main.py):

import os  
import random  
import algorithms as alg  
  
# uncomment to create a random input.txt  
#import randomgen  
  
if (os.path.exists("input.txt") == False):  
 print("Error accessing the file!")  
 exit(1)  
  
fileIn = open("input.txt", 'r')  
massLen = 0  
mass = []  
for i, line in enumerate(fileIn):  
 temp = line.split(" ")  
 map\_temp = map(int, temp)  
 temp = list(map\_temp)  
 if i == 0:  
 massLen = temp[0]  
 else:  
 mass.append(temp[0])  
  
fileIn.close()  
  
if massLen != len(mass):  
 print("Wrong array length!")  
 exit(1)  
  
mass1 = mass.copy()  
counter1 = alg.QuickSort(mass1, 0, len(mass1) - 1)  
print(mass1, counter1)  
  
mass2 = mass.copy()  
counter2 = alg.MedQuickSort(mass2, 0, len(mass2) - 1)  
print(mass2, counter2)  
  
fileOut = open("output.txt", 'w')  
fileOut.write(str(counter1) + " " + str(counter2))  
fileOut.close()

[algorithms.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab4/algorithms.py):

import math  
  
def Partition(A, p, r): # функция разделения подмассива пополам  
 x = A[r] # выбираем опорный элемент (pivot)  
 i = p # инициализируем итератор для левой части подмасива  
 counter = 0 # инициализируем текущий счётчик сравнений  
 for j in range(p, r): # от начала подмасива до конца, кроме последнего, то есть опорного, элемента  
 counter += 1 # увеличиваем каунтер на 1 перед сравнением  
 if A[j] <= x: # если текущий элемент меньше ключевого  
 A[i], A[j] = A[j], A[i] # отправляем его в левую часть подмасива на i-тое место  
 i += 1 # увиличиваем итератор левой части  
 A[i], A[r] = A[r], A[i] # ставим опорный элемент в начало правой части как разделяющий  
 # между правой (большей) и левой (меньшей) частями  
 return i, counter # возвращаем индекс опорного элемента и текущее количество сравнений  
   
def QuickSort(A, p, r): # функция быстрой сортировки  
 counter = 0 # инициализируем текущий счётчик сравнений  
 if p < r: # если индекс конца больше индекса начала  
 q, temp\_counter = Partition(A, p, r) # делим подмасив пополам функцией Partition  
 counter += temp\_counter # прибавляем количество сравнений на текущем этапе  
 counter += QuickSort(A, p, q-1) # рекурсивно вызываем быструю сортировку для левой части подмасива  
 counter += QuickSort(A, q+1, r) # рекурсивно вызываем быструю сортировку для правой части подмасива  
 return counter # возвращаем текущее количество сравнений  
  
def GetMedian(a, b, c):  
 if a > b:  
 if a < c:  
 return a  
 elif b > c:  
 return b  
 else:  
 return c  
 else:  
 if a > c:  
 return a  
 elif b < c:  
 return b  
 else:  
 return c  
  
def MedPartition(A, p, r): # функция разделения подмассива пополам  
 q = math.floor((p+r)/2) # индекс элемента посередине  
 median = GetMedian(A[p], A[q], A[r]) # находим медиану из трёх элементов: последнего, первого и того что посередине  
 medIndex = p # находим индекс медианы  
 if median == A[q]: # ...  
 medIndex = q # ...  
 elif median == A[r]: # ...  
 medIndex = r # ...  
 A[r], A[medIndex] = A[medIndex], A[r] # перемещаем медиану на последнее место, а последний элемент - на место медианы  
 # затем выполняем обычный Partition  
 x = A[r] # выбираем опорный элемент (pivot)   
 i = p # инициализируем итератор для левой части подмасива   
 counter = 0 # инициализируем текущий счётчик сравнений   
 for j in range(p, r): # от начала подмасива до конца, кроме последнего, то есть опорного, элемента   
 counter += 1 # увеличиваем каунтер на 1 перед сравнением   
 if A[j] <= x: # если текущий элемент меньше ключевого   
 A[i], A[j] = A[j], A[i] # отправляем его в левую часть подмасива на i-тое место   
 i += 1 # увиличиваем итератор левой части   
 A[i], A[r] = A[r], A[i] # ставим опорный элемент в начало правой части как разделяющий   
 # между правой (большей) и левой (меньшей) частями   
 return i, counter # возвращаем индекс опорного элемента и текущее количество сравнений   
  
def MedQuickSort(A, p, r): # функция быстрой сортировки   
 counter = 0 # инициализируем текущий счётчик сравнений  
 if p < r: # если индекс конца больше индекса начала   
 if r - p >= 3: # если подмасив длиной 3 или больше  
 q, temp\_counter = MedPartition(A, p, r) # делим подмасив пополам функцией MidPartition с использованием медианы  
 counter += temp\_counter # прибавляем количество сравнений на текущем этапе  
 counter += MedQuickSort(A, p, q - 1) # рекурсивно вызываем быструю сортировку для левой части подмасива  
 counter += MedQuickSort(A, q + 1, r) # рекурсивно вызываем быструю сортировку для правой части подмасива  
 else: # иначе  
 q, temp\_counter = Partition(A, p, r) # делим подмасив пополам обычной функцией Partition  
 counter += temp\_counter # прибавляем количество сравнений на текущем этапе   
 counter += MedQuickSort(A, p, q - 1) # рекурсивно вызываем быструю сортировку для левой части подмасива   
 counter += MedQuickSort(A, q + 1, r) # рекурсивно вызываем быструю сортировку для правой части подмасива  
 return counter # возвращаем текущее количество сравнений

[randomgen.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab4/randomgen.py):

import os  
import random  
  
size = 10  
  
# changing size of the array randomly to 10, 100 or 1000  
choose = random.randint(1, 3)  
for i in range(1, choose):  
 size \*= 10  
size += 1  
  
mass = [i for i in range(1, size)]  
random.shuffle(mass)  
  
fileIn = open("input.txt", 'w')  
  
for i in range(-1, len(mass)):  
 if i == -1:  
 fileIn.write(str(len(mass)))  
 else:  
 fileIn.write(str(mass[i]))  
 if i != len(mass) - 1:  
 fileIn.write("\n")  
  
fileIn.close()

**3. Приклад вхідного файлу, результат запуску програми та приклад вихідного файлу:**

Приклад вхідного файлу:



Результат запуску програми для перевірки чи сортують розроблені алгоритми:



Приклад вихідного файлу:



**4. Висновок:**

В цій лабораторній роботі ми вивчили швидкий алгоритм сортування (quick sort). Цей алгоритм має середню складність Θ(n\*log(n)) та не використовує додаткової пам’яті. Ми модифікували цей алгоритм і зробили швидке сортування з 3-медіаною в якості опорного елемента. Цей алгоритм дозволяє скоротити кількість порівнянь за допомогою визначення медіани з останнього, середнього та початкового елементів підмасиву на кожному кроці рекурсивного виклику функції Partition.

Вивід та ввід у програмі здійснюється через файли.