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: Тема работы
Вариант 1

Выполнил:
Барецкий М.С.
К3141

Проверила:
Афанасьев А. А.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета **2**

Задачи по варианту **3**

Задача №1. Улучшение Quick sort

Задача №1. Улучшение Quick sort

Задача №1. Улучшение Quick sort 3

Дополнительные задачи **4**

Задача №4. Бинарный поиск 6

Задача №5. Представитель большинства 7

Задача №7. Поиск максимального подмассива за линейное время 8

10

Вывод **5**

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:

```
import random


def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i = i + 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return (i + 1)



def randomized_quick_sort(arr, low, high):
    if len(arr) == 1:
        return
    if low < high:
        pi = random.randint(low, high)
        arr[pi], arr[high] = arr[high], arr[pi]
        pi = partition(arr, low, high)
```

```
        randomized_quick_sort(arr, low, pi - 1)
        randomized_quick_sort(arr, pi + 1, high)
    return arr
```

alg_lab3 > task1 > textf > input.txt

```
1   5
2   2 3 9 2 2
```

alg_lab3 > task1 > textf > output.txt

```
1   2 2 2 3 9
```

Этот код реализует **рандомизированную быструю сортировку (Quick Sort)**, где на каждом шаге случайным образом выбирается опорный элемент, после чего массив делится на две части с помощью функции partition. Элементы меньше опорного ставятся влево, а большие — вправо. Функция randomized_quick_sort рекурсивно сортирует массив, улучшая стабильность алгоритма, снижая вероятность худшего случая. Время работы в среднем — **O(n log n)**.

	Время выполнения	Затраты памяти
Пример из задачи(Без бесконечности)	0.000606500001595122 7	0.017294883728027344

Задача №5. Улучшение Quick sort

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

```
def h_index(citations):
    citations.sort(reverse=True)
    h = 0
    for i, c in enumerate(citations):
        if c > i:
            h = i + 1
        else:
            break
    return h
```

alg_lab3 > task5 > textf > input.txt

```
1    3 6 5 0 1
```

alg_lab3 > task5 > textf > output.txt

```
1    3
```

Этот код вычисляет **индекс Хирша (h-index)**, сортируя список цитирований по убыванию и затем проверяя, сколько работ имеют хотя бы такое количество цитирований, как их порядковый номер. Индекс Хирша — это максимальное значение h, при котором ученый имеет хотя бы h работ, цитируемых не менее h раз. Если условие не выполняется, подсчет останавливается.

	Время выполнения	Затраты памяти
Пример из задачи(Без бесконечности)	0.0007449000004271511	0.0172271728515625

Дополнительные задачи

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?

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0
    while low <= high:
        mid = (high + low) // 2
        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return mid
```

```
    return -1
```

alg_lab3 > task3 > textf > input.txt

```
1   3 2
2   2 1 3
```

alg_lab3 > task3 > textf > output.txt

```
1   No
```

Этот код реализует сортировку с использованием метода сортировки пугалом, который распределяет элементы массива в k групп, сортирует каждую группу, а затем собирает элементы обратно. После этого проверяется, совпадает ли результат с обычной сортировкой массива. Если совпадает, возвращается Yes, иначе — No.

	Время выполнения	Затраты памяти
Пример из задачи	0.0007777000027999748	0.01742267608642578

Задача №4. Точки и отрезки

Допустим, вы организовываете онлайн-лотерею. Для участия нужно сделать ставку на одно целое число. При этом у вас есть несколько интервалов последовательных целых чисел. В этом случае выигрыш участника пропорционален количеству интервалов, содержащих номер участника, минус количество интервалов, которые его не содержат. (В нашем случае для начала - подсчет только количества интервалов, содержащих номер участника). Вам нужен эффективный алгоритм для расчета выигрышей для всех участников. Наивный способ сделать это - просто просканировать для всех участников список всех интевалов. Однако ваша лотерея очень популярна: у вас тысячи участников и тысячи интервалов. По этой причине вы не можете позволить себе медленный наивный алгоритм.

```
def count_intervals(data):
    segment_count, point_count = data[0][0], data[0][1]
    intervals = [x for x in data[1:1 + segment_count]]
    points = data[-1]

    events = []
    point_results = {}

    for start, end in intervals:
        events.append([start, "L"])  # "L" для левого конца
        events.append([end, "R"])    # "R" для правого конца

    for point in points:
        events.append([point, "P"])  # "P" для точки
        point_results[point] = 0

    events.sort()

    active_segments = 0
    for position, event_type in events:
        if event_type == "L":
            active_segments += 1
        elif event_type == "R":
            active_segments -= 1
        elif event_type == "P":
            point_results[position] = active_segments

    return [point_results[point] for point in points]
```

Этот код решает задачу подсчета количества отрезков, которые содержат каждую точку из заданного множества точек. Для этого он создает события для начала, конца отрезков и каждой точки, затем сортирует их и проходит по отсортированным событиям, поддерживая количество активных отрезков в данный момент. Для каждой точки сохраняется количество активных отрезков, которые её содержат. В конце возвращается список, показывающий количество отрезков для каждой точки.

alg_lab3 > task4 > textf > input.txt

```
1 3
-10 10
-100 100 0
```

alg_lab3 > task4 > textf > output.txt

```
0 0 1
```

alg_lab3 > task4 > textf > input.txt

```
2 3
0 5
7 10
1 6 11
```

alg_lab3 > task4 > textf > output.txt

```
1 0 0
```

	Время выполнения	Затраты памяти
Пример из задачи	0.007017500000074506	0.017240524291992188
Пример из задачи	0.005433499998616753	0.017192840576171875

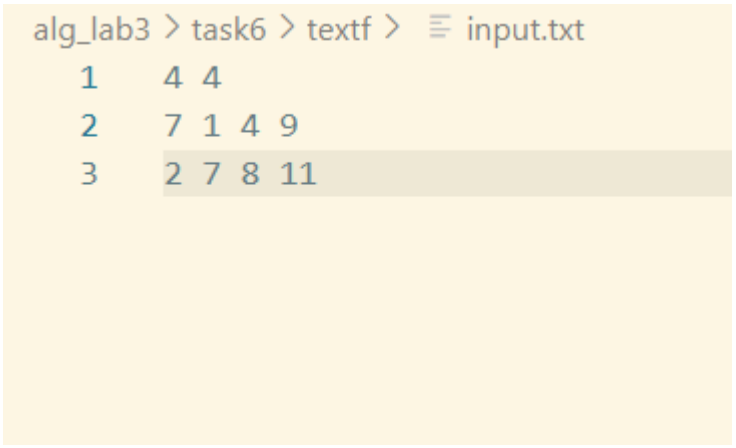
Задача №6. Сортировка целых чисел

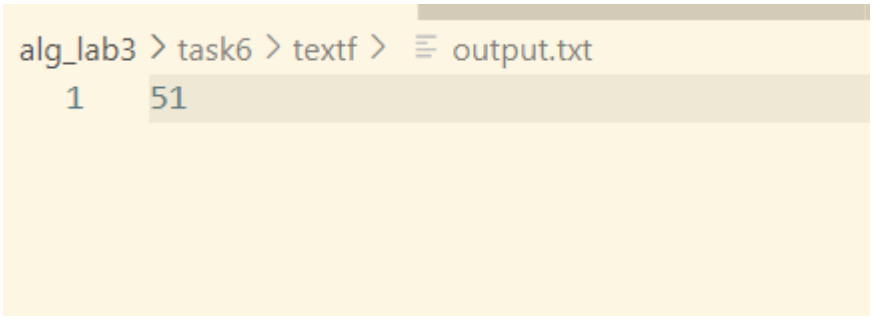
В этой задаче нужно будет отсортировать много неотрицательных целых чисел. Вам даны два массива, A и B, содержащие соответственно n и m элементов. Числа, которые нужно будет отсортировать, имеют вид Ai · Bj , где 1 ≤ i ≤ n и 1 ≤ j ≤ m. Иными словами, каждый элемент первого массива нужно умножить на каждый элемент второго массива. Пусть из этих чисел получится отсортированная последовательность C длиной n · m. Выведите сумму каждого десятого элемента этой последовательности (то есть, C1 + C11 + C21 + ...).

```python
from alg_lab3.task1.src.task1 import randomized_quick_sort

def sum_of_every_tenth_product(array_a, array_b):
    product_list = []
    for b in array_b:
        for a in array_a:
            product_list.append(a * b)

    randomized_quick_sort(product_list, 0, len(product_list) - 1)
        sum_tenth_elements = sum(product_list[i] for i in range(0,
len(product_list), 10))

    return sum_tenth_elements
```

```python
if __name__ == '__main__':
    result = sum_of_every_tenth_product()
    print(result)  # Вывод результата, если нужно
```

Этот код вычисляет сумму каждого десятого произведения элементов двух массивов array_a и array_b. Для этого он сначала генерирует список всех произведений, сортирует его с помощью рандомизированной быстрой сортировки, а затем суммирует элементы, стоящие на позициях, кратных 10. Результат выводится после выполнения функции sum_of_every_tenth_product().





	Время выполнения	Затраты памяти
Пример из задачи	0.002208999998401850	0.01723957061767578

	5	

Задача №8. K ближайших точек к началу координат

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

```
def find_k_nearest_points(points, k):
    points.sort(key=lambda p: p[0]**2 + p[1]**2)
    return points[:k]
```

Этот код находит **k ближайших точек** к началу координат из списка точек. Точки сортируются по квадрату их расстояния от начала (расстояние вычисляется как x2+y2x^2 + y^2x2+y2, без извлечения квадратного корня для упрощения вычислений). Затем возвращаются первые k точек с наименьшим расстоянием.

alg_lab3 > task8 > textf > input.txt

```
3 2
3 3
5 -1
-2 4
```

alg_lab3 > task8 > textf > output.txt

```
[3,3] [-2,4]
```

	Время выполнения	Затраты памяти
Пример из задачи	0.004082800001924625	0.017505645751953125

Вывод по лабораторной работе №3

В ходе выполнения лабораторной работы были рассмотрены различные алгоритмические задачи, включая улучшение быстрой сортировки, подсчет индекса Хирша, задачи, связанные с сортировкой и поиском, а также оптимизация работы с интервалами и точками.

Лабораторная работа позволила закрепить знания о различных алгоритмах сортировки, поиска и работы с данными, а также научиться эффективно использовать такие алгоритмы для решения практических задач.