Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Мегафакультет трансляционных информационных технологий

Факультет инфокоммуникационных технологий

Дисциплина: Алгоритмы и структуры данных

**Отчет по Лабораторной работе №3**

Выполнила: Микулина Алиса Романовна

Группа: K3143, 1 курс

Преподаватель: Харьковская Татьяна Александровна

Санкт-Петербург

10.09.2021

**Описание задания**

**Задание 1.**

1. Написать программу быстрой сортировки на Python и проверить ее, создав несколько рандомных массивов, подходящих под параметры.
2. Реализовать реализацию рандомизированного алгоритма быстрой сортировки, чтобы она работала быстро даже с последовательностями, содержащими много одинаковых элементов. Чтобы заставить алгоритм быстрой сортировки эффективно обрабатывать последовательности с несколькими уникальными элементами, нужно заменить двухстороннее разделение на трехстороннее

**Задание 2.**

Написать программу, которая будет из отсортированной в порядке возрастания последовательности от 1 до n делать последовательность, при которой Quick\_sort будет делать наибольшее число сравнений. Результат проверяется по ссылке: <https://ctlab.itmo.ru/~mbuzdalov/antiqs.html>

**Задание 5.**

Написать код, который будет считать индекс Хирша ученого по количеству цитат на его статьях.

**Задание 6.**

Написать код для сортировки большого количества неотрицательных целых чисел. Дается два массива, размером n и m соответственно. Элементы массива, который нужно отсортировать, являются результатами попарного перемножения элементов двух массивов между собой. Вывести необходимо сумму каждого 10 элемента отсортированной последовательности.

**Задание 8.**

Написать код, ищущий K ближайших точек к началу координат среди данных n точек.

**Описание решения и исходный код**

**Задача 1.**

В задании сказано переписать код, используя псевдокод из лекции. Мне было интересно разобраться самой, как работает quick-sort, поэтому я написала свой. В нем одна функция, вместо двух. По факту оптимизация с тройным разделением для элементарная. Просто добавляется elif для сравнения среднего элемента с числами, которые мы перебираем при распределении на два массива. Все элементы, равные сепаратору, записываются в отдельную переменную и таким образом и хранятся.

import random  
  
def quick\_sort(array):  
 if len(array) == 0:  
 return array  
 length = len(array)  
 separator\_start = random.randint(0, length - 1)  
 separator\_elem = array[separator\_start]  
 smaller, bigger, separator = [], [], []  
 for i in range(0, length):  
 if array[i] < separator\_elem:  
 smaller.append(array[i])  
 elif array[i] == separator\_elem:  
 separator.append(array[i])  
 else:  
 bigger.append(array[i])  
 return [\*quick\_sort(smaller), \*separator, \*quick\_sort(bigger)]  
  
f = open("input", "r")  
length = int(f.readline())  
array = list(map(int, f.readline().split(" ")))  
f.close()  
  
d = open("output", "w")  
d.write(' '.join(list(map(str,quick\_sort(array)))))  
d.close()

**Задача 2.**

В этой задаче все предельно просто, буквально на одну строку. Мы создаем массив от 1 до n и пускаем по нему анти-сортировку (которую почему-то легче писать, чем сортировку, ну да и ладно). Весь смысл в том, что мы пробегаемся по массиву, начиная с 3-го элемента. Поочередно меняя элементы местами друг с другом, а точнее, те, что сортировка бы наоборот меняла друг с другом местами. Этого мы добиваемся простой конструкцией, по замене элемента на i-том месте, и на месте с индексом i // 2. Таким образом, у нас получается наихудший вариант развития событий, что нам, собственно, и нужно.

def anti\_quick\_sort(array):  
 for i in range(2, len(array)):  
 array[i // 2], array[i] = array[i], array[i // 2]  
 return array  
  
with open("input.txt", "r") as f:  
 length = int(f.readline())  
  
arr = [i for i in range(1, length + 1)]  
anti\_list = anti\_quick\_sort(arr)  
  
with open("output.txt", "w") as d:  
 d.write(' '.join(list(map(str,anti\_list))))

**Задача 5.**

Та самая задача, в названии которой непонятное слово, и поэтому страшно, а на деле решила ни одного раза не отдебажив, настолько все просто. Логика весьма проста, и для того, чтобы она работала, нужно отсортировать массив с цитатами по убыванию (я это делаю count-sortом, предполагая, что у многих статей будет одинаковое количество цитат). Затем просто ищем, до какого момента значение элемента больше, чем его индекс. Тадаааам! Все работает.

def count\_sort(massive):  
 low, high = min(massive), max(massive)  
 counts = [0 for i in range(max(massive)+1)]  
 ans = []  
 for i in range(low, high+1):  
 for char in massive:  
 if char == i:  
 counts[i] += 1  
  
 for i in range(len(counts) - 1, -1, -1):  
 if i >= 0:  
 for j in range(counts[i]):  
 ans.append(i)  
 return ans  
  
def hirsh\_index(citations):  
 citations = count\_sort(citations)  
 i = 0  
 while citations[i] > i:  
 i += 1  
 return i  
  
f = open("input.txt", "r")  
citations = list(map(int, f.readline().split(" ")))  
f.close()  
  
d = open("output.txt", "w")  
d.write(str(hirsh\_index(citations)))  
d.close()

**Задача 6.**

Еще одна задачка, в которой quick-sort облегчает жизнь. Все невероятно просто. Сначала циклически перемножает элементы массивов попарно друг с другом, а потом счастливые направляем полученный массив сортироваться в уже всеми нами любимую функцию быстрой сортировки (еще и оптимизированную, вау). Получаем отсортированный массив, пробегаемся по нему счетчиком с шагом в 10 элементов и радуемся жизни.

import random  
  
def integer\_sort(arr\_1, arr\_2):  
 multiplied\_array = []  
 len\_1 = len(arr\_1)  
 len\_2 = len(arr\_2)  
 for i in range(len\_1):  
 for j in range(len\_2):  
 multiplied\_array.append(arr\_1[i] \* arr\_2[j])  
 sorted\_array = quick\_sort(multiplied\_array)  
 print(sorted\_array)  
 ans = 0  
 for i in range(0, len(sorted\_array), 10):  
 ans += sorted\_array[i]  
 return ans  
  
def quick\_sort(array):  
 if len(array) == 0:  
 return array  
 length = len(array)  
 separator\_start = random.randint(0, length - 1)  
 separator\_elem = array[separator\_start]  
 smaller, bigger, separator = [], [], []  
 for i in range(0, length):  
 if array[i] < separator\_elem:  
 smaller.append(array[i])  
 elif array[i] == separator\_elem:  
 separator.append(array[i])  
 else:  
 bigger.append(array[i])  
 return [\*quick\_sort(smaller), \*separator, \*quick\_sort(bigger)]  
  
  
with open("input.txt", "r") as f:  
 len\_1, len\_2 = list(map(int, f.readline().split(" ")))  
 array\_1 = list(map(int, f.readline().split(" ")))  
 array\_2 = list(map(int, f.readline().split(" ")))  
  
with open("output.txt", "w") as d:  
 d.write(str(integer\_sort(array\_1, array\_2)))

**Задача 8.**

Quick\_sort saves us again! О, да вы из Англии!

Пока писала, поняла, что нет ничего на свете лучше quick sortа. (А еще, что нет ничего хуже словарей, поэтому работаем с массивами.)

Считываем координаты, вносим их в массив, каждую пару отдельным подмассивом. Для каждой пары функция сначала просчитает расстояние до начала координат и вставит его на нулевое место в подмассиве (при этом сами координаты сохранятся, просто съедут на один индекс вправо). Далее (да, так можно, это не читерство, я проверила) мы сортируем список по определенному значению, то бишь по всем элементам, где у нас хранится расстояние = array[i][0]. Отсортировали массив, и возвращаемся в функцию поиска, где уже пробегаем по отсортированному массиву столько раз, сколько было указано в условии (необходимое количество точек). Выводим строкой скобочки с индексами, ведь x у нас всегда sorted\_dots[i][1], а y — sorted\_dots[i][2]. Если точек вывести надо более одной, выводим между ними запятую. Все!

import math  
import random  
import time  
t\_start = time.perf\_counter()  
  
def find\_K\_dots(coordinates, K):  
 for dot in coordinates:  
 distance = math.sqrt((dot[0] \*\* 2) + (dot[1] \*\* 2))  
 dot.insert(0, distance)  
 sorted\_dots = quick\_sort(coordinates)  
 ans = ''  
 for i in range(1, K + 1):  
 ans += "[" + str(sorted\_dots[i - 1][1]) + "," + str(sorted\_dots[i - 1][2]) + "]"  
 if i < K:  
 ans += ","  
 return ans  
  
  
def quick\_sort(array):  
 if len(array) == 0:  
 return array  
 length = len(array)  
 separator\_start = random.randint(0, length - 1)  
 separator\_elem = array[separator\_start][0]  
 smaller, bigger, separator = [], [], []  
 for i in range(0, length):  
 if array[i][0] < separator\_elem:  
 smaller.append(array[i])  
 elif array[i][0] == separator\_elem:  
 separator.append(array[i])  
 else:  
 bigger.append(array[i])  
 return [\*quick\_sort(smaller), \*separator, \*quick\_sort(bigger)]  
  
  
  
coordinates = []  
with open("input.txt", "r") as f:  
 num, K = list(map(int, f.readline().split(" ")))  
 for i in range(num):  
 x, y = list(map(int, f.readline().split(" ")))  
 coordinates. append([x, y])  
  
with open("output.txt", "w") as d:  
 d.write(find\_K\_dots(coordinates, K))

**Описание проведенных тестов.**

**1**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| -10^4 to 10^4 | -10^4 to 10^4 | 0.052350900000000006 секунд |
| Random array | Sorted random array | 0.055401500000000006 секунд |
| 10^4 to -10^4 | -10^4 to 10^4 | 0.052319299999999985 секунд |

**2**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 3 | 1 3 2 | 0.0013705999999999996 секунд |
| 5 | 1 4 5 3 2 | 0.0007901000000000019 секунд |
| 40 | 1 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 5 21 11 23 3 25 13 27 7 29 15 31 2 33 17 35 9 37 19 39 | 0.0011388999999999982 секунд |

**5**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 3 0 6 1 5 | 3 | 0.0008798 секунд |
| 1 3 1 | 1 | 0.0013573999999999947 секунд |
| 3 4 5 6 6 6 7 3 12 5 5 6 3 5 | 6 | 0.0007174000000000069 секунд |

**6**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 4 4  7 1 4 9  2 7 8 11 | 51 | 0.004238699999999998 секунд |
| 3 4  1 3 4  4 5 6 3 | 23 | 0.003231800000000007 секунд |
| 4 1  2 3 4 5  3 | 6 | 0.003292299999999998 секунд |

**8**

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Время выполнения** |
| 2 1  1 3  -2 2 | [-2,2] | 0.0008390999999999954 секунд |
| 3 2  3 3  5 -1  -2 4 | [3,3],[-2,4] | 0.0008625000000000022 секунд |
| 5 3  2 -8  2 -8  -8 2  5 5  12 0 | [5,5],[2,-8],[2,-8] | 0.0016401000000000054 секунд |

**Выводы по проделанной работе.**

Quick Sort украла мое сердечко. Сортировка простая, удобная и гибкая, легко интегрируется в задачи.

7-я задача – сущий ад. Сидела до 5:40 утра, так и не решила. Обсуждала с фронтендером с Яндекса – мы вместе не смогли. Теперь появилась идея сделать его с quick-sort, поэтому может, потом получится.