

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3
по курсу «Алгоритмы и структуры данных»
Тема: Быстрая сортировка, сортировки за линейное время
Вариант 8

Выполнил:
Макунина А.А.
К32421

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту	3
Задача №1. Улучшение Quick sort.....	3
Задача №6. Сортировка целых чисел.....	6
Задача №5. Индекс Хирша.....	9
Вывод:	12

Задачи по варианту

Задача №1. Улучшение Quick sort

Используя псевдокод процедуры Randomized - QuickSort, а также Partition из презентации к Лекции 3 (страницы 8 и 12), напишите программу быстрой сортировки на Python и проверьте ее, создав несколько рандомных массивов, подходящих под параметры:

- Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \leq n \leq 10^4$) — число элементов в массиве.

Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9 .

- Формат выходного файла (output.txt). Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.

- Ограничение по времени. 2 сек.

- Ограничение по памяти. 256 мб.

Основное задание. Цель задачи - переделать данную 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 (смотри в Лекции 3 слайд 17). То есть ваша новая процедура разделения должна разбить массив на три части:

- $A[k] < x$ для всех $l + 1 \leq k \leq m_1 - 1$

- $A[k] = x$ для всех $m_1 \leq k \leq m_2$

- $A[k] > x$ для всех $m_2 + 1 \leq k \leq r$

- Формат входного и выходного файла аналогичен п.1.

Это так сказать BASE. Quicksort принимает на вход массив array, индекс первого элемента и индекс последнего элемента. После сравнения разбиение выполняется вызовом функции Partition(array, start, end). Эта функция возвращает числовое значение (сохраненное в pIndex), которое определяет индекс pivot (который соответствует элементу, уже правильно расположенному в нашем массиве). Поскольку начальная последовательность может иметь вероятность наихудшего случая, вводя

случайность (random), алгоритм может получить лучшую ожидаемую производительность для всех входных данных.

```
from random import randint

def Quicksort(array, start, end): #рекурсия по
индексу
    if start < end:
        pIndex = Partition(array, start, end)
        Quicksort(array, start, pIndex - 1)
        Quicksort(array, pIndex + 1, end)

    return array

def Partition(array, start, end):
    pivot_index = randint(start, end) #случайные
значения целого типа
    temp = array[end]
    array[end] = array[pivot_index]
    array[pivot_index] = temp
    pIndex = start

    for i in range(start, end):
        if array[i] <= array[end]:
            temp = array[i]
            array[i] = array[pIndex]
            array[pIndex] = temp
            pIndex += 1
    temp1 = array[end]
    array[end] = array[pIndex]
    array[pIndex] = temp1

    return pIndex

f = open("input.txt")
n = int(f.readline())
array = list(map(int, f.readline().split()))
f.close()
w = open("output.txt", "w")
```

```
w.write(str(Quicksort(array, 0, len(array) - 1)))
w.close()
```

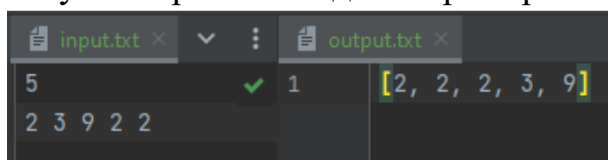
Объединив функции, выполнена оптимизация на 3 подзадачи (+elif).
Разделитель хранится под переменной partition.

```
from random import randint

def Quicksort(array):
    if len(array) == 0:
        return array
    p_start = randint(0, len(array) - 1)
    p_element = array[p_start]
    min, max, partition = [], [], []
    for i in range(0, len(array)):
        if array[i] < p_element:
            min.append(array[i])
        elif array[i] == p_element:
            partition.append(array[i])
        else:
            max.append(array[i])
    return [*Quicksort(min), *partition,
*Quicksort(max)] #каждый элемент отдельно

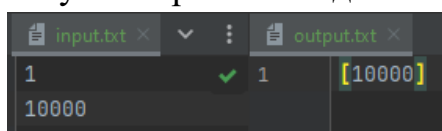
f = open("input.txt")
n = int(f.readline())
array = list(map(int, f.readline().split()))
f.close()
w = open("output.txt", "w")
w.write(str(Quicksort(array)))
w.close()
```

Результат работы кода на примере из текста задачи:



input.txt	output.txt
5	1
2 3 9 2 2	[2, 2, 2, 3, 9]

Результат работы кода на максимальных и минимальных значениях:



input.txt	output.txt
1	1
10000	[10000]

```

input.txt x  output.txt x
10000  ✓ 1  [-999836, -999566, -999503, -999286, -999027, -998790, -998777,
-64909 139675  ↵ -998755, -998215, -998194, -998156, -997785, -997598,
↵ 756959  ↵ -997407, -997354, -997273, -997253, -997216, -997125, -996682,
↵ -229278  ↵ -996486, -996426, -996377, -996192, -996076, -995897,
↵ -620913  ↵ -995495, -995490, -995406, -995369, -995209, -995140, -994940,
↵ 560971 955787 ↵ -994755, -994661, -994318, -994293, -993987, -993958,
↵ -209336  ↵ -993602, -993412, -993009, -992715, -992635, -992128, -992091,
↵ 950042  ↵ -991745, -991588, -991583, -990982, -990380, -990349,
↵ -614926  ↵ -990267, -989989, -989908, -989587, -989509, -989087, -989079,
↵ 264674 20847  ↵ -988656, -987894, -987760, -987487, -987463, -987394,
↵ 58842 929507  ↵ -987275, -986936, -986275, -985908, -985903, -985815, -985739,
↵ -659254  ↵ -985315, -985196, -985106, -983876, -983063, -982645,
↵ -511019  ↵ -982401, -982362, -982329, -982313, -981923, -981204, -981117,
↵ -181235  ↵ -981089, -981040, -980987, -980814, -980494, -980118,
↵ -555066  ↵ -979895, -979810, -979584, -979334, -979212, -978825, -978804,
↵ 742277  ↵ -978767, -978551, -978514, -978433, -978360, -978275,

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.007403600000000001	0.13176631927490234
Пример из задачи	0.0066120999999999996	0.1317739486694336
Верхняя граница диапазона значений входных данных из текста задачи	0.1313517	1.1662673950195312

Вывод по задаче: я рассмотрела алгоритм (кстати без лекции), который мне очень понравился. Надеюсь, это было полезно.

Задача №6. Сортировка целых чисел

В этой задаче нужно будет отсортировать много неотрицательных целых чисел.

Вам даны два массива, A и B , содержащие соответственно n и m элементов. Числа, которые нужно будет отсортировать, имеют вид $A_i \cdot B_j$, где $1 \leq i \leq n$ и $1 \leq j \leq m$. Иными словами, каждый элемент первого массива нужно умножить на каждый элемент второго массива.

Пусть из этих чисел получится отсортированная последовательность C длиной $n \cdot m$. Выведите сумму каждого десятого элемента этой последовательности (то есть, $C_1 + C_{11} + C_{21} + \dots$).

- **Формат входного файла (input.txt).** В первой строке содержатся числа n и m ($1 \leq n, m \leq 6000$) – размеры массивов. Во второй строке содержится

n чисел — элементы массива A . Аналогично, в третьей строке содержится m чисел — элементы массива B . Элементы массива неотрицательны и не превосходят 40000.

- **Формат выходного файла (output.txt).** Выведите одно число — сумму каждого десятого элемента последовательности, полученной сортировкой попарных произведений элементов массивов A и B .
- **Ограничение по времени.** 2 сек.

Сначала в цикле перемножаются элементы массива A и B , далее полученный массив сортируется в функцию Quicksort из прошлой задачи. Отсортированный массив входит в цикл for с шагом 10, где суммируются элементы соответственно.

```
from random import randint

def Multiplication(A, B):
    multi_AB = []
    for i in range(len(A)):
        for j in range(len(B)):
            multi_AB.append(A[i] * B[j])
    sorted_array = Quicksort(multi_AB)
    sum = 0
    for i in range(0, len(sorted_array), 10):
        sum += sorted_array[i]
    return sum

def Quicksort(array):
    if len(array) == 0:
        return array
    p_start = randint(0, len(array) - 1)
    p_element = array[p_start]
    min, max, partition = [], [], []
    for i in range(0, len(array)):
        if array[i] < p_element:
            min.append(array[i])
        elif array[i] == p_element:
            partition.append(array[i])
        else:
            max.append(array[i])
    return [*Quicksort(min), *partition,
```

```
*Quicksort(max)] # каждый элемент отдельно

f = open("input.txt")
n, m = list(map(int, f.readline().split()))
A, B = list(map(int, f.readline().split())),
list(map(int, f.readline().split()))
f.close()
w = open("output.txt", "w")
w.write(str(Multiplication(A, B)))
w.close()
```

Результат работы кода на примере из текста задачи:

input.txt	output.txt
4 4	1 51
7 1 4 9	
2 7 8 11	

Результат работы кода на максимальных и минимальных значениях:

input.txt	output.txt
1 1	1 24
6	
4	

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.008018999999999998	0.1309070587158203
Пример из задачи	0.009003399999999995	0.13100433349609375
Верхняя граница диапазона значений входных данных из текста задачи	Короче я не знаю, потому что котёнок умер как говорится	аналогично

Вывод по задаче: задача простая, кайф, но я навела шуму в пайчарме, когда дала ему перемножить матрицу 40000*40000...

Задача №5. Индекс Хирша

Для заданного массива целых чисел `citations`, где каждое из этих чисел – число цитирований i -ой статьи ученого-исследователя, посчитайте индекс Хирша этого ученого. По определению Индекса Хирша на Википедии: Учёный имеет индекс h , если h из его/её N_p статей цитируются как минимум h раз каждая, в то время как оставшиеся $(N_p - h)$ статей цитируются не более чем h раз каждая. Иными словами, учёный с индексом h опубликовал как минимум h статей, на каждую из которых сослались как минимум h раз. Если существует несколько возможных значений h , в качестве h -индекса принимается максимальное из них.

- Формат ввода или входного файла (`input.txt`). Одна строка `citations`, содержащая n целых чисел, по количеству статей ученого (длина `citations`), разделенных пробелом или запятой.
- Формат выхода или выходного файла (`output.txt`). Одно число - индекс Хирша (h -индекс).
- Ограничения: $1 \leq n \leq 5000$, $0 \leq citations[i] \leq 1000$.

На вход приходит массив с количеством цитирований, он сортируется по убыванию. Далее используем функцию `enumerate()` в цикле `for/in` в качестве счетчика элементов последовательности. В конце сравниваем значение элемента и его индекс. Если `idx` (количество статей) $>$ соответствующего количества цитирований, то это первая статья, которая лишняя (то есть если мы её уберем из расчета индекса, то он будет $\geq idx$). `idx` начинается с 1, потому что количество не считается с 0.

```
def Get_h_index(citations):
    citations = sorted(citations, reverse=True) #
    #возвращает новый отсортированный список по убыванию
    for idx, item in enumerate(citations, 1):
        #счётчик количества элементов в последовательности
        if item < idx:
            break
    return idx - 1

f = open("input.txt")
citations = list(map(int, f.readline().split(',')))
f.close()
```

```
w = open("output.txt", "w")
w.write(str(Get_h_index(citations)))
w.close()
```

Результат работы кода на примерах из текста задачи:

input.txt	3,0,6,1,5	✓	1	3
input.txt	1	✓	1	1

Результат работы кода на максимальных и минимальных значениях:

input.txt	1	✓	1	0
input.txt	1	✓	1	833

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0010407999999999945	0.017284393310546875
Пример из задачи	0.0009221999999999998	0.017347335815429688
Пример из задачи	0.0009650999999999965	0.017339706420898438

Верхняя граница диапазона значений входных данных из текста задачи	0.007346700000000005	0.4336357116699219
---	----------------------	--------------------

Вывод по задаче: задача казалось сложной, но оказалась лёгкой. Использован самый примитивный алгоритм, как мне кажется. Столкнулась только с потерей логики при сравнении индекса и количества цитирований, что необходимо учесть первую «лишнюю» статью.

Вывод:

Quick Sort – хороший алгоритм, наверное, один из самых приятных и простых. Далее была небольшая головоломка, с которой я справилась, ура!