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 1

Выполнил:
Бен Шамех Абделазиз
Группа: K3239

Проверила:
Ромакина Оксана Михайловна

Санкт-Петербург
2025 г.

Содержание

1 Задача 1. Улучшение Quick sort

1.1 Условие

Цель задачи — переделать рандомизированный алгоритм быстрой сортировки, чтобы он эффективно обрабатывал последовательности с несколькими уникальными элементами, используя трехстороннее разделение.

1.2 Листинг кода

```python
import random

def quick_sort_p3(arr, l, r):
    if l < r:
        lt, gt = partition3(arr, l, r)
        quick_sort_p3(arr, l, lt - 1)
        quick_sort_p3(arr, gt + 1, r)

def partition3(arr, l, r):
    x_ind = random.randint(l, r)
    arr[l], arr[x_ind] = arr[x_ind], arr[l]
    x = arr[l]
    lt = l
    eq = l
    gt = r
    while eq <= gt:
        if arr[eq] < x:
            arr[lt], arr[eq] = arr[eq], arr[lt]
            lt += 1
            eq += 1
        elif arr[eq] > x:
            arr[gt], arr[eq] = arr[eq], arr[gt]
            gt -= 1
        else:
            eq += 1
    return lt, gt
```

1.3 Объяснение решения

Разделяем массив на числа большие, меньшие и равные заданному «pivot» и находим границы разделения. Рекурсивно делим массив по этим разделителям.

1.4 Результаты выполнения

Пример из задачи:

- **Input:** `n=5, [12, 3, 24, 16, 29]`
- **Output:** `[3, 12, 16, 24, 29]`

Метрики эффективности:

Сценарий	**Время (сек)**	**Память (Mb)**
Нижняя граница (n=2)	0.0094 sec	30.43 Mb
Пример из задачи	0.00894 sec	30.25 Mb
Верхняя граница (n=100000)	0.16941 sec	31.46 Mb

2 Задача 2. Анти-quick sort

2.1 Условие

Написать программу, генерирующую тест (перестановку чисел от 1 до n), на котором алгоритм быстрой сортировки выполнит максимальное число сравнений.

2.2 Листинг кода

```
def generate_worst_case(ln):
    arr = []
    for i in range(0, ln):
        arr += [i + 1]
        if i > 1:
            arr[-1], arr[i // 2] = arr[i // 2], arr[-1]
    return arr
```

2.3 Результаты выполнения

Пример (n=5):

- **Output:** `[1, 4, 5, 3, 2]`

Метрики эффективности:

Сценарий	**Время (сек)**	**Память (Mb)**
Нижняя граница (n=2)	0.00451 sec	30.24 Mb
Пример из задачи	0.00173 sec	30.23 Mb
Верхняя граница (n=100000)	0.08889 sec	31.90 Mb

3 Задача 4. Точки и отрезки

3.1 Условие

Вычислить для каждой точки количество отрезков, содержащих эту точку.

3.2 Результаты выполнения

Метрики эффективности:

Сценарий	**Время (сек)**	**Память (Mb)**
Нижняя граница	0.0055 sec	30.16 Mb
Пример из задачи	0.00554 sec	30.25 Mb
Верхняя граница	0.77409 sec	31.79 Mb

4 Задача 6. Сортировка целых чисел

4.1 Условие

Отсортировать массив попарных произведений элементов массивов A и B и вывести сумму каждого десятого элемента.

4.2 Результаты выполнения

Метрики эффективности:

Сценарий	**Время (сек)**	**Память (Mb)**
Нижняя граница	0.0055 sec	30.16 Mb
Пример из задачи	0.00563 sec	30.57 Mb
Верхняя граница	0.77409 sec	31.79 Mb

5 Задача 7. Цифровая сортировка

5.1 Условие

Дано n строк, выведите их порядок после k фаз цифровой сортировки.

5.2 Результаты выполнения

- **Input:** `3 3 1 bab bba baa` → **Output:** `2 3 1`
- **Input:** `3 3 2 bab bba baa` → **Output:** `3 2 1`

Метрики эффективности:

Сценарий	**Время (сек)**	**Память (Mb)**
Нижняя граница	0.002 sec	30.15 Mb
Пример из задачи	0.003 sec	30.15 Mb
Верхняя граница	2.15 sec	215.40 Mb

6 Задача 8. K ближайших точек к началу координат

6.1 Условие

Заданы n точек на поверхности, найти K точек, которые находятся ближе всего к началу координат $(0, 0)$.

6.2 Листинг кода

```
def find_k_closest_points(points, k):
    points = sorted([[a, b, find_dist(a, b)] for a, b in points], key=
   lambda x: x[2])
    points = points[:k]
    ans = ''
    for i in range(len(points) - 1):
        a, b, c = points[i]
        ans += f'[{a}, {b}], '
    a, b, c = points[-1]
    ans += f'[{a}, {b}]'
    return ans

def find_dist(dot1, dot2):
    return (dot1**2 + dot2**2)**0.5
```

6.3 Объяснение решения

Подсчитаем для каждой точки расстояние до $(0, 0)$ и отсортируем по этому ключу, затем обрежем массив до k элементов.

6.4 Результаты выполнения

Пример:

- **Output:** `[(3, 2), (3, 3), (5, -1), (-2, 4)]`

Метрики эффективности:

Сценарий	Время (сек)	Память (Mb)
Нижняя граница (n=1, k=1)	0.00312 sec	30.20 Mb
Пример из задачи	0.00264 sec	30.13 Mb
Верхняя граница (n=100000, k=1000)	0.75601 sec	30.20 Mb

7 Задача 9. Ближайшие точки

7.1 Условие

Найти пару ближайших точек среди данных n точек на плоскости. Ответ вывести с 4 знаками после запятой.

7.2 Листинг кода

```python
def recursion_pair_closest(arr_s_x, arr_s_y):
    if len(arr_s_x) <= 3:
        min_dist = 10**18
        best_pair = ()
        for i in range(len(arr_s_x)):
            for j in range(i + 1, len(arr_s_x)):
                min_d = find_dist(arr_s_x[i], arr_s_x[j])
                if min_d < min_dist:
                    min_dist = min_d
                    best_pair = (arr_s_x[i], arr_s_x[j])
        return min_dist, best_pair
    # ... logic for divide and conquer ...
    # (Full code implementation here)
```

7.3 Объяснение решения

Используется метод «раздели и властвуй». Массив сортируется по X-координате, делится пополам, рекурсивно ищется минимальное расстояние в каждой половине и на стыке (полоса шириной d).

7.4 Результаты выполнения

Пример:

- **Output:** `[(4, 4), (-2, -2), (-3, -4), ...]`

Метрики эффективности:

Сценарий	Время (сек)	Память (Mb)
Нижняя граница (n=1)	0.00153 sec	30.19 Mb
Пример из задачи	0.00253 sec	30.49 Mb
Верхняя граница (n=100000)	0.81082 sec	30.41 Mb

8 Вывод

1. В этой лабораторной работе были изучены алгоритмы быстрой сортировки (*Quick Sort*), её улучшенная версия с трехсторонним разделением (*Partition 3*), а также алгоритмы цифровой сортировки (*Radix Sort*) и нахождения минимального расстояния между точками за $O(n \log n)$.

2. Было практически подтверждено, что алгоритмы со сложностью $O(n)$ и $O(n \log n)$ выполняются значительно быстрее квадратичных алгоритмов на больших входных данных.

3. С помощью методов `time.perf_counter()` и `psutil.Process().memory_info().rss` была проведена оценка ресурсозатратности реализованных алгоритмов.