Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
(Университет ИТМО)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
по курсу «Алгоритмы и структуры данных»

Тема: СОРТИРОВКА ВСТАВКАМИ, ВЫБОРОМ, ПУЗЫРЬКОМ
Вариант 1

Санкт-Петербург, 2024

СОДЕРЖАНИЕ

ЗАДАЧИ ПО ВАРИАНТУ

1 Сортировка вставкой

Задание

1 задача. Сортировка вставкой

Используя код процедуры Insertion-sort, напишите программу и проверьте сортировку массива $A = \{31, 41, 59, 26, 41, 58\}$.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^3$) — 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. 2сек.
- Ограничение по памяти. 256 мб.

Выберите любой набор данных, подходящих по формату, и протестируйте алгоритм.

Код

InsertationSort.py

```
def insertion_sort_bin_pow(lst: list) -> None:
        for i in range(1, len(lst)):
                key = lst[i]
                left = 0
                right = i
                while left < right:
                        mid = left + (right - left) // 2
                        if key < lst[mid]:
                                right = mid
                        else:
                                left = mid + 1

                for j in range(i, right, -1):
                        lst[j] = lst[j - 1]

                lst[right] = key

def insertion_sort(lst: list) -> None:
        for i in range(1, len(lst)):
                cur = i
                while cur > 0 and lst[cur - 1] > lst[cur]:
                        lst[cur - 1], lst[cur] = lst[cur], lst[cur - 1]
                        cur -= 1
```

main.py

```
from InsertionSort import insertion_sort

def main():
    with open('../txtf/input.txt') as file:
```

```
        lst = list(map(int, file.readline().split()))

    insertion_sort(lst)

    with open('../txtf/output.txt', 'w') as file:
        print(*lst, file=file)

if __name__ == "__main__":
    main()
```

Анализ кода

Код представляет собой две реализации алгоритма сортировки вставкой: *insertion_sort_bin_pow* и *insertion_sort*. Функции принимает список *lst* в качестве входного параметра и сортирует его на месте.

Первая функция итерирует по списку от начала до конца, для каждого элемента ищет его позицию в отсортированной части списка с помощью бинарного поиска, затем сдвигает элементы в отсортированной части списка, чтобы освободить место для вставки элемента.

Вторая функция итерирует по списку от начала до конца, для каждого элемента сравнивает его с предыдущим элементом, и меняет их местами, если предыдущий элемент больше. Так происходит до тех пор, пока предыдущий элемент не окажется меньше.

insertion_sort в лучшем случае, когда список практически отсортирован, будет работать за O(n), а в худшем случае – за O(n^2).

insertion_sort_bin_pow независимо от исходных данных выполняет бинарный поиск за O(log_n) и сдвигает элементы, и время работы всегда будет около O(n*(log_n + n)), что чуть быстрее, чем сортировка при помощи предыдущей реализации, но время работы все равно O(n^2).

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-2\tests>cd ..\..\Task-1\tests

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-1\tests>type ..\txtf\input.txt
31 41 59 26 41 58
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-1\tests>python Time_Memory.py
Excecution time = 0.0011163000017404556 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-1\tests>type ..\txtf\output.txt
26 31 41 41 58 59
```

Вывод

В целом, *insertion_sort_bin_pow* является более эффективной для больших списков, а *insertion_sort* - для маленьких или почти отсортированных списков.

2 Сортировка вставкой с запоминанием индексов

Задание

2 задача. Сортировка вставкой +

Измените процедуру Insertion-sort для сортировки таким образом, чтобы в выходном файле отображалось в первой строке n чисел, которые обозначают новый индекс элемента массива после обработки.

- **Формат выходного файла (input.txt).** В первой строке выходного файла выведите n чисел. При этом i-ое число равно индексу, на который, в момент обработки его сортировкой вставками, был перемещен i-ый элемент исходного массива. Индексы нумеруются, начиная с единицы. Между любыми двумя числами должен стоять ровно один пробел.

Код

InsertationSort.py

```
def insertion_sort_indexes_bin_pow(lst: list) -> list:
        output_lst = [1]

        for i in range(1, len(lst)):
                key = lst[i]
                left = 0
                right = i
                while left < right:
                        mid = left + (right - left) // 2
                        if key < lst[mid]:
                                right = mid
                        else:
                                left = mid + 1

                for j in range(i, right, -1):
                        lst[j] = lst[j - 1]

                lst[right] = key
                output_lst.append(right + 1)

        return output_lst

def insertion_sort_indexes(lst: list) -> list:
        output_lst = [1]

        for i in range(1, len(lst)):
                cur = i
                while cur > 0 and lst[cur - 1] > lst[cur]:
                        lst[cur - 1], lst[cur] = lst[cur], lst[cur - 1]
                        cur -= 1
                output_lst.append(cur + 1)

        return output_lst
```

main.py

```
from InsertionSort import insertion_sort_indexes

def main():
    with open('../txtf/input.txt') as file:
        lst = list(map(int, file.readline().split()))
```

```
    with open('../txtf/output.txt', 'w') as file:
        print(*insertion_sort_indexes(lst), file=file)


if __name__ == "__main__":
    main()
```

Анализ кода

Код аналогичен коду из задачи 1, различие состоит в том, что параллельно сортировке, выполняется запись индексов в список, который потом возвращается функцией.

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-1\tests>cd ..\..\Task-2\tests

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-2\tests>type ..\txtf\input.txt
1 8 4 2 3 7 5 6 9 0
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-2\tests>python Time_Memory.py
Excecution time = 0.000971200002823025 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-2\tests>type ..\txtf\output.txt
1 2 2 2 3 5 5 6 9 1
```

3 Сортировка вставкой по убыванию

Задание

3 задача. Сортировка вставкой по убыванию

Перепишите процедуру Insertion-sort для сортировки в невозрастающем порядке вместо неубывающего с использованием процедуры Swap.

Формат входного и выходного файла и ограничения - как в задаче 1.

Подумайте, можно ли переписать алгоритм сортировки вставкой с использованием рекурсии?

Код

InsertationSort.py

```
def insertion_sort_reversed_bin_pow(lst: list) -> None:
        for i in range(1, len(lst)):
                key = lst[i]
                left = 0
                right = i
                while left < right:
                        mid = left + (right - left) // 2
                        if key > lst[mid]:
                                right = mid
                        else:
                                left = mid + 1

                for j in range(i, right, -1):
                        lst[j] = lst[j - 1]

                lst[right] = key

def insertion_sort(lst: list) -> None:
        for i in range(1, len(lst)):
                cur = i
                while cur > 0 and lst[cur - 1] < lst[cur]:
                        lst[cur - 1], lst[cur] = lst[cur], lst[cur - 1]
                        cur -= 1
```

main.py

```
from InsertionSort import insertion_sort_reversed

def main():
    with open('../txtf/input.txt') as file:
        lst = list(map(int, file.readline().split()))

    insertion_sort_reversed(lst)

    with open('../txtf/output.txt', 'w') as file:
        print(*lst, file=file)

if __name__ == "__main__":
    main()
```

Анализ кода

Код аналогичен коду из задачи 1, различие состоит в том, что в первой реализации бинарный поиск идет по отсортированному в обратном порядке массиву, поэтому поменяно условие *if key < lst[mid]:* на *if key > lst[mid]:*

Во второй реализации мы меняем элементы не тогда, когда предыдущий больше текущего, а наоборот.

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-3\tests>type ..\txtf\input.txt
31 41 59 26 41 58
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-3\tests>python Time_Memory.py
Excecution time = 0.0007708000048296526 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-3\tests>type ..\txtf\output.txt
59 58 41 41 31 26
```

Вывод

Чтобы сделать сортировку в обратном порядке, достаточно поменять в нужном месте знак > на знак <. Очень удобно.

4 Линейный поиск

Задание

4 задача. Линейный поиск

Рассмотрим задачу поиска.

- **Формат входного файла**. Последовательность из n чисел $A = a_1, a_2, \ldots, a_n$ в первой строке, числа разделены пробелом, и значение V во второй строке. Ограничения: $0 \leq n \leq 10^3$, $-10^3 \leq a_i, V \leq 10^3$
- **Формат выходного файла**. Одно число - индекс i, такой, что $V = A[i]$, или значение -1, если V в отсутствует.
- Напишите код линейного поиска, при работе которого выполняется сканирование последовательности в поисках значения V.
- Если число встречается несколько раз, то выведите, сколько раз встречается число и все индексы i через запятую.
- Дополнительно: попробуйте найти свинью, как в лекции. Используйте во входном файле последовательность слов из лекции, и найдите соответствующий индекс.

Код

Find.py

```
def find(lst: list, value) -> list:
    indexes = []
    for i in range(len(lst)):
        if lst[i] == value:
            indexes.append(i + 1)

    if not indexes:
        indexes.append(-1)
    return indexes
```

main.py

```
from Find import find

def main():
    with open('../txtf/input.txt') as file:
        lst = list(map(int, file.readline().split()))
        value = int(file.readline())

    with open('../txtf/output.txt', 'w') as file:
        print(*find(lst, value), file=file)


if __name__ == "__main__":
    main()
```

Анализ кода

Идем по списку, если элемент списка совпадает с искомым значением, запоминаем его индекс. Если список с индексами окажется в итоге пустым, добавим в него -1, чтобы было понятно, что ничего не нашлось.

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-3\tests>cd ..\..\Task-4\tests

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-4\tests>type ..\txtf\input.txt
31 41 59 26 41 58
41
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-4\tests>type ..\txtf\output.txt

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-4\tests>python Time_Memory.py
Excecution time = 0.0008033000003706547 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-4\tests>type ..\txtf\output.txt
2 5
```

Вывод

Линейный, последовательный поиск — алгоритм нахождения заданного значения произвольной функции на некотором отрезке. Данный алгоритм является простейшим алгоритмом поиска. Выполняется на O(n).

5 Сортировка выбором

Задание

5 задача. Сортировка выбором.

Рассмотрим сортировку элементов массива , 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.

Напишите код этого алгоритма, также известного как сортировка выбором (selection sort). Определите время сортировки выбором в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой.

Формат входного и выходного файла и ограничения - как в задаче 1.

Код

SelectionSort.py

```python
def selection_sort(lst: list) -> None:
    for start in range(len(lst) - 1):
        mn = [lst[start], start]
        for i in range(start + 1, len(lst)):
            if lst[i] < mn[0]:
                *mn, = lst[i], i
        lst[start], lst[mn[1]] = lst[mn[1]], lst[start]
```

main.py

```python
from SelectionSort import selection_sort
```

```python
def main():
    with open('../txtf/input.txt') as file:
        lst = list(map(int, file.readline().split()))

    selection_sort(lst)

    with open('../txtf/output.txt', 'w') as file:
        print(*lst, file=file)

if __name__ == "__main__":
    main()
```

Анализ кода

Функция *selection_sort* принимает список *lst* в качестве входного параметра и сортирует его на месте.

Функция использует два цикла для итерации по списку. Внешний цикл итерирует по списку от начала до конца, а внутренний цикл итерирует от текущей позиции внешнего цикла до конца списка в поисках минимального элемента в неотсортированной части. Найденный элемент меняется с элементов в текущей позиции, т.е. становится последним в отсортированной части.

Время работы такой сортировки в любом случае будет O(n), тес не менее, можно оптимизировать поиск минимального элемента, например, храня исходные данные в куче (это позволяет искать минимум за O(log_n), т.е. общее время работы алгоритма улучшится до O(n log_n))

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-4\tests>cd ..\..\Task-5\tests

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-5\tests>type ..\txtf\input.txt
31 41 59 26 41 58
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-5\tests>type ..\txtf\output.txt

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-5\tests>python Time_Memory.py
Excecution time = 0.0008913000001484761 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-5\tests>type ..\txtf\output.txt
26 31 41 41 58 59
```

Вывод

Сортировка вставкой пока выглядит более презентабельно, т.к. она хотя бы для почти отсортированных списков хорошо работает. К тому же сортировку выбором сложнее оптимизировать и есть более легкие в написании алгоритмы, время выполнения которых лучше оптимизированной сортировки выбором.

6 Сортировка пузырьком

Задание

Пузырьковая сортировка представляет собой популярный, но не очень эффективный алгоритм сортировки. В его основе лежит многократная перестановка соседних элементов, нарушающих порядок сортировки. Вот псевдокод этой сортировки:

```
Bubble_Sort(A):
  for i = 1 to A.length - 1
    for j = A.length downto i+1
      if A[j] < A[j-1]
        поменять A[j] и A[j-1] местами
```

Напишите код на Python и докажите корректность пузырьковой сортировки. Для доказательства корректоности процедуры вам необходимо доказать, что она завершается и что $A'[1] \leq A'[2] \leq ... \leq A'[n]$, где A' - выход процедуры Bubble_Sort, а n - длина массива A.

Определите время пузырьковой сортировки в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой.

Формат входного и выходного файла и ограничения - как в задаче 1.

Код

BubbleSort.py

```python
def bubble_sort(lst: list) -> None:
    for left in range(len(lst) - 1):
        flag = False
        for i in range(len(lst) - 1, left, -1):
            if lst[i - 1] > lst[i]:
                lst[i - 1], lst[i] = lst[i], lst[i - 1]
                flag = True

        if not flag: break
```

main.py

```python
from BubbleSort import bubble_sort

def main():
    with open('../txtf/input.txt') as file:
        lst = list(map(int, file.readline().split()))

    bubble_sort(lst)

    with open('../txtf/output.txt', 'w') as file:
        print(*lst, file=file)

if __name__ == "__main__":
    main()
```

Анализ кода

Функция *bubble_sort* принимает список *lst* в качестве входного параметра и сортирует его на месте.

Внешний цикл итерирует по списку от начала до конца. Во внутреннем цикле, начиная с конца списка и заканчивая первой парой после уже отсортированной части, сравнивается каждый элемент с его соседом справа и, если элемент больше, чем его сосед, обменивает их местами. Таким образом, на каждой итерации внешнего цикла неотсортированная часть списка изменяется таким образом, что минимальный элемент этой части оказывается первым.
Функция оптимизирована с помощью флага: если при выполнении внутреннего цикла не поменялись никакие два соседних элемента, значит, неотсортированная часть списка уже отсортирована и можно закончить внешний цикл.
Благодаря оптимизации в лучшем случае алгоритм отработает за O(n), но в худшем время работы остается O(n^2). В этом плане эта сортировка похожа на сортировку вставкой.

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>cd ..\..\Task-6\tests

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>type ..\txtf\input.txt
9 8 7 6 5 4 3 2 1
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>type ..\txtf\output.txt

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>python Time_Memory.py
Excecution time = 0.0007253999992826721 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>type ..\txtf\output.txt
1 2 3 4 5 6 7 8 9
```

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>type ..\txtf\input.txt
1 2 3 4 5 6 7 8 9
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>type ..\txtf\output.txt

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>python Time_Memory.py
Excecution time = 0.0006395999998858315 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>type ..\txtf\output.txt
1 2 3 4 5 6 7 8 9
```

Запишем в файл input.txt последовательность от 0 до 1000 и посмотрим время выполнения программы

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>for /l %i in (0,1,1000) do <nul set /p "= %i " >> ..\txtf\input.txt
```

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>python Time_Memory.py
Excecution time = 0.00925680000000284 seconds
Memory used: 0.001 MB; Memory peak: 0.142 MB
```

А теперь запишем туда последовательность от 1000 до 0

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>for /l %i in (1000,-1,0) do <nul set /p "= %i " >> ..\txtf\input.txt
```

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-6\tests>python Time_Memory.py
Excecution time = 0.6536955000001399 seconds
Memory used: 0.001 MB; Memory peak: 0.142 MB
```

Время в наилучшем и наихудшем случае сильно отличается.

Вывод

7 Сортировка пузырьком с индексами

Задание

7 задача. Знакомство с жителями Сортлэнда

Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со своими подданными. Число жителей в графстве нечетно и составляет n, где n может быть достаточно велико, поэтому граф решил ограничиться знакомством с тремя представителями народонаселения: с самым бедным жителем, с жителем, обладающим средним достатком, и с самым богатым жителем.

Согласно традициям Сортлэнда, считается, что житель обладает средним достатком, если при сортировке жителей по сумме денежных сбережений он оказывается ровно посередине. Известно, что каждый житель графства имеет уникальный идентификационный номер, значение которого расположено в границах от единицы до n. Информация о размере денежных накоплений жителей хранится в массиве M таким образом, что сумма денежных накоплений жителя, обладающего идентификационным номером i, содержится в ячейке $M[i]$. Помогите секретарю графа мистеру Свопу вычислить идентификационные номера жителей, которые будут приглашены на встречу с графом.

- **Формат входного файла (input.txt).** Первая строка входного файла содержит число жителей n ($3 \le n \le 9999$, n нечетно). Вторая строка содержит описание массива M, состоящее из положительных вещественных чисел, разделенных пробелами. Гарантируется, что все элементы массива M различны, а их значения имеют точность не более двух знаков после запятой и не превышают 10^6.

- **Формат выходного файла (output.txt).** В выходной файл выведите три целых положительных числа, разделенных пробелами — идентификационные номера беднейшего, среднего и самого богатого жителей Сортлэнда.

Код

BubbleSort.py

```python
def bubble_sort_indexes(lst: list) -> list:
    indexes = list(range(1, len(lst) + 1))

    for left in range(len(lst) - 1):
        flag = False
        for i in range(len(lst) - 1, left, -1):
            if lst[i - 1] > lst[i]:
                lst[i - 1], lst[i] = lst[i], lst[i - 1]
                indexes[i - 1], indexes[i] = indexes[i], indexes[i - 1]
                flag = True
```

```
        if not flag: break

    return indexes
```

main.py

```
from BubbleSort import bubble_sort_indexes
from decimal import Decimal

def main():
    with open('../txtf/input.txt') as file:
        lst = list(map(int, file.readline().split()))
        if len(lst) % 2 == 0:
            raise ValueError("The list must contain an odd number of
elements")

    indexes = bubble_sort_indexes(lst)
    print(*lst)

    with open('../txtf/output.txt', 'w') as file:
        print(indexes[0],  indexes[len(indexes) // 2], indexes[-1], sep=" ",
file=file)

if __name__ == "__main__":
    main()
```

Анализ кода

Код аналогичен коду из задачи 6, различие состоит в том, что параллельно сортировке, выполняется запись индексов в список, который потом возвращается функцией.

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-7\tests>type ..\txtf\input.txt
31 41 59 26 41 58 3
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-7\tests>type ..\txtf\output.txt

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-7\tests>python Time_Memory.py
Excecution time = 0.0007443000013154233 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-7\tests>type ..\txtf\output.txt
7 2 3
```

8 Сортировка выбором +

Задание

8 задача. Секретарь Своп

Дан массив, состоящий из n целых чисел. Вам необходимо его отсортировать по неубыванию. Но делать это нужно так же, как это делает мистер Своп — то есть, каждое действие должно быть взаимной перестановкой пары элементов. Вам также придется записать все, что Вы делали, в файл, чтобы мистер Своп смог проверить Вашу работу.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($3 \leq n \leq 5000$) — число элементов в массиве. Во второй строке находятся n целых чисел, по модулю не превосходящих 10^9. Числа могут совпадать друг с другом.
- **Формат выходного файла (output.txt).** В первых нескольких строках выведите осуществленные Вами операции перестановки элементов. Каждая строка должна иметь следующий формат:

  `Swap elements at indices X and Y.`

  Здесь X и Y — различные индексы массива, элементы на которых нужно переставить ($1 \leq X, Y \leq n$). Мистер Своп любит порядок, поэтому сделайте так, чтобы $X < Y$.

  После того, как все нужные перестановки выведены, выведите следующую фразу:

  `No more swaps needed.`

Код

SelectionSort.py

```python
def selection_sort(lst: list):
    for start in range(len(lst) - 1):
        mn = [lst[start], start]
        for i in range(start + 1, len(lst)):
            if lst[i] < mn[0]:
                *mn, = lst[i], i
        if start != mn[1]:
            lst[start], lst[mn[1]] = lst[mn[1]], lst[start]
            yield f"Swap elements at indices {start + 1} and {mn[1] + 1}."

    yield "No more swaps needed."
```

main.py

```python
from SelectionSort import selection_sort

def main():
    with open('../txtf/input.txt') as file:
        lst = list(map(int, file.readline().split()))

    with open('../txtf/output.txt', 'w') as file:
        print(*selection_sort(lst), sep="\n", file=file)

if __name__ == "__main__":
    main()
```

Анализ кода

Код аналогичен коду из задачи 5, различие состоит в том, что параллельно сортировке, выполняется генерация сообщений с информацией о том, на какой позиции был найден минимум. Также генерируется сообщение о том, что сортировка завершена.

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-7\tests>cd ..\..\Task-8\tests

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-8\tests>type ..\txtf\input.txt
3 1 4 2 2
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-8\tests>type ..\txtf\output.txt

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-8\tests>python Time_Memory.py
Excecution time = 0.0006306999985099537 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-8\tests>type ..\txtf\output.txt
Swap elements at indices 1 and 2.
Swap elements at indices 2 and 4.
Swap elements at indices 3 and 5.
No more swaps needed.
```

9 Сложение бинарных чисел

Задание

9 задача. Сложение двоичных чисел

Расмотрим задачу сложения двух n-битовых двоичных целых чисел, хранящихся в n-элементных массивах A и B. Сумму этих двух чисел необходимо занести в двоичной форме в $(n+1)$-элементный массив C. Напишите скрипт для сложения этих двух чисел.

- **Формат входного файла (input.txt).** В одной строке содержится два n-битовых двоичных числа, записанные через пробел ($1 \leq n \leq 10^3$)
- **Формат выходного файла (output.txt).** Одна строка - двоичное число, которое является суммой двух чисел из входного файла.
- Оцените асимптотическое время выполнение вашего алгоритма.

Код

BinAdd.py

```
def bin_add(first: list, second: list) -> list:
        if len(first) < len(second):
                first = [0] * (len(second) - len(first)) + first
        elif len(first) > len(second):
                second = [0] * (len(first) - len(second)) + second

        n = len(first)
        result = [0] * (n + 1)
        for i in range(n - 1, -1, -1):
                if not(first[i] == 0 or first[i] == 1) or not(second[i] == 0
                                                              or second[i] == 1):
                        raise ValueError("Incorrect number")

                result[i + 1] += first[i] + second[i]
                if result[i + 1] > 1:
                        result[i] += 1
                        result[i + 1] -= 2

        return result
```

main.py

```
from BinAdd import bin_add

def main():
    with open('../txtf/input.txt') as file:
        first, second = map(list, file.readline().split())
        first = list(map(int, first))
        second = list(map(int, second))

    with open('../txtf/output.txt', 'w') as file:
        print(*bin_add(first, second), sep="", file=file)

if __name__ == "__main__":
    main()
```

Анализ кода

Функция *bin_add* принимает два списка *first* и *second* в качестве входных параметров и возвращает результат сложения в виде списка.

Сначала сравниваются длины входных списков и дополняет короче список нулями до длины более длинного списка.

Затем создается список *result* длиной на 1 больше длины входных списков для хранения результата.

Функция также проверяет, что элементы входных списков являются бинарными (0 или 1), и если это не так, генерирует исключение *ValueError*.

Функция итерирует по спискам в обратном порядке, начиная с конца, и для каждого элемента суммирует соответствующие элементы из обоих списков.

Если сумма элементов превышает 1, функция добавляет 1 к предыдущему элементу результата и вычитает 2 из текущего элемента результата.

Достаточно просто вычитать 2, т.к. результат сложения не превышает 3 (максимум 2 после сложения соотв. разрядов, максимум 1 от переноса единицы после сложения предыдущих разрядов).

Алгоритм работает за O(n).

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-9\tests>type ..\txtf\input.txt
101001110111 1010101111100000
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-9\tests>type ..\txtf\output.txt

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-9\tests>python Time_Memory.py
Excecution time = 0.01550199999473989 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-9\tests>type ..\txtf\output.txt
01011011001010111
```

Вывод

В целом, функция *bin_add* является эффективным алгоритмом для сложения двоичных чисел, представленных в виде списков.

10 Построение палиндромов

Задание

10 задача★. Палиндром

Палиндром - это строка, которая читается одинаково как справа налево, так и слева направо.

На вход программы поступает набор больших латинских букв (не обязательно различных). Разрешается переставлять буквы, а также удалять некоторые буквы. Требуется из данных букв по указанным правилам составить палиндром наибольшей длины, а если таких палиндромов несколько, то выбрать первый из них в алфавитном порядке.

- **Формат входного файла (input.txt).** В первой строке входных данных содержится число n ($1 \leq n \leq 100000$). Во второй строке задается последовательность из n больших латинских букв (буквы записаны без пробелов).
- **Формат выходного файла (output.txt).** В единственной строке выходных данных выдайте искомый палиндром.
- Пример:

Код

MakePalindrome.py

```
from string import ascii_uppercase

def make_palindrome(letters: str) -> str:
    letters_cnt = [0] * len(ascii_uppercase)

    for sym in letters:
        if sym not in ascii_uppercase:
            raise ValueError("Unknown symbol")

        letters_cnt[ord(sym) - ord("A")] += 1

    center = str()
    palindrome_half = []
    for i in range(len(letters_cnt)):
        if letters_cnt[i] % 2 == 1 and center == "":
            center = chr(ord("A") + i)

        palindrome_half.extend([chr(ord("A") + i)] * (letters_cnt[i] // 2))

    return "".join(palindrome_half + [center] +
list(reversed(palindrome_half)))
```

main.py

```
from MakePalindrome import make_palindrome

def main():
    with open('../txtf/input.txt') as file:
        letters = file.readline()

    with open('../txtf/output.txt', 'w') as file:
        print(make_palindrome(letters), file=file)
```

```
if __name__ == "__main__":
    main()
```

Анализ кода

Функция *make_palindrome* принимает строку *letters* в качестве входного параметра и возвращает палиндром в виде строки.

Функция создает список *letters_cnt* длиной 26, где каждый элемент представляет собой количество вхождений соответствующей буквы в строке *letters*. Заполняя его, она итерирует по строке *letters* и для каждого символа увеличивает соответствующий элемент в списке *letters_cnt*. Если символ не является буквой латинского алфавита, генерируется исключение *ValueError*.

Функция затем итерирует по списку *letters_cnt* и для каждой буквы, которая встречается нечетное количество раз, добавляет ее в центр палиндрома.

Для каждой буквы, которая встречается четное количество раз, функция добавляет половину этого количества в начало и конец палиндрома. Первая буква, которая встречается нечетное количество раз, становится центральной.

Функция возвращает палиндром в виде строки.

```
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-10\tests>type ..\txtf\input.txt
AAAABBBCCC
C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-10\tests>type ..\txtf\output.txt

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-10\tests>python Time_Memory.py
Excecution time = 0.0022944999946048483 seconds
Memory used: 0.001 MB; Memory peak: 0.018 MB

C:\Users\Admin\Desktop\ITMO\Algorithms\Lab-1\Task-10\tests>type ..\txtf\output.txt
AABCBCBAA
```