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

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: “Быстрая сортировка. Сортировки за линейное время”
Вариант 20

Выполнила:
Толстухина К.А.
К3139

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 год

Задачи по варианту

Задача №1. Улучшение Quick Sort

Текст задачи

1. Используя псевдокод процедуры Randomized - QuickSort, атак 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.
2. 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[к] < x для всех + 1 < k<M1 -1
- AIN=2 AI8 BCeX Mi ≤ K≤ m2
- A[к] > x для всех m2 + 1 ≤ k ≤r

Листинг кода

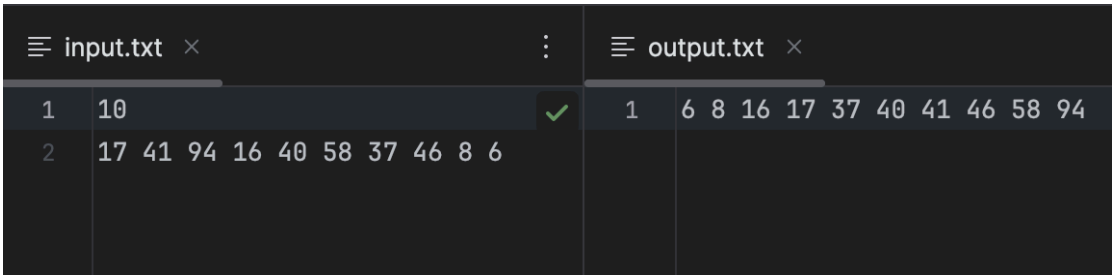
```
import random




def partition(arr, left, right):
    """Функция для разбиения массива на два
       подмассива """
    x = arr[right]
    j = left
```

```
    for i in range(left, right):
        if arr[i] <= x:
            arr[j], arr[i] = arr[i], arr[j]
            j = j + 1
    arr[right], arr[j] = arr[j], arr[right]
    return j



def randomizer_quicksort(arr, left, right):
    """Функция для сортировки массива"""
    if left >= right:
        return
    if left < right:
        k = random.randint(left, right)
        arr[right], arr[k] = arr[k], arr[right]
        m = partition(arr, left, right)
        randomizer_quicksort(arr, left, m - 1)
        randomizer_quicksort(arr, m + 1, right)
```

Текстовое объяснение

Я реализую две функции, одну для разбиения массива, вторую непосредственно для быстрой сортировки. Для начала из массива выбирается рандомный элемент, после чего он ставиться на последнее место, затем запускается функция разбиения и рекурсивно вызывается функция сортировки для каждой части. В результате массив отсортирован.

Пример работы(скрин файлов)



	Время	Память
Нижняя граница	0.0012236250331625342 с	0.013301849365234375 MB
Верхняя граница	0.008264375035651028 с	0.15083026885986328 MB

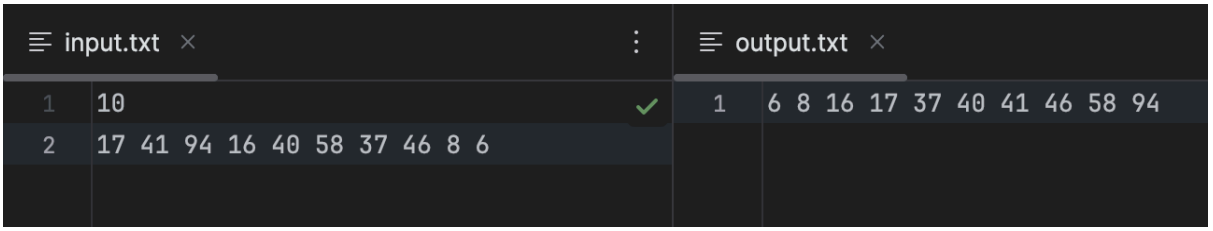
2)

Листинг кода

```
import random


def partition(arr, left, right):
    """функция разбиения массива на три части"""
    x = arr[left]
    m1 = left
    m2 = right
    i = left + 1
    while i <= m2:
        if arr[i] < x:
            m1 += 1
            arr[m1], arr[i] = arr[i], arr[m1]
            i += 1
        elif arr[i] > x:
            arr[i], arr[m2] = arr[m2], arr[i]
            m2 -= 1
        else:
            i += 1
    arr[left], arr[m1] = arr[m1], arr[left]
    return m1, m2


def randomizer_quicksort(arr, left, right):
    """функция рандомизированной сортировки массива"""
    if left < right:
        k = random.randint(left, right)
        arr[left], arr[k] = arr[k], arr[left]
        m1, m2 = partition(arr, left, right)
        randomizer_quicksort(arr, left, m1 - 1)
```

```
        randomizer_quicksort(arr, m2 + 1, right)
```

Текстовое объяснение
Алгоритм отличается от предыдущего только разбиением на три части, вместо двух. Я реализую это через цикл while, где меняю элементы в зависимости от того, больше, меньше или равны они исходному.
Пример(скрин файлов)



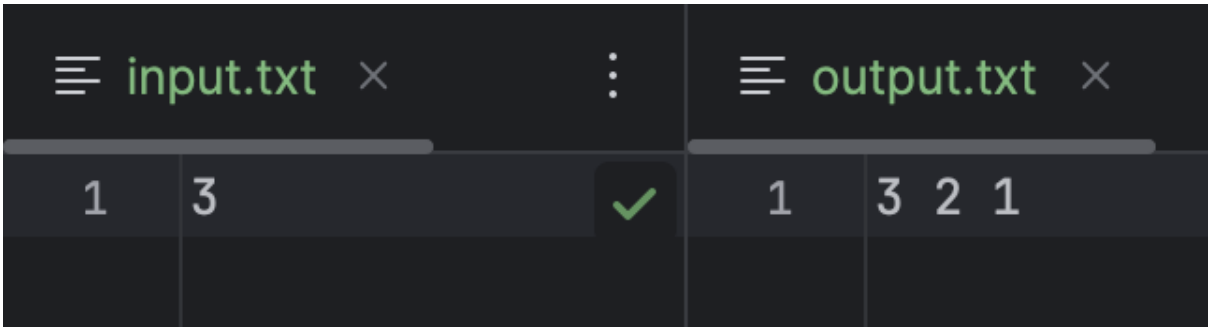
	Время	Память
Нижняя граница	0.007112833089195192 с	0.013301849365234375 MB
Верхняя граница	0.012634753456554328 с	0.158962004975627458 MB

Вывод: алгоритм быстрой сортировки действительно работает быстро даже на больших массивах данных.

Задача №2 Анти-quick sort
Текст задачи
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

```
def generation_test_for_max_swap(n):
"""функция генерации теста"""
   return reversed(sorted(range(1, n + 1)))
```

Наибольшее количество сравнений происходит, когда массив уже отсортирован в обратном порядке. Если массив отсортирован в обратном порядке, то QuickSort будет работать, выполняя O(n^2) сравнений, поскольку каждый выбор опорного элемента будет делить массив только на одну часть, заставляя рекурсию работать на почти всех элементах. Таким образом - это наихудший вариант. Поэтому я генерирую массив от 1 до n и сортирую его в обратном направлении.
Пример



Здесь будет 2 перестановки

	Время	Память
Нижняя граница	0.0007582500111311674 с	0.013178825378417969 MB
Верхняя граница	1.43760874995496124 с	65.08500671386719 MB

Вывод: самый наихудший случай для быстрой сортировки, когда массив отсортирован в обратном порядке.

Задача №9. Ближайшие точки
Текст задачи
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.

Листинг кода

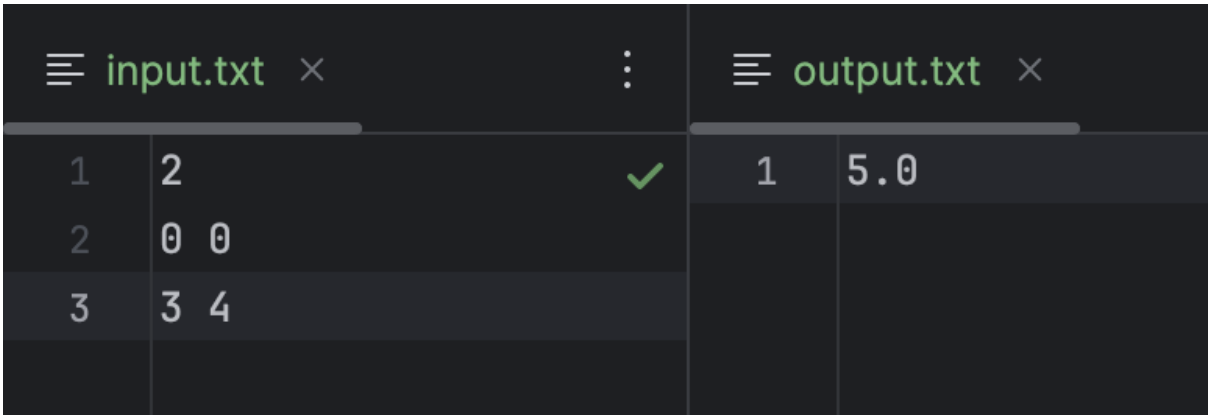
```
import math


def dist(coor1, coor2):
    """Функция для нахождения расстояния между
координатами"""
    return math.dist(coor1, coor2)


def recurse(coor_x, coor_y):
    """Функция, которая рекурсивно, с помощью метода
"разделяй и властвуй" находит
    минимальное расстояние между точками"""
    n = len(coor_x)
    if n <= 3:
        return round(min(dist(coor_x[i], coor_x[j])
for i in range(n) for j in range(i + 1, n)), 7)

    mid = n // 2

    left_part_coor_x = coor_x[:mid]
    right_part_coor_x = coor_x[mid:]

    left_part_coor_y = [c for c in coor_y if c[0] <=
coor_x[mid][0]]
    right_part_coor_y = [c for c in coor_y if c[0] >
coor_x[mid][0]]

    left_dist = recurse(left_part_coor_x,
left_part_coor_y)
    right_dist = recurse(right_part_coor_x,
right_part_coor_y)

    min_dist = min(left_dist, right_dist)

    string = [c for c in coor_y if abs(c[0] -
coor_x[mid][0]) < min_dist]
```

```
    if len(string) <= 1:
        return round(min(dist(coor_x[i], coor_x[j])
for i in range(n) for j in range(i + 1, n)), 7)


   return round(min(min_dist, min(dist(string[i],
string[j]) for i in range(len(string)) for j in
range(i + 1, min(i + 7, len(string))))), 7)


def min_distan(coord):
   """Функция которая сортирует координаты"""
   coor_x = sorted(coord, key=lambda x: x[0])
   coor_y = sorted(coord, key=lambda y: y[1])
   return recurse(coor_x, coor_y)
```

Текстовое объяснение

Для начала в функции `min_distan` я сортирую координаты по иксу и по игреку, затем вызываю рекурсивный алгоритм, в который передаю оба массива. В `recurse` в случае, если массив менее 3 элементов, “ручной” перебор не потратит много времени, поэтому я использую его. Если массив больше трех элементов, то я нахожу средний, делю его на две части по иксу и две по игреку и вызываю этот же алгоритм для обоих частей. Нахожу минимальную из обеих. Затем необходимо проверить есть ли точки, подходящие под условия, из разных частей. Проверяю это с помощью вложенных списков, с нахождением минимального элемента. Который в итоге возвращаю.

Тесты(скрины файлов)



input.txt

```
4
7 7
1 100
2 4
7 7
```

output.txt

```
0.0
```

input.txt

```
11
4 4
-2 -2
-3 -4
-1 3
2 3
-4 0
1 1
-1 -1
3 -1
-4 2
-2 4
```

output.txt

```
2.0
```

	время	память
тест из условия	0.0004500419599935 4124 с	0.0133209228515625 MB

тест из условия	0.0005300829652696848 с	0.013418197631835938 MB
тест из условия	0.0007322090677917004 с	0.013824462890625 MB
максимальные данные	13.705668749986216 с	1.8246431350708008 MB

Вывод: метод разделяй и властвуй, работает намного быстрее.

Дополнительные задачи

Задача №8. K ближайших точек к началу координат

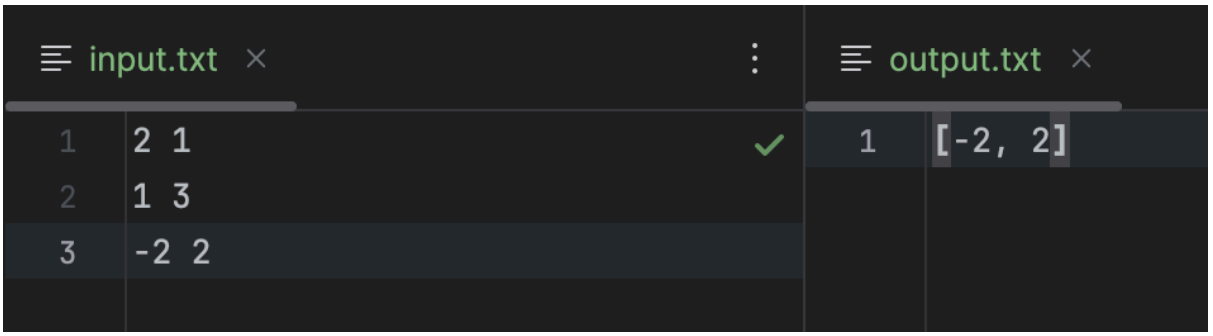
Текст задачи

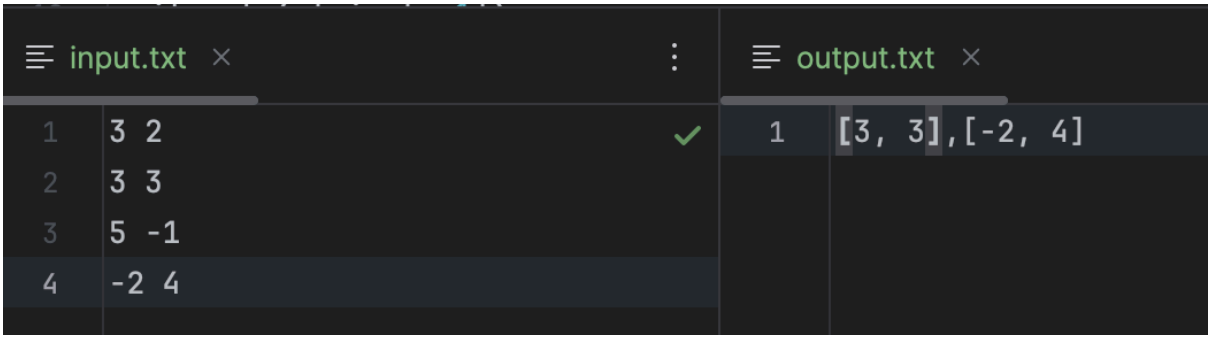
В этой задаче, ваша цель - найти K ближайших точек к началу координат среди данных n точек. • Цель. Заданы и точек на поверхности, найти K точек, которые находятся ближе к началу координат (0, 0), т.е. имеют наименьшее расстояние до начала координат. Напомним, что расстояние между двумя точками (X1, У1) и (X2, У2) равно V (x1 - X2)? + (У1 - У2)?

Листинг кода

```
def dist(coor1: (float, float)) -> float:
    """Функция для нахождения расстояние между точкой и началом координат"""
    return coor1[0] ** 2 + coor1[1] ** 2


def count(coor: (float, float), k: int) -> [float]:
    """Функция для подсчета количества точек"""
    return [i[1] for i in sorted([(dist(cor), [cor[0], cor[1]]) for cor in coor])[:k]]
```

Текстовое объяснение
Для начала я создаю функцию для вычисления расстояния, так как вторая точка (0, 0), то это делается по формуле x^2 + y^2. Далее, я создаю функцию, где сначала нахожу расстояние между точками и началом координат и записываю в массив в виде кортежа (расстояние, координаты), далее сортирую массив по первому элементу(по умолчанию по первому), делаю срез по первым K элементам и через цикл записываю в итоговый массив только координаты. Возвращаю этот массив.
Тесты(скрины файлов)





	время	память
тест 1	0.0008640419691801071 с	0.013367652893066406 MB
тест 2	0.0003246249398216605 с	0.013416290283203125 MB

Вывод: программа работает быстро

Задача №5. Индекс Хирша

Текст задачи

Для заданного массива целых чисел citations, где каждое из этих чисел - число цитирований i-ой стать ученого-исследователя, посчитайте индекс Хирша этого ученого.

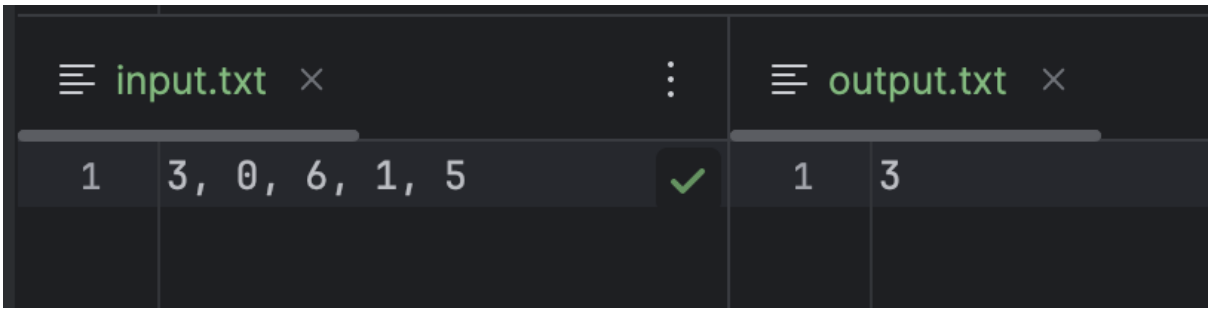
Листинг кода

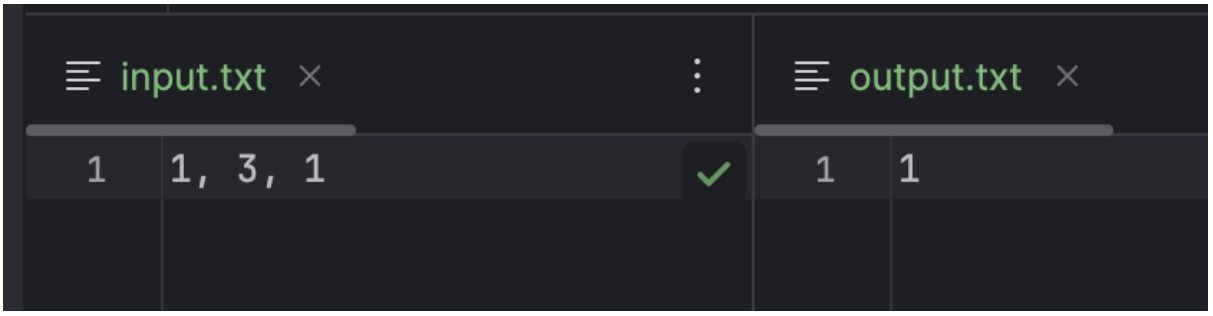
```
def find_index_h(arr: [int]) -> int:
    """Функция для нахождения индекса Хирша"""
    arr.sort(reverse=True)
    for i, count_citation in enumerate(arr, 1):
        if count_citation < i:
            return i - 1
    return len(arr)
```

Объяснение

Для поиска индекса Хирша я сначала сортирую массив по убыванию. Потом прохожу циклом по массиву, разбивая его на элементы и индексы этих элементов, начиная с 1. Когда, номер элемента становится больше, чем сам элемент - это и есть индекс Хирша, так как все элементы выше этого, будут точно больше(массив отсортирован) и таким образом h будет равно количеству статей, которые цитируются не меньше, чем h раз.

Тесты(скрины файлов)





	время	память
тест 1	0.0009081250755116343	0.013230323791503906 MB
тест 2	0.0005589999491348863	0.013218879699707031 MB
max	0.004180958028882742	0.3917818069458008 MB

При максимальных значениях для n, алгоритм работает так же быстро. Однако если бы массив изначально был отсортирован, время бы было гораздо быстрее. Время алгоритма O(n)