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:
Заботкина М.А.
К3139

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №1. Улучшение Quick sort

Используя псевдокод процедуры Randomized - QuickSort, а так же 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. Цель задачи - переделать данную реализацию рандомизированного алгоритма быстрой сортировки, чтобы она работала быстро даже с последовательностями, содержащими много одинаковых элементов

• Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \le n \le 10^4$) — 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.

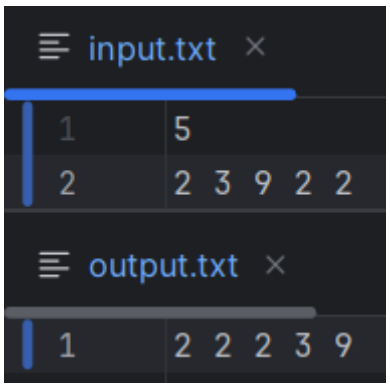
```python
import random

def partition(a, l, r):
    """Разбиение для обычной сортировки"""
    x = a[l]
    j = l
    for i in range(l + 1, r + 1):
        if a[i] <= x:
            j = j + 1
            a[j], a[i] = a[i], a[j]
    a[l], a[j] = a[j], a[l]
    return j

def randomized_quick_sort(a, l, r):
    """Обычная быстрая сортировка"""
    if l < r:
        k = random.randint(l, r)
        a[l], a[k] = a[k], a[l]
        m = partition(a, l, r)
        randomized_quick_sort(a, l, m - 1)
        randomized_quick_sort(a, m + 1, r)



def randomized_quick_sort_best(a, l, r):
    """Улучшенная версия быстрой сортировки"""
    if l < r:
        k = random.randint(l, r)
        a[l], a[k] = a[k], a[l]
        m1, m2 = partition_best(a, l, r)
```

```
        randomized_quick_sort_best(a, l, m1 - 1)
        randomized_quick_sort_best(a, m2 + 1, r)

def partition_best(a, l, r):
    """Трёхстороннее разделение"""
    x = a[l]
    m1 = l
    m2 = l
    for i in range(l + 1, r + 1):
        if a[i] < x:
            m1 += 1
            a[m1], a[i] = a[i], a[m1]
            if m1 != m2:
                m2 += 1
                a[m2], a[m1] = a[m1], a[m2]
        elif a[i] == x:
            m2 += 1
            a[m2], a[i] = a[i], a[m2]
    a[l], a[m1] = a[m1], a[l]
    return m1, m2
```

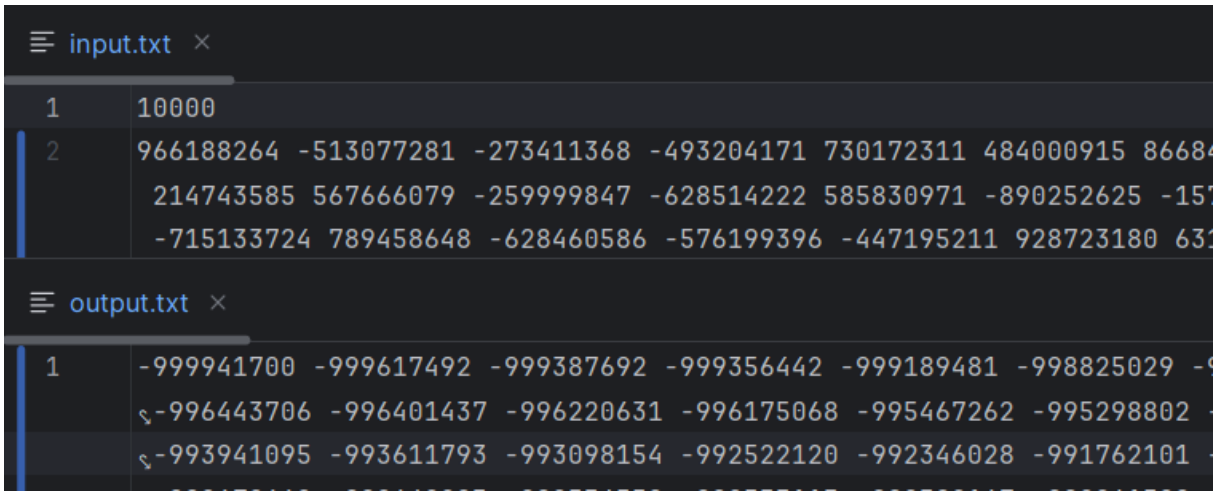
Программа реализует два алгоритма быстрой сортировки: стандартную и улучшенную. Обычная быстрая сортировка (randomized_quick_sort) делит массив на две части с помощью функции partition, где элементы меньше опорного значения перемещаются влево, а больше — вправо, после чего рекурсивно сортируются обе части.

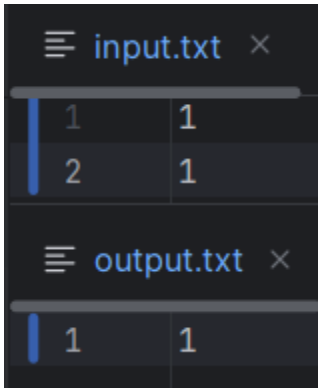
Улучшенная версия (randomized_quick_sort_best) использует трёхстороннее разбиение через partition_best, где массив делится на три части: элементы меньше опорного значения, равные ему и больше его, что ускоряет сортировку массивов с большим числом одинаковых элементов. В обоих алгоритмах для случайного выбора опорного элемента используется random.randint, что уменьшает вероятность худшего времени работы.

Результат работы кода на примерах из текста задачи:



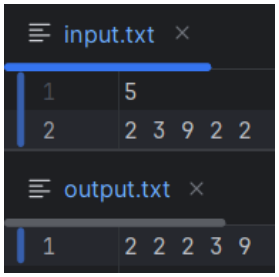
Результат работы кода на максимальных и минимальных значениях:



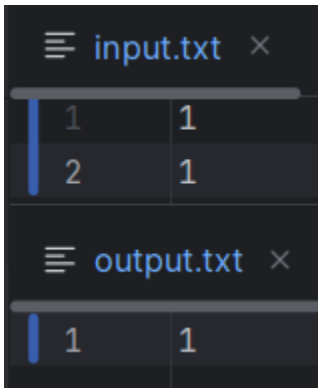


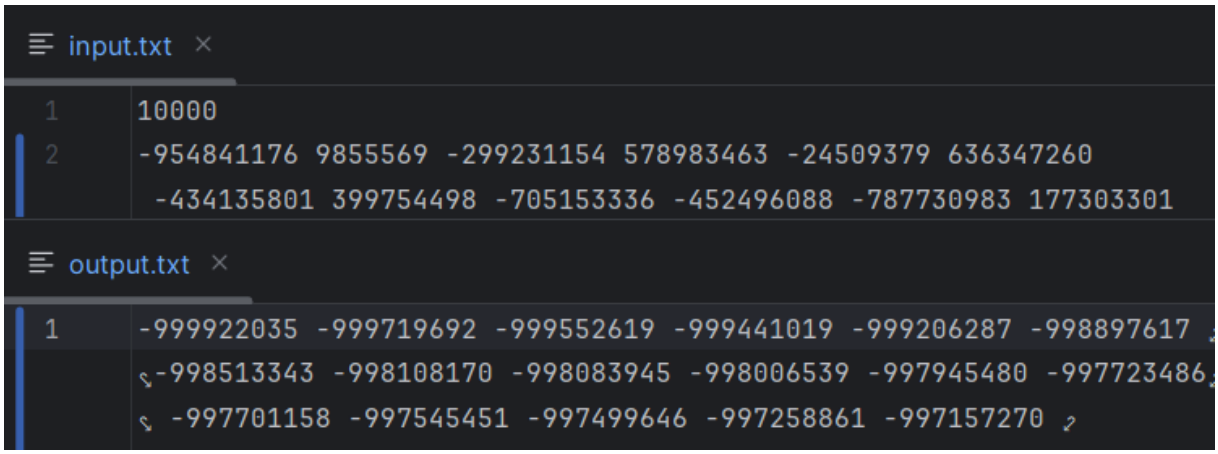
	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0009648799896240234	0.0
Пример из задачи	0.0009424686431884766	7.62939453125e-05
Верхняя граница диапазона значений входных данных из текста задачи	0.21392011642456055	0.0025787353515625

Результат работы улучшенного кода на примерах из текста задачи:



Результат работы улучшенного кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0009226799011230469	0.0
Пример из задачи	0.0009264945983886719	7.62939453125e-05
Верхняя граница диапазона значений входных данных из текста задачи	0.2709965705871582	0.0034332275390625

Вывод по задаче: реализован улучшенный алгоритм быстрой сортировки с трёхсторонним разбиением, который эффективно обрабатывает массивы с большим количеством одинаковых элементов. Улучшение показало снижение времени работы по сравнению с классическим подходом.

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 Np статей цитируются как минимум h раз каждая, в то время как оставшиеся (Np − h) статей цитируются не более чем h раз каждая. Иными словами, 5 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).

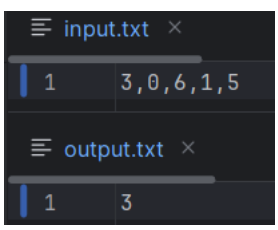
• Ограничения: 1 ≤ n ≤ 5000, 0 ≤ citations[i] ≤ 1000.

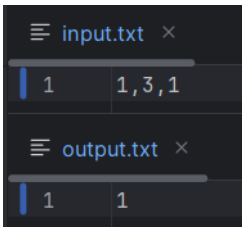
•Ограничений по времени (и памяти) не предусмотрено, проверьте максимальный случай при заданных ограничениях на данные, и оцените асимптотическое время.

```
def h_index(citations):
    """Вычисление индекса Хирша"""
    citations.sort(reverse=True)
    for i in range(len(citations)):
        if citations[i] < i + 1:
            return i
    return len(citations)
```

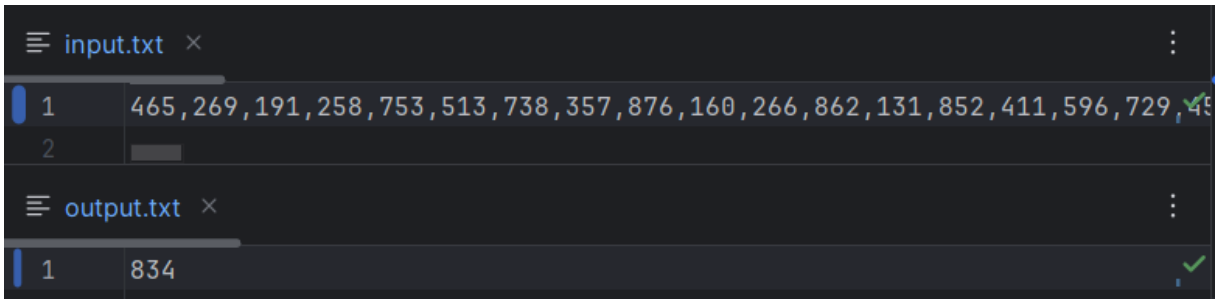
Функция h_index вычисляет индекс Хирша, сортируя массив citations (число цитирований статей) по убыванию. Затем, проходя по массиву, она проверяет, сколько статей имеют не меньше i + 1 цитирований. Как только встречается статья с количеством цитирований меньше, чем её порядок в массиве (i + 1), текущий индекс i возвращается как индекс Хирша. Если условие выполняется для всех статей, возвращается общее количество статей.

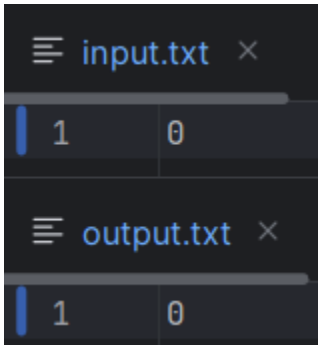
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:





	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0010063648223876953	7.62939453125e-05
Пример из задачи	0.0009386539459228516	7.62939453125e-05
Пример из задачи	0.0009999275207519531	7.62939453125e-05
Верхняя граница диапазона значений входных данных из текста задачи	0.0010552406311035156	0.01891326904296875

Вывод по задаче: выполнена реализация вычисления индекса Хирша, отражающего научную продуктивность. Использование сортировки и линейного прохода по массиву позволяет быстро определить значение индекса.

Задача №9. Ближайшие точки

В этой задаче, ваша цель - найти пару ближайших точек среди данных n точек (между собой). Это базовая задача вычислительной геометрии, которая находит применение в компьютерном зрении, систем управления трафиком.

• *Цель. Заданы n точек на поверхности, найти наименьшее расстояние между двумя (разными) точками. Напомним, что расстояние между двумя точками (x1, y1) и (x2, y2) равно p (x1 − x2) 2 + (y1 − y2) 2.*

• *Формат ввода или входного файла (input.txt). Первая строка содержит n - количество точек. Каждая следующая из n строк содержит 2 целых числа x_i , y_i , определяющие точку (x_i , y_i). Ограничения: $1 \le n \le 10^5$; $-10^9 \le x_i , y_i \le 10^9$ - целые числа.*

• *Формат выхода или выходного файла (output.txt). Выведите минимальное расстояние. Абсолютная погрешность между вашим ответом и оптимальным решением должна быть не более 10^{-3} . Чтобы это обеспечить, выведите ответ с 4 знаками после запятой. 9*

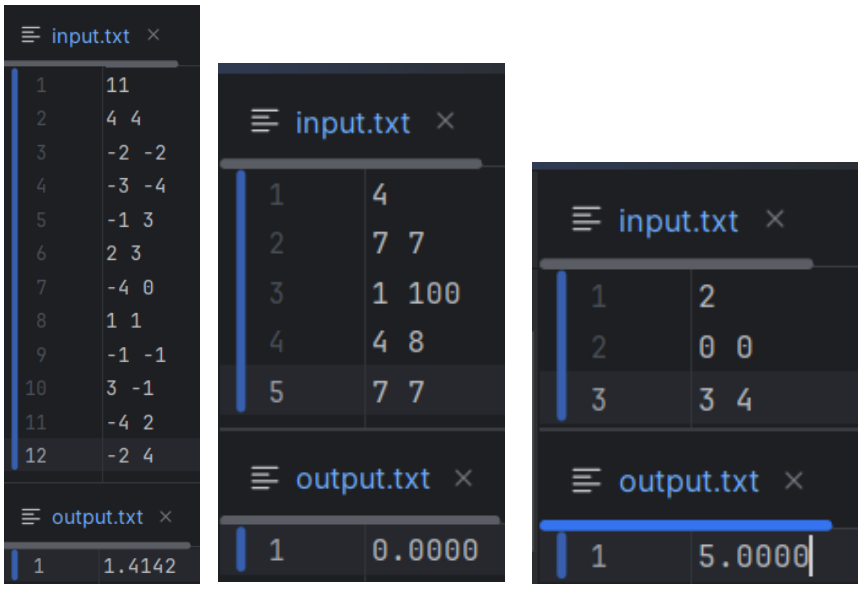
• *Ограничение по времени. 10 сек.*

• *Ограничение по памяти. 256 мб*

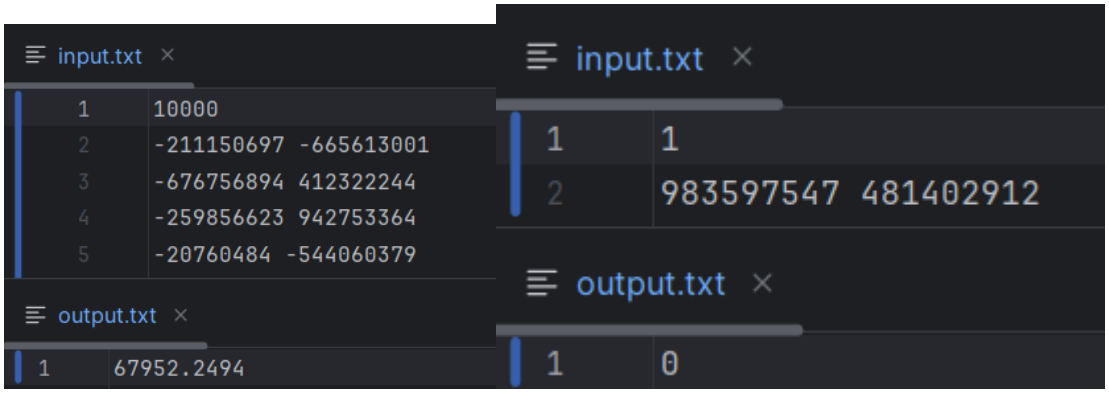
```
import math

def distance(point1, point2):
    return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1])
** 2)


def closest_pair(points_x, points_y):
    n = len(points_x)
    if n == 1:
        return 0
    elif n <= 3:
        return min(distance(points_x[i], points_x[j])
                   for i in range(n) for j in range(i + 1, n))

    mid = n // 2
    mid_x = points_x[mid][0]
    left_points = points_x[:mid]
    right_points = points_x[mid:]

    left_sorted_by_y = [p for p in points_y if p[0] <= mid_x]
    right_sorted_by_y = [p for p in points_y if p[0] > mid_x]

    min_dist = min(closest_pair(left_points, left_sorted_by_y),
                   closest_pair(right_points, right_sorted_by_y))

    strip = [p for p in points_y if abs(p[0] - mid_x) < min_dist]

    for i in range(len(strip)):
        for j in range(i + 1, min(i + 7, len(strip))):
            min_dist = min(min_dist, distance(strip[i], strip[j]))

    return min_dist


def find_closest_pair(points):
    points_x = sorted(points, key=lambda x: x[0])
    points_y = sorted(points, key=lambda y: y[1])
    return closest_pair(points_x, points_y)
```

Функция find_closest_pair находит минимальное расстояние между двумя точками из заданного множества, используя метод "Разделяй и властвуй". Сначала точки сортируются по координатам x и y, чтобы ускорить поиск, а затем вызывается рекурсивная функция closest_pair. Эта функция делит множество точек на две равные части по координате x и рекурсивно вычисляет минимальное расстояние в левой и правой частях. После этого минимальное расстояние сравнивается с расстоянием между точками, находящимися в "разделительной полосе" вокруг центральной линии. Для ускорения работы в полосе рассматриваются только точки, находящиеся ближе, чем текущее минимальное расстояние, и проверяются только максимум 7 ближайших соседей каждой точки. Итоговое минимальное расстояние вычисляется как минимум из расстояний в левой, правой частях и полосе. Этот подход позволяет сократить время выполнения до O(n log n).

Результат работы кода на примерах (простой метод):



Результат работы кода на минимальных значениях (простой метод):



	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0009646415710449219	0.000244140625
Пример из задачи	0.0010001659393310547	0.0006866455078125
Пример из задачи	0.0009989738464355469	0.000946044921875
Пример из задачи	0.0009870529174804688	0.0014495849609375
Верхняя граница диапазона значений входных данных из текста задачи	11.390852928161621	4.729404449462891

Вывод по задаче: реализован алгоритм нахождения минимального расстояния между двумя точками на плоскости с использованием подхода "Разделяй и властвуй". Алгоритм продемонстрировал высокую эффективность даже для больших наборов данных.

Дополнительные задачи

Задача №2. Анти-quick sort

Для сортировки последовательности чисел широко используется быстрая сортировка - QuickSort. 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.

• Формат входного файла (input.txt). В первой строке находится единственное число n ($1 \le n \le 10^6$).

• Формат выходного файла (output.txt). Вывести перестановку чисел от 1 до n, на которой быстрая сортировка выполнит максимальное число сравнений. Если таких перестановок несколько, вывести любую из них.

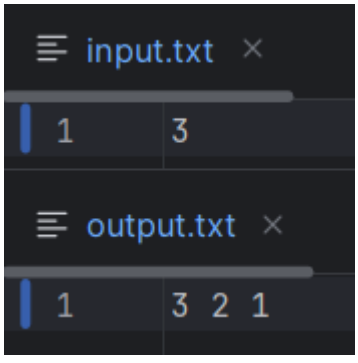
• Ограничение по времени. 2 сек.

• Ограничение по памяти. 256 мб.

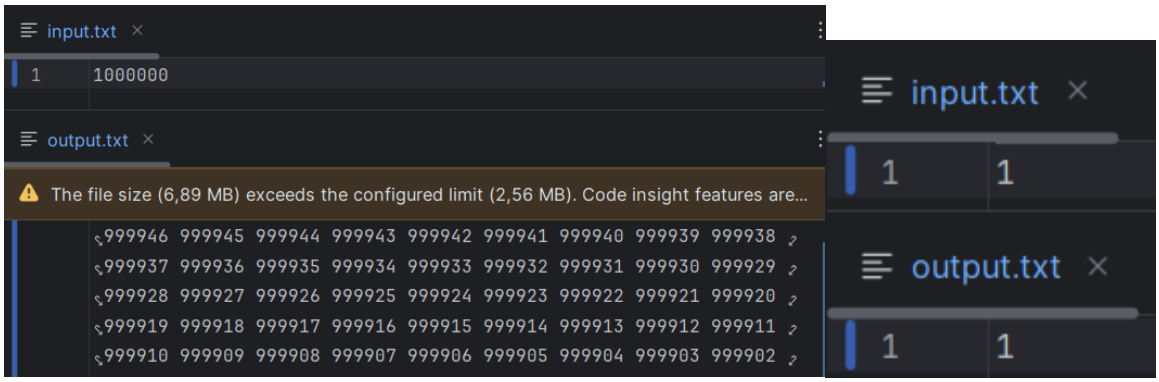
```
def generate_worst_case(n):
    n = int(n)
    return list(range(n, 0, -1))
```

Функция generate_worst_case(n) создаёт массив длины n, содержащий числа в убывающем порядке от n до 1. Это имитирует худший случай для быстрой сортировки.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.0009706020355224609	0.00014495849609375
Пример из задачи	0.0009965896606445312	0.00016021728515625
Верхняя граница диапазона значений входных данных из текста задачи	0.0010685920715332031	38.139320373535156

Вывод по задаче: создан алгоритм для генерации массива, представляющего худший случай для быстрой сортировки. Он позволяет проанализировать максимальное количество сравнений, выполняемых алгоритмом, что полезно для тестирования производительности.

Задача №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 матрешки 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 ≤ n, k ≤ 10^5) – число матрёшек и размах рук. Во второй*

строчке содержится n целых чисел, которые по модулю не превосходят 109 – размеры матрёшек.

• Формат выходного файла (output.txt). Выведите «ДА», если возможно отсортировать матрёшки по неубыванию размера, и «НЕТ» в противном случае.

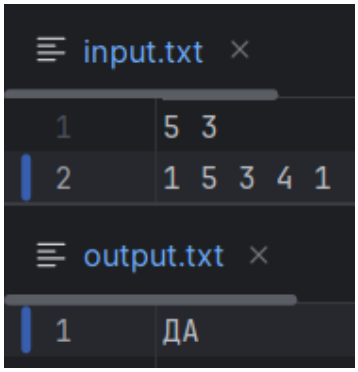
• Ограничение по времени. 2 сек.

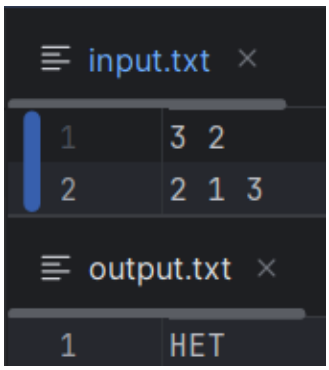
• Ограничение по памяти. 256 мб.

```
def scarecrow_sort(n, k, matr):
    groups = [[] for i in range(k)]
    for i in range(n):
        groups[i % k].append(matr[i])
    for group in groups:
        group.sort()
    sorted_matr = [groups[i % k][i // k] for i in range(n)]
    return "ДА" if sorted_matr == sorted(matr) else "НЕТ"
```

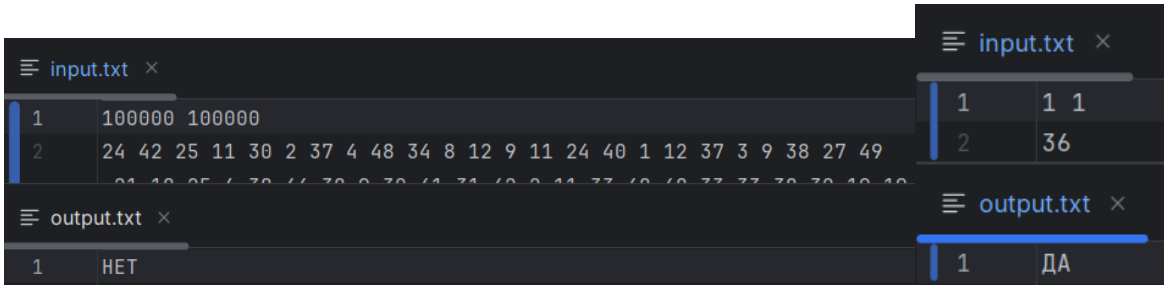
Функция scarecrow_sort делит массив matr длины n на k групп по остаткам от деления индекса на k, сортирует каждую группу, а затем собирает элементы обратно. Если полученный массив совпадает с отсортированным matr, возвращает "ДА", иначе "НЕТ".

Результат работы кода на примерах:





Результат работы кода на максимальных и минимальных значениях:



	Время выполнения, с	Затраты памяти, Mb
Нижняя граница диапазона значений входных данных из текста задачи	0.00099945068359375	0.00035858154296875
Пример из задачи	0.0009393692016601562	0.00043487548828125
Пример из задачи	0.0009949207305908203	0.00038909912109375
Верхняя граница диапазона значений входных данных из текста задачи	0.001003265380859375	11.055206298828125

Вывод по задаче: разработан алгоритм для проверки возможности сортировки массива при ограниченных условиях обмена элементов. Группировка элементов и их сортировка внутри групп обеспечивают эффективное решение задачи.

Вывод

В результате выполнения лабораторной работы я изучил и реализовал несколько алгоритмов, основанных на методе "Разделяй и властвуй". Реализация быстрой сортировки и её улучшенной версии позволила понять принципы оптимизации за счёт трёхстороннего разбиения, а изучение сортировки слиянием и сортировки пугалом продемонстрировало подходы к группировке и слиянию данных. Также я освоил алгоритмы нахождения худшего случая для сортировки и определения индекса Хирша, что укрепило навыки работы с массивами, анализом данных и оптимизацией вычислений. Кроме того, алгоритм поиска ближайшей пары точек помог изучить применение декомпозиции для минимизации расстояния, что улучшило понимание структур данных и временной сложности.