

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: Сортировка вставками, выбором, пузырьковая

Выполнил:  
Марченко В.А.  
К3141

Проверил:  
Афанасьев А.В.

Санкт-Петербург  
2024 г.

## Содержание отчета

Содержание отчета.....	2
Задания .....	3
Задание 1. Сортировка вставкой .....	3
Задание 2. Сортировка вставкой +.....	5
Задание 3. Сортировка вставкой по убыванию .....	7
Задание 4. Линейный поиск .....	9
Задание 5. Сортировка выбором .....	11
Задание 10. Палиндром.....	13
Вывод.....	16

## Задания

### Задание 1. Сортировка вставкой

В данной задаче требуется отсортировать массив чисел из файла с помощью сортировки вставкой (Insertion-sort).

Листинг кода:

```
def insertionsort(input_name, output_name):
    input_file = open(input_name, 'r')
    output_file = open(output_name, 'w')

    n = int(input_file.readline())
    arr = list(map(int, input_file.readline().split()))

    for i in range(1, n):
        key = arr[i]
        j = i - 1

        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

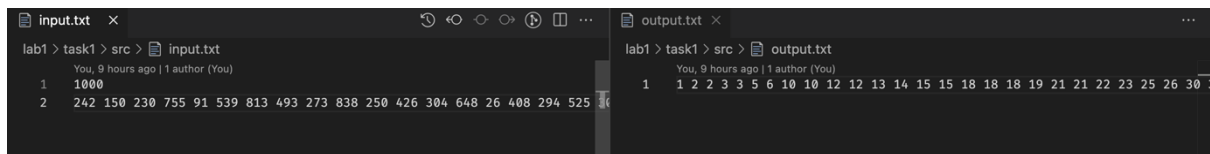
    output_file.write(' '.join(map(str, arr)))

    input_file.close()
    output_file.close()
```

Текстовое объяснение решения:

Сначала открываются два файла: один для чтения исходных данных, другой – для записи отсортированных результатов. Из входного файла считывается первое число, которое указывает количество элементов в массиве, затем сам массив чисел. После этого выполняется сортировка методом вставки: для каждого элемента массива, начиная со второго, находится его место среди уже отсортированных элементов слева. Внутренний цикл сдвигает элементы вправо, пока не найдётся позиция для текущего элемента. После завершения сортировки, отсортированный массив преобразуется в строку и записывается в выходной файл. В конце файлы закрываются.

## Результат работы кода:



The screenshot shows a code editor with two tabs: 'input.txt' and 'output.txt'. The 'input.txt' tab is active, showing a list of numbers: 1000, 242, 150, 230, 755, 91, 539, 813, 493, 273, 838, 250, 426, 304, 648, 26, 408, 294, 525. The 'output.txt' tab is also visible, showing a list of numbers: 1, 2, 2, 3, 3, 5, 6, 10, 10, 12, 12, 13, 14, 15, 15, 18, 18, 18, 19, 21, 21, 22, 23, 25, 26, 30.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00036 ms	0.01805 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.18421 ms	0.11822 MB

## Вывод по задаче:

Таким образом, в этой задаче мы изучили и реализовали сортировку вставкой. Алгоритм работает эффективно для небольших наборов данных.

## Задание 2. Сортировка вставкой +

В данной задаче требуется отредактировать код из прошлого задания так, чтобы в первой строчке файла от так же выводил для каждого числа новый индекс, соответствующего ниже элемента.

Листинг кода:

```
def insertionsort_indexes(input_name, output_name):
    input_file = open(input_name, 'r')
    output_file = open(output_name, 'w')

    n = int(input_file.readline())
    arr = list(map(int, input_file.readline().split()))
    indexes = list(range(1, n + 1))

    for i in range(1, n):
        key = arr[i]
        j = i - 1

        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
        indexes[i] = j + 2

    output_file.write(' '.join(map(str, indexes)) + "\n")
    output_file.write(' '.join(map(str, arr)))

    input_file.close()
    output_file.close()
```

Текстовое объяснение решения:

Сначала открываются входной и выходной файлы: первый для чтения исходных данных, второй для записи результатов. Из входного файла считывается количество элементов массива и сам массив, после чего создаётся массив `indexes`, содержащий начальные индексы каждого элемента. Затем начинается сортировка методом вставки, при которой каждый элемент вставляется на своё место в отсортированной части массива. При этом одновременно обновляются соответствующие индексы в массиве `indexes`, чтобы отслеживать новые позиции элементов. Индекс

обновляется таким образом, чтобы он отражал положение элемента в отсортированном массиве с учётом того, что индексация начинается с 1. После завершения сортировки в выходной файл записываются два ряда данных: в первой строке – новые индексы элементов, а во второй – сам отсортированный массив.

Результат работы кода:

```
input.txt  ×
lab1 > task2 > src > input.txt
You, 10 hours ago | 1 author (You)
1 10
2 1 8 4 2 3 7 5 6 9 0

output.txt  ×
lab1 > task2 > src > output.txt
You, 10 hours ago | 1 author (You)
1 1 2 2 2 3 5 5 6 9 1
2 0 1 2 3 4 5 6 7 8 9
```

Вывод по задаче:

В этой задаче мы успешно реализовали сортировку вставки с отслеживанием индексов элементов, что помогает проследить, как менялись индексы элементов в ходе сортировки.

### Задание 3. Сортировка вставкой по убыванию

В данной задаче требуется отредактировать код из задания 1 так, чтобы сортировка происходила в невозрастающем порядке. Так же необходимо использовать операцию Swap.

Листинг кода:

```
def insertionsort_reversed(input_name, output_name):
    input_file = open(input_name, 'r')
    output_file = open(output_name, 'w')

    n = int(input_file.readline())
    arr = list(map(int, input_file.readline().split()))

    for i in range(1, n):
        j = i
        while j > 0 and arr[j - 1] < arr[j]:
            arr[j], arr[j - 1] = arr[j - 1], arr[j]
            j -= 1

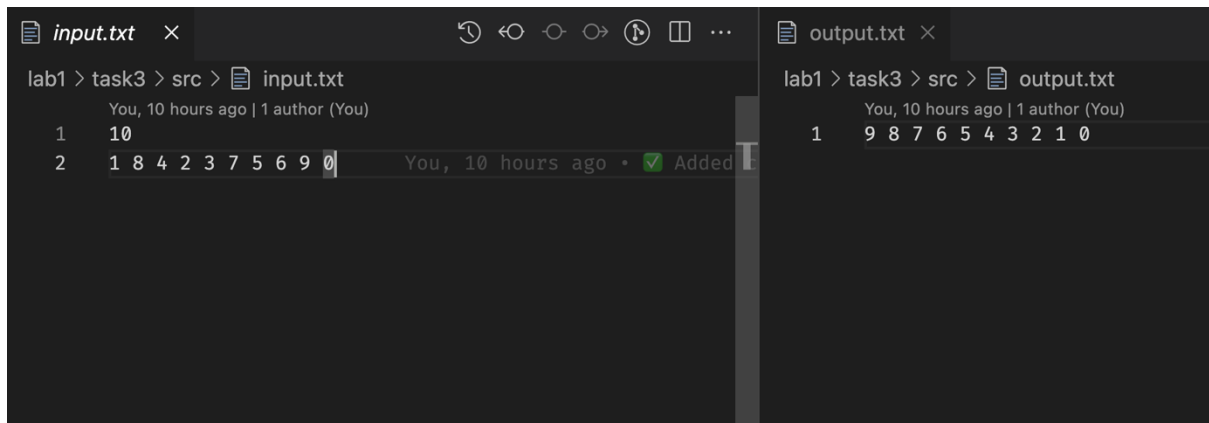
    output_file.write(' '.join(map(str, arr)))

    input_file.close()
    output_file.close()
```

Текстовое объяснение решения:

Сначала открываются входной и выходной файлы, считывается количество элементов и сам массив чисел. Затем начинается сортировка: для каждого элемента начиная со второго, выполняется цикл, который перемещает элемент на правильное место среди уже отсортированных элементов слева, если он больше предыдущего элемента. Для перемещения используется операция Swap, которая меняет местами текущий элемент и предыдущий, если первый больше. После завершения сортировки отсортированный массив записывается в выходной файл.

Результат работы кода:



The screenshot shows a code editor with two tabs: `input.txt` and `output.txt`. Both tabs show the same file path: `lab1 > task3 > src > [filename].txt`. The `input.txt` tab displays the following content:

```
1 10
2 1 8 4 2 3 7 5 6 9 0
```

The `output.txt` tab displays the following content:

```
1 9 8 7 6 5 4 3 2 1 0
```

Вывод по задаче:

В этой задаче мы успешно поменяли порядок сортировки вставкой, а также узнали о том, как мы можем в Python менять местами элементы с помощью операции Swap.



#### Задание 4. Линейный поиск

Необходимо написать код линейного поиска, которые ищет заданный элемент  $V$  в последовательности. Если такого элемента нет, то выводится  $-1$ . Если один такой элемент есть, выводится его индекс в последовательности. Если их несколько, то сначала выводится сколько раз встречается число, а потом индексы равных элементов через запятую.

Листинг кода:

```
def linearsearch(input_name, output_name):
    input_file = open(input_name, 'r')
    output_file = open(output_name, 'w')

    numbers = list(map(int, input_file.readline().split()))
    v = int(input_file.readline())

    repeat_indexes = []
    for i in range(len(numbers)):
        if numbers[i] == v:
            repeat_indexes.append(i)

    if len(repeat_indexes) == 0:
        output_file.write("-1")
    elif len(repeat_indexes) == 1:
        output_file.write(str(repeat_indexes[0]))
    else:
        output_file.write("{}\n".format(len(repeat_indexes)))
        output_file.write(','.join(map(str, repeat_indexes)))

    input_file.close()
    output_file.close()
```

Текстовое объяснение решения:

Сначала открываются файлы для чтения и записи данных. Из входного файла считываются последовательность чисел и искомое число  $V$ . Затем происходит линейный проход по последовательности, в ходе которого индексы всех найденных вхождений числа  $v$  добавляются в список `repeat_indexes`. Если число не встречается, программа выводит  $-1$ . Если найдено одно вхождение, выводится индекс этого элемента. Если найдено

несколько вхождений, сначала выводится их количество, а затем индексы всех вхождений через запятую. В конце файлы закрываются.

Результаты работы кода:

```
lab1 > task4 > src > input.txt
You, 11 hours ago | 1 author (You)
1 1 2 3 4 5 6 7 8 9 0
2 8
You, 11 hours ago • ✓ Added codes for lab1

lab1 > task4 > src > output.txt
You, 11 hours ago | 1 author (You)
1 7

lab1 > task4 > src > input.txt
You, 1 second ago | 1 author (You)
1 1 2 3 4 5 6 7 8 9 0 8 8 9
2 8

lab1 > task4 > src > output.txt
You, 1 second ago | 1 author (You)
1 3
2 7,10,11

lab1 > task4 > src > input.txt
You, 1 second ago | 1 author (You)
1 1 2 3 4 5 6 7 9 0
2 8

lab1 > task4 > src > output.txt
You, 1 second ago | 1 author (You)
1 -1
```

Вывод по задаче:

В этой задаче мы реализовали линейный поиск, а также различный вывод данных в зависимости от результата.

## Задание 5. Сортировка выбором

Необходимо написать код сортировки выбором (Selection-sort), которая выполняется следующим образом – сначала определяется наименьший элемент массива, который ставится на место элемента  $A[1]$ . Затем производится поиск второго наименьшего элемента массива  $A$ , который ставится на место элемента  $A[2]$ . Этот процесс продолжается для первых  $n - 1$  элементов массива  $A$ .

Листинг кода:

```
def selectionsort(input_name, output_name):
    input_file = open(input_name, 'r')
    output_file = open(output_name, 'w')

    n = int(input_file.readline())
    arr = list(map(int, input_file.readline().split()))

    for i in range(n):
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        (arr[i], arr[min_index]) = (arr[min_index], arr[i])

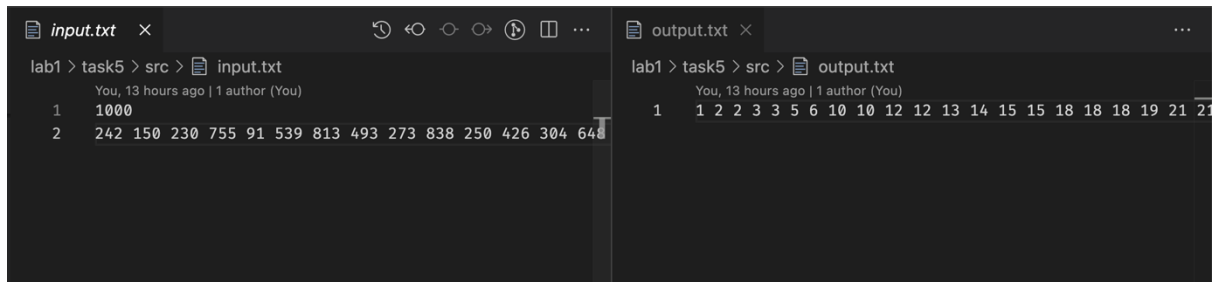
    output_file.write(' '.join(map(str, arr)))

    input_file.close()
    output_file.close()
```

Текстовое объяснение решения:

Сначала открываются входной и выходной файлы, считывается количество элементов и сам массив. Внешний цикл перебирает каждый элемент массива, начиная с первого, а внутренний цикл ищет наименьший элемент в оставшейся части массива. После нахождения минимального элемента, он меняется местами с текущим элементом. Таким образом, массив постепенно становится отсортированным. В конце отсортированный массив записывается в выходной файл.

## Результат работы кода:



```
input.txt
lab1 > task5 > src > input.txt
You, 13 hours ago | 1 author (You)
1 1000
2 242 150 230 755 91 539 813 493 273 838 250 426 304 648

output.txt
lab1 > task5 > src > output.txt
You, 13 hours ago | 1 author (You)
1 1 2 2 3 3 5 6 10 10 12 12 13 14 15 15 18 18 18 19 21 21
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00051 ms	0.01805 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.19209 ms	0.11826 MB

## Вывод по задаче:

В этой задаче мы узнали о новом методе сортировке Selection-sort, а затем успешно реализовали его. Такой подход обеспечивает стабильную работу для сортировки массива за фиксированное количество операций, что делает его подходящим для небольших наборов данных.

## Задание 10. Палиндром

В этой задаче нам необходимо составить палиндром максимальной длины из данного набора символов. Если палиндромов максимальной длины несколько, то нужно выбрать первый из них в алфавитном порядке.

Листинг кода:

```
def count_letters(string):
    letters = {}
    for letter in string:
        if letter in letters:
            letters[letter] += 1
        else:
            letters[letter] = 1
    return letters

def palindrome(input_name, output_name):
    input_file = open(input_name, 'r')
    output_file = open(output_name, 'w')

    input_file.readline()
    letters = count_letters(input_file.readline().replace("\n", ""))

    even_part = []
    odd_part = []

    for letter, freq in sorted(letters.items()):
        if freq % 2 == 0:
            even_part.append(letter * (freq // 2))
        else:
            even_part.append(letter * (freq // 2))
            odd_part.append(letter)

    left = ''.join(even_part)
    center = odd_part[0] if odd_part else ''
    right = left[::-1]

    output_file.write(left + center + right)

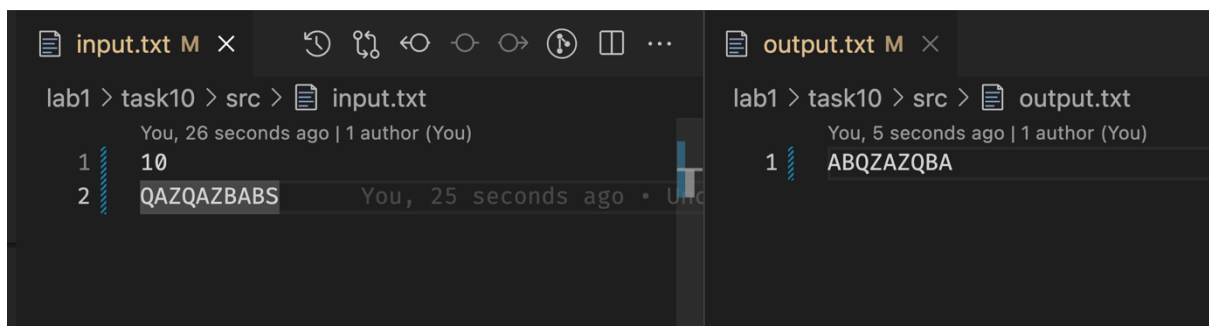
    input_file.close()
    output_file.close()
```

Текстовое объяснение решения:

Сначала открываются входной и выходной файлы, и считываются данные. Затем вызывается вспомогательная функция `count_letters`, которая

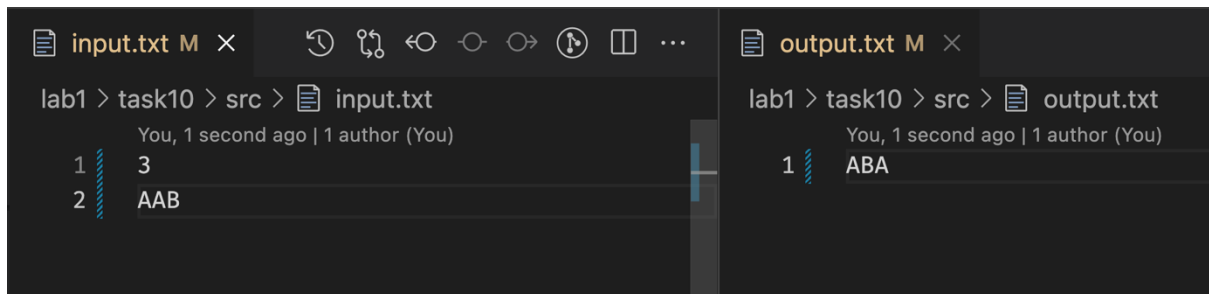
подсчитывает количество каждого символа в строке и возвращает словарь с частотой появления каждой буквы. После этого символы сортируются по алфавиту, и каждый символ с чётной частотой добавляется к левой и правой части будущего палиндрома, а символы с нечётной частотой сохраняются для центральной части. В результате формируются три части палиндрома: левая – из половин символов с чётной частотой, центральная – если есть символ с нечётной частотой, и правая – зеркальное отображение левой. Готовый палиндром записывается в выходной файл.

Результаты работы кода:



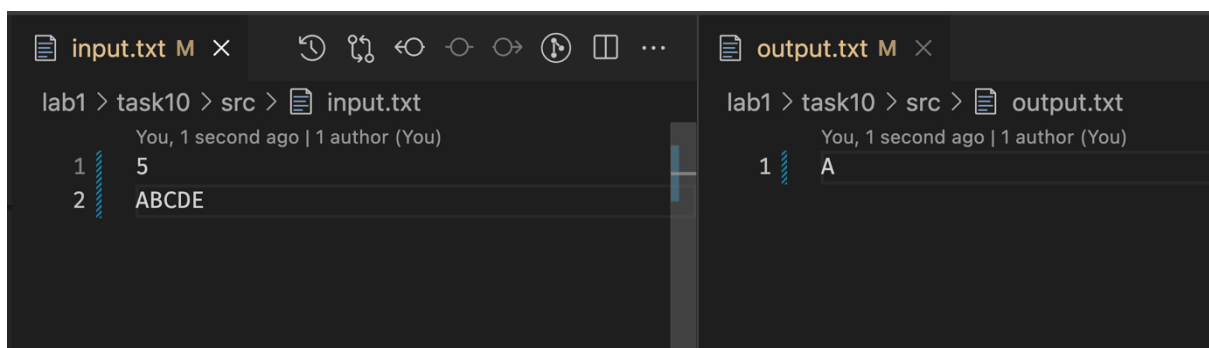
```
input.txt M x
lab1 > task10 > src > input.txt
You, 26 seconds ago | 1 author (You)
1 10
2 QAZQAZBABS You, 25 seconds ago | 1 author (You)

output.txt M x
lab1 > task10 > src > output.txt
You, 5 seconds ago | 1 author (You)
1 ABQZAZQBA
```



```
input.txt M x
lab1 > task10 > src > input.txt
You, 1 second ago | 1 author (You)
1 3
2 AAB

output.txt M x
lab1 > task10 > src > output.txt
You, 1 second ago | 1 author (You)
1 ABA
```



```
input.txt M x
lab1 > task10 > src > input.txt
You, 1 second ago | 1 author (You)
1 5
2 ABCDE

output.txt M x
lab1 > task10 > src > output.txt
You, 1 second ago | 1 author (You)
1 A
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00025 ms	0.01805 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.02936 ms	0.34772 MB

Вывод по задаче:

Таким образом, мы успешно написали код для алгоритма для составления палиндрома из набора символов.

## **Вывод**

В ходе данной лабораторной работы были изучены и реализованы различные алгоритмы сортировки и поиска, такие как сортировка вставками, выбором, сортировка вставками в обратном порядке, а также линейный поиск. В процессе работы также был изучен алгоритм составления палиндрома максимальной длины из набора символов. Каждое задание позволило глубже понять принципы работы алгоритмов сортировки и поиска, а также их применение на практике.