

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 7

Выполнила:  
Пожидаева Е.Р.

Санкт-Петербург  
2024

## Содержание отчета

Задание №1. Улучшение Quick sort.....	3
Задание №5. Индекс Хирша .....	6
Задание №8. К ближайших точек к началу координат.....	10
Дополнительные задачи .....	14
Задание №2 .....	14
Задание №3 .....	17
Задание №6.....	22

# Задание №1. Улучшение Quick sort

1. 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 случайных 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.
- Для проверки можно выбрать наихудший случай, когда сортируется массив размера  $10^3, 10^4, 10^5$  чисел порядка  $10^9$ , отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний - случайный. Сравните на данных наборах Randomized-QuickSort и простой QuickSort. (А также есть Median-QuickSort, см. задание 10.2; и Tail-Recursive-QuickSort, см. [Кормен. 2013, стр. 217](#))

**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  $\ell + 1 \leq k \leq m_1 - 1$
- $A[k] = x$  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.

• Аналогично п.1 этого задания сравните Randomized-QuickSort + Partition и ее с Partition3 на наборах случайных данных, в которых содержатся всего несколько уникальных элементов при  $n = 10^3, 10^4, 10^5$ . Что быстрее, Randomized-QuickSort + Partition3 или Merge-Sort?

• Пример:

input.txt	output.txt
5	2 2 2 3 9
2 3 9 2 2	

Листинг кода:

```
f = open('input.txt')
n = int(f.readline())
arr = [int(x) for x in f.readline().split()]

def quicksort(arr):
    if len(arr) <= 1:
        return arr
    else:
        q = arr[randint(0, len(arr) - 1)]
        l_arr = [n for n in arr if n < q]
        e_arr = [q] * arr.count(q)
        b_arr = [n for n in arr if n > q]
        return quicksort(l_arr) + e_arr + quicksort(b_arr)

f = open('output.txt', 'w')
f.write(' '.join(map(str, quicksort(arr))))
```

### Текстовое объяснение кода:

Из файла input.txt считываются две строки, n - кол-во чисел в списке, A - сам список. Переменной l (первый индекс списка) присваивается значение нуля.

Запускается функция QuickSort с параметрами (A - список, l - первый индекс списка, n - кол-во чисел в списке). Условие - если кол-во чисел в списке меньше или равно единице, список возвращается без изменений. В противном случае - переменной q присваивается значение среднего индекса списка. Переменной i присваивается значение l, переменной j - (n-1) - последний индекс списка. Запускается цикл: если i-тое число списка меньше значения среднего числа списка, то к i прибавляется один.

Запускается следующий цикл: если i-тое число списка больше значения среднего числа списка, то из j вычитается один.

Если выполняется условие i меньше или равно j, то элементы списка под номерами i и j меняются местами. К i прибавляется 1, из j вычитается 1.

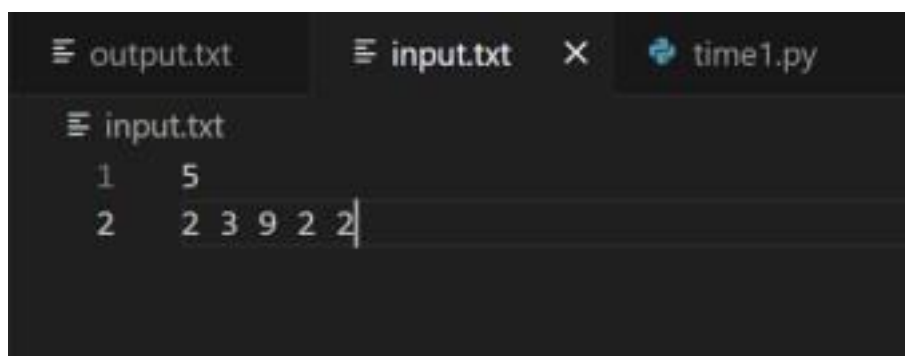
Далее сортируем два подсписка.

Отсортированный список записывается в output.txt.

### Вывод:

я научилась писать сортировку quick sort.

Примеры работы кода:



```
output.txt input.txt x time1.py
input.txt
1 5
2 2 3 9 2 2
```



```
output.txt x input.txt time1.py
output.txt
1 2 2 3 2 9
```

	Время выполнения/сек Затраты памяти/мб
--	--

Нижняя граница диапазона значений входных данных из текста задачи	0.0010094642639160156 14.765625
Пример из задачи	0.00801229476928711 14.78515625

Верхняя граница

0.8997068405151367 14.94921875

диапазона значений  
входных данных из  
текста задачи

## Задание №5. Индекс Хирша

### 5 задача. 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:  $1 \leq n \leq 5000$ ,  $0 \leq citations[i] \leq 1000$ .
- Пример.

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

Пояснение. `citations = [3,0,6,1,5]` означает, что ученый опубликовал 5 статей в целом, и каждая из них оказалась процитирована 3, 0, 6, 1, 5 раз соответственно. Поскольку у ученого есть 3 статьи с минимум тремя цитированиями, а у оставшихся двух - не более 3 цитирований, его индекс Хирша равен 3.

- Пример.

input.txt	output.txt
1,3,1	1

Листинг кода:

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

def sorted(citiz):
    for i in range(1, len(citiz)):
        for j in range(i, 0, -1):
            if citiz[j] > citiz[j - 1]:
                citiz[j], citiz[j - 1] = citiz[j - 1], citiz[j]
    return citiz

def hIndex(citiz):
    sorted(citiz)
    n = len(citiz) - 1
    i = 0
    while i < n:
        if citiz[n - i] < i:
            break
        else:
            i += 1
    res = i - 1
    return res

y = hIndex(citiz)
f = open('output.txt', 'w')
f.write(str(y))
```

Текстовое объяснение кода:

Из input.txt считывается список.

Далее список сортируется в убывающем порядке с помощью сортировки вставкой.

Далее определяем функцию hIndex с параметром citiz(список из input). Переменной n присваивается значение (длина списка минус 1). Переменной i присваивается ноль. Запускается цикл с условием - пока i меньше n:

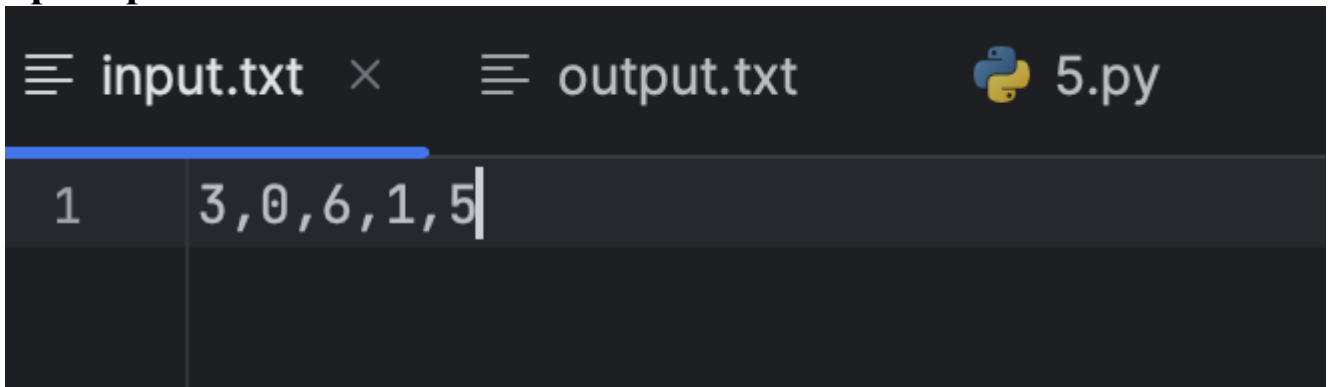
Далее при условии, что элемент под номером (n-i) меньше i, цикл

прерывается. Если это условие не выполняется, то к  $i$  прибавляется единица. Переменной `res` присваивается значение  $i$  минус 1. В файл `output.txt` записывается результат сортировки.

Вывод:

я узнала, что такое индекс Хирша.

## Примеры




The image shows a code editor interface with three tabs: `input.txt`, `output.txt`, and `5.py`. The `input.txt` tab is active and contains the text `3,0,6,1,5` on the first line. The `output.txt` and `5.py` tabs are inactive. The editor has a dark theme and a blue cursor is visible at the end of the text in the `input.txt` tab.

File	Content
input.txt	3,0,6,1,5
output.txt	
5.py	



≡ input.txt

≡ output.txt ×

 5.py

1

3

	Время выполнения/сек	Затраты памяти/мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0003221035003662109 4	14.51953125
Пример из задачи	0.0016486644744873047	14.7734375

Верхняя граница диапазона значений входных данных из текста задачи

0.8404355049133301  
15.3046875

## Задание №8. К ближайших точек к началу координат

В этой задаче, ваша цель - найти  $K$  ближайших точек к началу координат среди данных  $n$  точек.

- **Цель.** Заданы  $n$  точек на поверхности, найти  $K$  точек, которые находятся ближе к началу координат  $(0, 0)$ , т.е. имеют наименьшее расстояние до начала координат. Напомним, что расстояние между двумя точками  $(x_1, y_1)$  и  $(x_2, y_2)$  равно  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ .
- **Формат ввода или входного файла (input.txt).** Первая строка содержит  $n$  - общее количество точек на плоскости и через пробел  $K$  - количество ближайших точек к началу координат, которые надо найти. Каждая следующая из  $n$  строк содержит 2 целых числа  $x_i, y_i$ , определяющие точку  $(x_i, y_i)$ . Ограничения:  $1 \leq n \leq 10^5$ ;  $-10^9 \leq x_i, y_i \leq 10^9$  - целые числа.
- **Формат выхода или выходного файла (output.txt).** Выведите  $K$  ближайших точек к началу координат в строку в квадратных скобках через запятую. Ответ вывести в порядке возрастания расстояния до начала координат. Если оно равно, порядок произвольный.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 256 мб.

## Листинг кода

```
f = open('input.txt')
n, k = map(int, f.readline().split())
points = [list(map(int, line.split())) for line in f]

def distance(point):
    x, y = point
    return x ** 2 + y ** 2

def quick_sort(a):
    if len(a) <= 1:
        return a
    elem = a[len(a) // 2][1]
    left = [x for x in a if x[1] < elem]
    middle = [x for x in a if x[1] == elem]
    right = [x for x in a if x[1] > elem]
    return quick_sort(left) + middle + quick_sort(right)

def close(points, k):
    sorted_points = quick_sort(points)
    return sorted_points[:k]

points_closest_to_k = close(points, k)
```

```
f = open('output.txt', 'w')
for i, point in enumerate(points_closest_to_k):
    f.write(f"[{point[0]}, {point[1]}]")
```

Текстовое объяснение кода

Открываем текстовый файл input.txt и читаем из него данные: n - кол-во точек на плоскости, k - точки, которые необходимо найти. Список points - координаты n числа точек.

Определяем функцию distance, где с помощью формулы мы рассчитываем расстояние между точкой и началом координат.

Далее определяем функцию быстрой сортировки, где мы сортируем координаты точек. Сортируем массив так, чтобы в нём были подмассивы с координатами точек.

Определяем функцию close с параметрами points и k. В переменную sorted\_points определяем отсортированный список координат точек. Далее мы возвращаем не весь список, а только заданное k число элементов списка.

Создаю переменную points\_closest\_to\_k и в неё записывается результат выполнения функции close с параметрами points и k.

Открываем файл output.txt для записи. Запускаем цикл for, чтобы вывести координаты x и y расстояния.

Наконец, записываем результат в файл output.txt.

Пример:

input.txt		output.txt		8.py
1	2 1			
2	1 3			
3	-2 2			

```
≡ input.txt    ≡ output.txt  ×  8.py
1  [-2, 2]
```

	Время выполнения/сек	Затраты памяти/мб
Нижняя граница диапазона значений входных данных из текста задачи	0.0009982585906982422	14.77734375
Пример из задачи	0.0005059242248535156	14.8359375
Пример из задачи	0.000514984130859375	14.56640625

## Дополнительные задачи

### Задание №2. Анти-quick sort

Для сортировки последовательности чисел широко используется быстрая сортировка - QuickSort. Далее приведена программа на языке Pascal Python, которая сортирует массив a, используя этот алгоритм.

```
def qsort (left, right):
    key = a [(left + right) // 2]
    i = left
    j = right
    while i <= j:
        while a[i] < key: # first while
            i += 1
        while a[j] > key : # second while
            j -= 1
        if i <= j :
            a[i], a[j] = a[j], a[i]
            i += 1
            j -= 1
    if left < j:
        qsort(left, j)
    if i < right:
        qsort(i, right)

qsort(0, n - 1)
```

Хотя QuickSort является очень быстрой сортировкой в среднем, существуют тесты, на которых она работает очень долго. Оценивать время работы алгоритма будем числом сравнений с элементами массива (то есть, суммарным числом сравнений в первом и втором while). Требуется написать программу, генерирующую тест, на котором быстрая сортировка сделает наибольшее число таких сравнений.

### Листинг кода

```
f = open('input.txt', 'r')
n = int(f.readline())

def quicksort(arr): # сортируем в порядке возрастания
    if len(arr) <= 1:
        return arr
```

```

pivot = arr[len(arr) // 2]
left = [x for x in arr if x < pivot]
middle = [x for x in arr if x == pivot]
right = [x for x in arr if x > pivot]
return quicksort(left) + middle + quicksort(right)

def generate_permutation(n): # генерируем список чисел от
1 до n (включительно) в порядке возрастания
    return list(range(1, n + 1))

def find_max_comparisons_permutation(
    n): # находим перестановку чисел от 1 до n,
которая приводит к максимальному количеству сравнений при
# применении алгоритма быстрой сортировки
    permutations = []
    for i in range(1, n + 1):
        permutation = generate_permutation(n)
        permutation.remove(i)
        permutation.insert(0, i)
        permutations.append(permutation)
    max_comparisons = 0
    max_comparisons_permutation = []
    for permutation in permutations:
        comparisons = len(quicksort(permutation)) - 1
        if comparisons > max_comparisons:
            max_comparisons = comparisons
            max_comparisons_permutation = permutation
    return max_comparisons_permutation

max_comparisons_permutation =
find_max_comparisons_permutation(n) # поиск перестановки
с максимальным числом сравнений

f = open('output.txt', 'w')
f.write(' '.join(map(str, max_comparisons_permutation)))

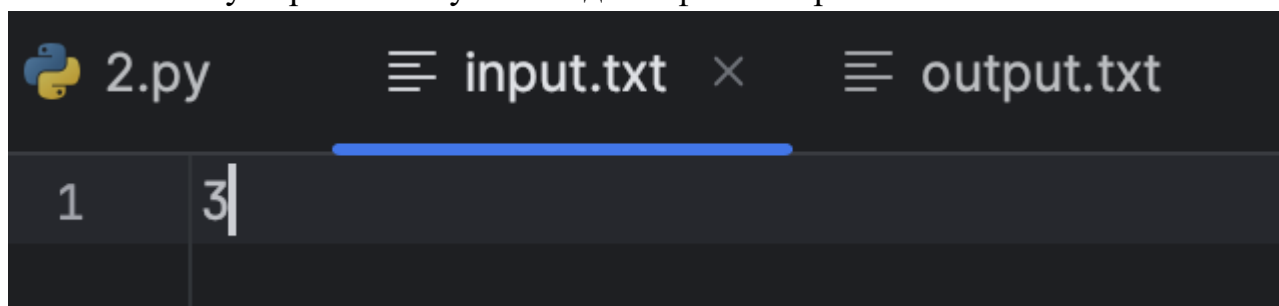
```

Текстовое объяснение кода:

Данный код открывает файл 'input.txt' и считывает из него число  $n$ . Затем определяет функцию quicksort для сортировки массива в порядке возрастания с использованием алгоритма быстрой сортировки. Также задаются функции generate\_permutation для генерации перестановки чисел от 1 до  $n$  и find\_max\_comparisons\_permutation для поиска перестановки, при которой алгоритм быстрой сортировки совершает максимальное число сравнений.

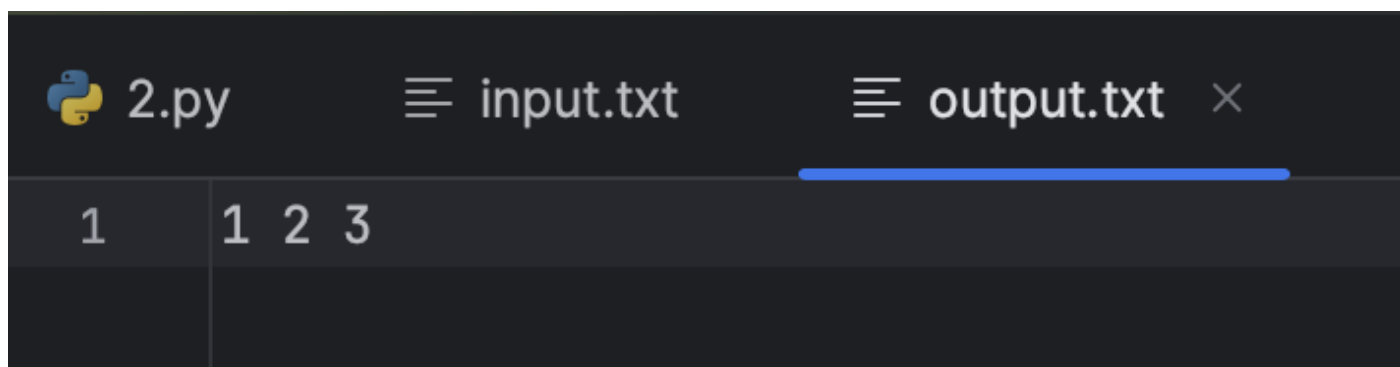
В основной части кода вызывается функция find\_max\_comparisons\_permutation с параметром  $n$ , и результат записывается в переменную max\_comparisons\_permutation. Затем открывается файл 'output.txt' для записи, и найденная перестановка записывается в этот файл в виде строк, разделенных пробелами.

Кратко, код выполняет чтение входных данных из файла, находит перестановку, при которой быстрая сортировка совершает максимальное количество сравнений, и записывает эту перестановку в выходной файл 'output.txt'.



The screenshot shows a code editor with a dark theme. At the top, there are tabs for '2.py', 'input.txt', and 'output.txt'. The 'input.txt' tab is active and highlighted with a blue underline. Below the tabs, the content of 'input.txt' is displayed in a table-like structure. The first row has a column index '1' and a value '3'. The rest of the table is empty.

1	3



The screenshot shows the same code editor with the 'output.txt' tab active and highlighted with a blue underline. Below the tabs, the content of 'output.txt' is displayed in a table-like structure. The first row has a column index '1' and a value '1 2 3'. The rest of the table is empty.

1	1 2 3



### Задание №3. Сортировка пугалом

«Сортировка пугалом» — это давно забытая народная потешка. Участнику под верхнюю одежду продевают деревянную палку, так что у него оказываются растопырены руки, как у огородного пугала. Перед ним ставятся  $n$  матрёшек в ряд. Из-за палки единственное, что он может сделать — это взять в руки две матрёшки на расстоянии  $k$  друг от друга (то есть  $i$ -ую и  $i + k$ -ую), развернуться и поставить их обратно в ряд, таким образом поменяв их местами.

Задача участника — расположить матрёшки по неубыванию размера. Может ли он это сделать?

- **Формат входного файла (input.txt).** В первой строчке содержатся числа  $n$  и  $k$  ( $1 \leq n, k \leq 10^5$ ) — число матрёшек и размах рук. Во второй строчке содержится  $n$  целых чисел, которые по модулю не превосходят  $10^9$  — размеры матрёшек.
- **Формат выходного файла (output.txt).** Выведите «ДА», если возможно отсортировать матрёшки по неубыванию размера, и «НЕТ» в противном случае.

```
def is_sorted(arr):
    for i in range(1, len(arr)):
        if int(arr[i]) < int(arr[i - 1]):
            return False
    return True

def scarecrow_sort(arr, k):
    for i in range(len(arr) - k):
        if int(arr[i]) > int(arr[i + k]):
            arr[i], arr[i + k] = arr[i + k], arr[i]

    return 'YES' if is_sorted(arr) else 'NO'

f = open('input.txt')
a = f.readline().split()
for i in range(len(a)):
    n = int(a[0])
    k = int(a[1])
arr = f.readline().split()
f.close()
```

```
f1 = open('output.txt', 'w')
f1.write(scarecrow_sort(arr,k))
f1.close()
```

### Текстовое объяснение:

Данный код открывает файл 'input.txt' и считывает из него первую строку, которая представляет собой два числа, разделенных пробелом - количество элементов в массиве и значение переменной k. Затем он считывает следующую строку, которая содержит сам массив.

Далее выполняется функция `scarecrow_sort(arr, k)`, которая сортирует массив `arr` таким образом, чтобы каждый  $i$ -й элемент был меньше или равен  $(i + k)$ -му элементу. Если сортировка прошла успешно, функция `is_sorted(arr)` возвращает `True`, иначе - `False`.

Затем функция `scarecrow_sort(arr, k)` возвращает строковое значение 'YES', если массив `arr` является отсортированным, и 'NO' в противном случае.

Затем данный код записывает результат работы функции `scarecrow_sort(arr, k)` в файл 'output.txt'.

### Пример:

input.txt		output.txt		3.py	5.py
1	3 2				
2	2 1 3				

input.txt		output.txt		3.py	5.py
1	НЕТ				

## Задание №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.

```
f=open('input.txt')
n, k = f.readline().split()
ls1 = [int(x) for x in f.readline().split()]
ls2 = [int(x) for x in f.readline().split()]
```

```

n = int(n)
k = int(k)

def insertion_sort(new):
    for i in range(1, len(new)): # проходим по массиву,
        # начиная со второго элемента
        item = new[i] # элемент, для которого хотим найти
        # правильную позицию в массиве
        j = i - 1 # элемент, который изначально был впереди
        # элемента
        while j >= 0 and new[j] > item: # если элемент
        # массива соседнего значения, сравниваем его с оставшимися
        new[j + 1] = new[j] # сдвигаем значение на одну
        # позицию влево так, чтобы j указывал на следующий элемент
        j -= 1
        new[j + 1] = item # когда мы закончили сравнение со
        # всеми элементами, помещаем элемент на правильную позицию
    s=0
    for x in range(0, len(new)):
        if x % 10 == 0:
            s += new[x]

    return s

def multiply(ls1, ls2):
    new = []
    for i in ls2:
        for j in ls1:
            a=i*j
            new.append(a)
    return insertion_sort(new)

res = str(multiply(ls1, ls2))
f = open('output.txt', 'w')
f.write(" ".join(map(str, res)))

```

Данный код открывает файл "input.txt", считывает первую строку и разделяет ее на два числа n и k. Затем считывает следующие две строки, преобразует каждую строку в

список целых чисел `ls1` и `ls2` соответственно. После этого приводит значения `n` и `k` к целочисленному типу.

Далее определена функция `"insertion_sort"`, которая реализует сортировку вставками. Эта функция принимает на вход список `new` и применяет алгоритм сортировки вставками, чтобы отсортировать элементы в порядке возрастания. Затем функция проходит по отсортированному списку `new` и суммирует каждый 10-й элемент, начиная с 0-го индекса. В итоге функция возвращает полученную сумму.

Функция `"multiply"` принимает на вход два списка `ls1` и `ls2`. Затем она создает новый пустой список `new` и проходит по каждому элементу списка `ls2`. Для каждого элемента `ls2` функция проходит по каждому элементу списка `ls1` и перемножает их. Результат умножения добавляется в список `new`. После завершения всех операций умножения и добавления элементов в список `new`, функция передает этот список на сортировку, вызывая функцию `insertion_sort`.

Затем в основной части кода вызывается функция `multiply`, передавая в нее список `ls1` и `ls2`. Результат перемножения и сортировки сохраняется в переменную `res`. Затем открывается файл `"output.txt"` и в него записывается строковое представление `res` с помощью метода `write`.

Таким образом, основная цель кода - умножить элементы списков `ls1` и `ls2`, отсортировать полученные значения и записать результат в файл `"output.txt"`.

Пример:

input.txt		output.txt	
1	4 4		
2	7 1 4 9		
3	2 7 8 11		

input.txt		output.txt	
1	5 1		