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.
Вариант 11

Выполнил:
Лютый Никита Артемович
К3140

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задача №1. Сортировка вставкой

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.

Листинг кода

```
def insertion_sort(n, lst):
    for i in range(1,n):
        elem = lst[i]
        j=i-1
        while j>=0 and lst[j]>elem:
            lst[j+1]=lst[j]
            j-=1
        lst[j+1]=elem

    return lst

def main():
    file_inp = open("input.txt")
    n = int(file_inp.readline())
    lst = list(map(int, file_inp.readline().split()))
    file_inp.close()

    if len(lst)!= n:
        quit("Указанная длина списка не соответствует действительной!")

    if n>1000 or n<1 or max(lst)>10**9:
        quit("Входные данные не соответствуют условию задачи!")

    lst = list(map(str, insertion_sort(n, lst)))
    out = " ".join(lst)

    file_out = open("output.txt", 'w')
    file_out.write(out)
    file_out.close()

main()
```

Тесты:

1) Тест на время работы:

```
from lab1.task1.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))
```

2) Тест на занимаемую память:

```
from lab1.task1.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

Текстовое объяснение решения:
Реализовано 2 функции:

1) main():

Открывает файл input.txt и считывает из него n – количество элементов и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается) и перезаписывает переменную lst, положив туда список, отсортированный функцией insertion_sort(), каждый элемент которого переведен в строку. С помощью метода join() формирует выходное сообщение, которое далее записывает в файл output.txt.

2) insertion_sort():

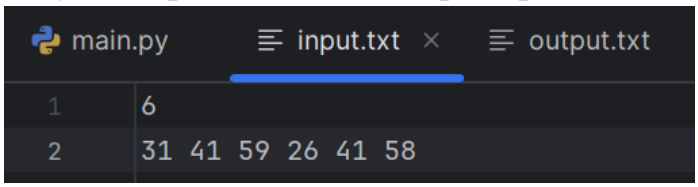
Получает на вход длину списка и сам список. Циклом for начинает бежать по нему от 1 до n-го индекса. В переменную elem записываем текущий элемент и создаем j, содержащую индекс предыдущего элемента. Запускаем цикл while, который в случае, если предыдущий элемент больше текущего, сдвинет больший вперед. В конце сдвигов элементу, который был сдвинут последним, выдаем сохраненное в elem значение. И так продолжаем до конца списка, после чего возвращаем его.

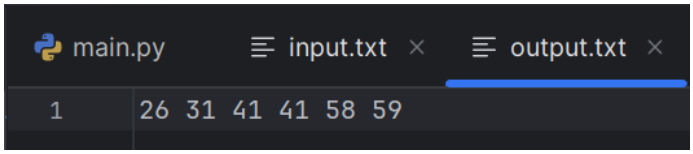
Тесты:

1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.

2) Импортируем нашу функцию main() и с помощью встроенной библиотеки tracemalloc замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

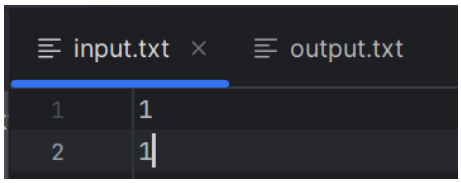
Результат работы кода на примерах из текста задачи:

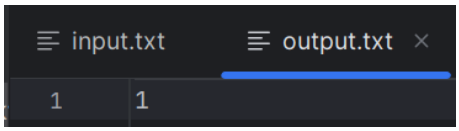




Результат работы кода на максимальных и минимальных значениях:

1) Минимальные значения:

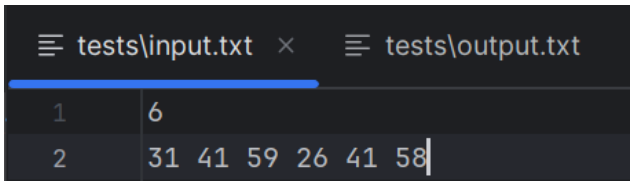


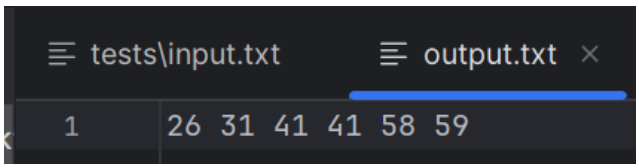



```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task1\tests\test_memory_main.py
Максимально занимаемая память: 17.4091796875 KB

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task1\tests\test_time_main.py
Время работы программы 0.0007081000076141208 секунд.

Process finished with exit code 0
```

2) Значения из примера:

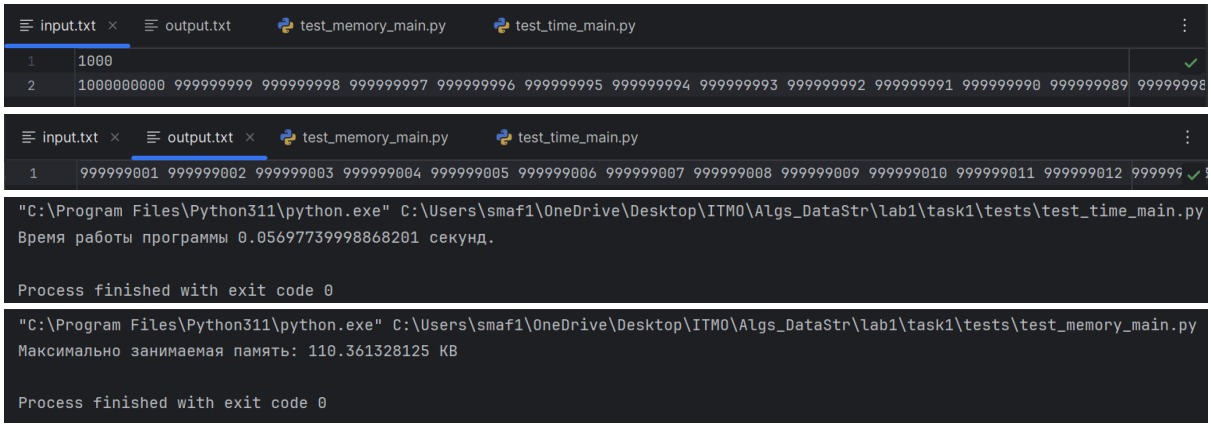





```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task1\tests\test_time_main.py
Время работы программы 0.0007864999934099615 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task1\tests\test_memory_main.py
Максимально занимаемая память: 17.4892578125 KB

Process finished with exit code 0
```

3) Максимальные значения:



	Время выполнения, с	Затраты памяти, KB
Нижняя граница диапазона значений входных данных из текста задачи	0.000708	17.40917
Пример из задачи	0.000786	17.48926
Верхняя граница диапазона значений входных данных из текста задачи	0.056977	110.36133

Вывод по задаче:
Сортировка вставкой является довольно затратной сортировкой по времени и памяти.

Задача №3. Сортировка вставкой по убыванию

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?

Листинг кода. (именно листинг, а не скрины)

```
def insertion_sort(n, lst):
    for i in range(1,n):
        j=i-1
        ind = i
        while j>=0 and lst[j]<lst[ind] and ind>=0:
            lst[ind], lst[j] = lst[j], lst[ind]
            j-=1
            ind-=1

    return lst

def main():
    file_inp = open("input.txt")
    n = int(file_inp.readline())
    lst = list(map(int, file_inp.readline().split()))
    file_inp.close()

    if len(lst)!= n:
        quit("Указанная длина списка не соответствует действительной!")

    if n>1000 or n<1 or max(lst)>10**9:
        quit("Входные данные не соответствуют условию задачи!")

    lst = list(map(str, insertion_sort(n, lst)))
    out = " ".join(lst)

    file_out = open("output.txt", 'w')
    file_out.write(out)
    file_out.close()

main()
```

Тесты:

1) Тест на время работы

```
from lab1.task3.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))
```

2) Тест на занимаемую память

```
from lab1.task3.src.main import main
import tracemalloc
```

```
tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

Текстовое объяснение решения:

Реализовано 2 функции:

1) main():

Открывает файл input.txt и считывает из него n – количество элементов и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается) и перезаписывает переменную lst, положив туда список, отсортированный функцией insertion_sort(), каждый элемент которого переведен в строку. С помощью метода join() формирует выходное сообщение, которое далее записывает в файл output.txt.

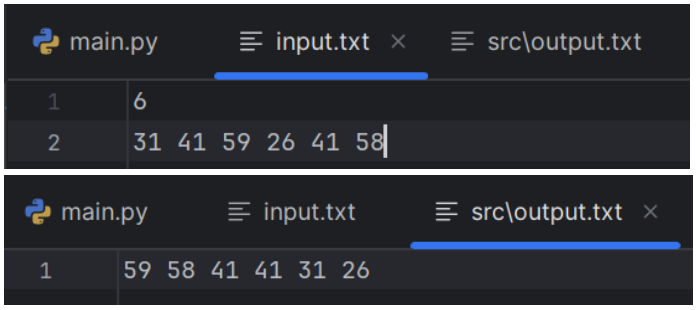
2) insertion_sort():

Получает на вход длину списка и сам список. Циклом for начинает бежать по нему от 1 до n-го индекса. В переменную ind запоминаем индекс текущего элемента, а в переменную j – предыдущего. В цикле while мы меняем местами 2 выбранных элемента списка с помощью swap, пока текущий элемент больше предыдущего, после чего сдвигаем ind и j на 1 влево. И так продолжаем до конца списка, после чего возвращаем его.

Тесты:

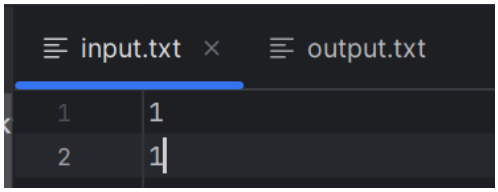
1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.
2) Импортируем нашу функцию main() и с помощью встроенной библиотеки tracemalloc замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

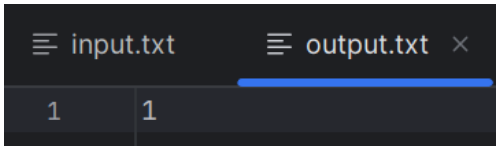
Результат работы кода на примерах из задачи:



Результат работы кода на максимальных и минимальных значениях:

1) Минимальные значения:

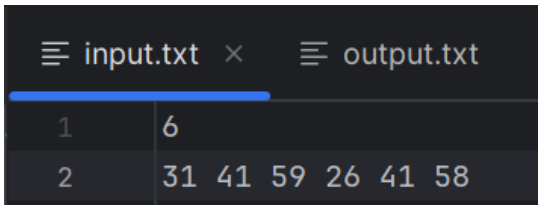


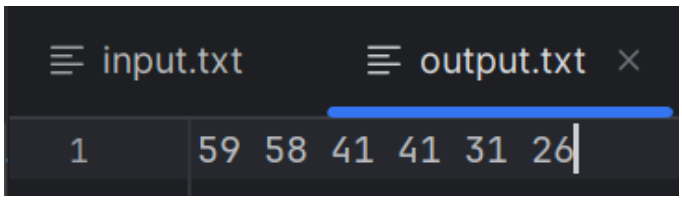



```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task3\tests\test_time_main.py
Время работы программы 0.0007976000051712617 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task3\tests\test_memory_main.py
Максимально занимаемая память: 17.4091796875 KB

Process finished with exit code 0
```

2) Значения из примера:

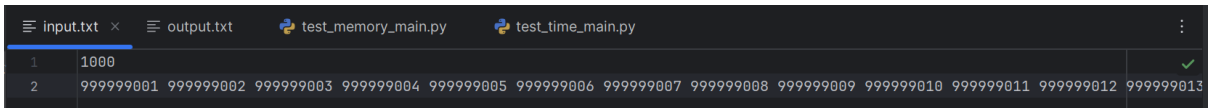





```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task3\tests\test_time_main.py
Время работы программы 0.0008308999967994168 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task3\tests\test_memory_main.py
Максимально занимаемая память: 17.4892578125 KB

Process finished with exit code 0
```

3) Максимальные значения:




```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task3\tests\test_time_main.py
Время работы программы 0.07559150000452064 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task3\tests\test_memory_main.py
Максимально занимаемая память: 110.361328125 KB

Process finished with exit code 0
```

	Время выполнения, с	Затраты памяти, KB

Нижняя граница диапазона значений входных данных из текста задачи	0.0007976	17.409
Пример из задачи	0.0008309	17.489
Верхняя граница диапазона значений входных данных из текста задачи	0.0755915	110.361

Вывод по задаче:
Сортировка вставкой является довольно затратной сортировкой по времени и памяти, а добавление регулярных swap’ов ее еще больше замедляет.

Задача №8. Секретарь Своп

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.
  ```

- Пример:

input.txt	output.txt
5 3 1 4 2 2	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.

- Ограничение по времени. 1 сек.
- Ограничение по памяти. 256 мб.

Семья секретаря Свопа занималась сортировками массивов, и именно с помощью перестановок пар элементов, как минимум с XII века, поэтому все Свопы владеют этим искусством в совершенстве. Мы не просим Вас произвести минимальную последовательность перестановок, приводящую к правильному ответу. Однако учтите, что для вывода слишком длинной последовательности у Вашего алгоритма может не хватить времени (или памяти — если выводимые строки хранятся в памяти перед выводом). Подумайте, что с этим можно сделать. Решение существует!

Листинг кода:

```
def parse_lst_oper(arr):
    idx_pair_to_str = lambda x: f'Swap elements at indices {x[0]} and {x[1]}'
    return "\n".join(map(idx_pair_to_str, arr)) + "\nNo more swap needed."


def list_sort(n, lst):
    lst_oper = []

    for i in range(n-1):
        cached_elem = (lst[i], i)

        for j in range(i+1, n):
            if cached_elem[0] > lst[j]:
                cached_elem = (lst[j], j)

        if cached_elem[1] != i:
            lst[i], lst[cached_elem[1]] = lst[cached_elem[1]], lst[i]
            lst_oper.append((i + 1, cached_elem[1] + 1))

    return parse_lst_oper(lst_oper)


def main():
    with open("input.txt") as file_inp:
        n = int(file_inp.readline().strip())
        lst = list(map(int, file_inp.readline().strip().split()))

    if len(lst) != n:
        quit("Указанная длина списка не соответствует действительной!")

    if n > 5000 or n < 3 or max(lst) > 10 ** 9:
        quit("Входные данные не соответствуют условию задачи!")

    with open("output.txt", 'w+') as file_out:
        file_out.write(list_sort(n, lst))


main()
```

Тесты:

1) Тест на время работы:

```
from lab1.task8.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))
```

2) Тест на занимаемую память:

```
from lab1.task8.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

Текстовое объяснение решения:
Реализовано 3 функции:

1) main():

Открывает файл input.txt и считывает из него n – количество элементов и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). Открывает для редактирования файл output.txt и записывает туда результат работы функции list_sort().

2) list_sort():

Получает на вход длину списка и сам список. Создает переменную lst_oper, которая будет содержать пары индексов элементов, которые мы меняем местами. Пробегаемся циклом for по индексам до n-1. В cached_elem мы записываем кортеж, где на 1 месте стоит текущий элемент, а на 2-ом – его индекс. Запускаем цикл for для j от i+1 до n. Мы проверяем, если текущий элемент больше следующего, то в cached_elem записываем наименьший из них и его индекс. После цикла мы проверяем, произошли ли изменения индекса. Если да, то меняем элементы с этими индексами местами и записываем кортеж из 2-х этих индексов в lst_oper. И так продолжаем до конца списка, после чего возвращаем результат работы функции parse_lst_oper() с данным списком кортежей с индексами.

3) parse_lst_oper():

Принимает на вход список с кортежами пар индексов. В переменную idx_pair_to_str мы записываем lambda-функций по преобразованию наших кортежей из индексов в строку, требуемую по условию задачи. В результате возвращаем элементы, превращенные в строки и разделенные «\n» (отступ для перехода на следующую строку в файле), и добавленную к этому строку об окончании операций.

Тесты:

1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.
2) Импортируем нашу функцию main() и с помощью встроенной библиотеки tracemalloc замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:

main.py input.txt output.txt

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

main.py input.txt 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 swap needed.
```

Результат работы кода на максимальных и минимальных значениях:

1) Минимальные значения:

input.txt output.txt

```
3
1 4 3
```

input.txt output.txt test_time_m

```
Swap elements at indices 2 and 3
No more swap needed.
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task8\tests\test_time_main.py
Время работы программы 0.0008011999889276922 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task8\tests\test_memory_main.py
Максимально занимаемая память: 17.5283203125 КБ

Process finished with exit code 0
```

2) Значения из примера:

input.txt output.txt

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

input.txt output.txt test_time_

```
Swap elements at indices 1 and 2
Swap elements at indices 2 and 4
Swap elements at indices 3 and 5
No more swap needed.
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task8\tests\test_time_main.py
Время работы программы 0.0009634000016376376 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task8\tests\test_memory_main.py
Максимально занимаемая память: 17.5361328125 KB

Process finished with exit code 0
```

3) Максимальные значения:

input.txt | output.txt | test_time_main.py | test_memory_main.py

```
5000
1000000000 999999999 999999998 999999997 999999996 999999995 999999994 999999993 999999992 999999991 999999990 999999989 99999998
```

input.txt | output.txt | test_time_main.py

```
Swap elements at indices 1 and 5000
Swap elements at indices 2 and 4999
Swap elements at indices 3 and 4998
Swap elements at indices 4 and 4997
Swap elements at indices 5 and 4996
Swap elements at indices 6 and 4995
Swap elements at indices 7 and 4994
Swap elements at indices 8 and 4993
Swap elements at indices 9 and 4992
Swap elements at indices 10 and 4991
Swap elements at indices 11 and 4990
Swap elements at indices 12 and 4989
Swap elements at indices 13 and 4988
Swap elements at indices 14 and 4987
Swap elements at indices 15 and 4986
Swap elements at indices 16 and 4985
Swap elements at indices 17 and 4984
Swap elements at indices 18 and 4983
Swap elements at indices 19 and 4982
Swap elements at indices 20 and 4981
Swap elements at indices 21 and 4980
Swap elements at indices 22 and 4979
Swap elements at indices 23 and 4978
Swap elements at indices 24 and 4977
Swap elements at indices 25 and 4976
Swap elements at indices 26 and 4975
Swap elements at indices 27 and 4974
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task8\tests\test_time_main.py
Время работы программы 0.8509703000017907 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task8\tests\test_memory_main.py
Максимально занимаемая память: 712.0869140625 KB

Process finished with exit code 0
```

	Время выполнения, с	Затраты памяти, КВ
Нижняя граница диапазона значений входных данных из текста задачи	0.0008	17.528
Пример из задачи	0.0009	17.536
Верхняя граница диапазона значений входных данных из текста задачи	0.8510	712.087

Вывод по задаче: реализованный алгоритм сортировки работает быстрее сортировки вставками, но по-прежнему довольно затратный по времени и памяти.

Задача №2. Сортировка вставкой +

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.

Пример.

input.txt	output.txt
10 1 8 4 2 3 7 5 6 9 0	1 2 2 2 3 5 5 6 9 1 0 1 2 3 4 5 6 7 8 9

В примере сортировка вставками работает следующим образом:

- Первый элемент остается на своем месте, поэтому первое число в ответе — единица. Отсортированная часть массива: [1]
- Второй элемент больше первого, поэтому он тоже остается на своем месте, и второе число в ответе — двойка. [1 8]
- Четверка меньше восьмерки, поэтому занимает второе место. [1 4 8]
- Двойка занимает второе место. [1 2 4 8]
- Тройка занимает третье место. [1 2 3 4 8]

Листинг кода

```
def insertion_sort(n, lst):
  lst_ind = [1 for i in range(n)]
  for i in range(1,n):
    elem = lst[i]
    j=i-1
    while j>=0 and lst[j]>elem:
      lst[j+1]=lst[j]
      j-=1
    lst[j+1]=elem

    if i==1:
      if j==0:
        lst_ind[0] = 1
        lst_ind[1] = 2
      else:
        lst_ind[1] = 1
        lst_ind[0] = 2
    else:
      lst_ind[i] = lst.index(elem)+1
```

```
    return (lst, lst_ind)

def main():
    file_inp = open("input.txt")
    n = int(file_inp.readline())
    lst = list(map(int, file_inp.readline().split()))
    file_inp.close()

    if len(lst)!= n:
        quit("Указанная длина списка не соответствует действительной!")

    if n>1000 or n<1 or max(lst)>10**9:
        quit("Входные данные не соответствуют условию задачи!")

    lst_ans = insertion_sort(n, lst)

    lst_ind, lst_num = list(map(str, lst_ans[1])), list(map(str, lst_ans[0]))
    out_1 = " ".join(lst_ind) + '\n'
    out_2 = " ".join(lst_num)

    file_out = open("output.txt", 'w')
    file_out.write(out_1)
    file_out.write(out_2)
    file_out.close()

main()
```

Тесты:

1) Тест на время работы:

```
from lab1.task2.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))
```

2) Тест на занимаемую память:

```
from lab1.task2.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

Текстовое объяснение решения.

Реализовано 2 функции:

1) main():

Открывает файл input.txt и считывает из него n – количество элементов и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). В переменную lst_ans записывает результат работы функции insertion_sort(). Затем разделяет этот список на 2: lst_num и lst_ind, содержащие числа и индексы, превращенные в строки, соответственно. С помощью метода join() формирует 2 выходных сообщения, которые далее записывает в файл output.txt.

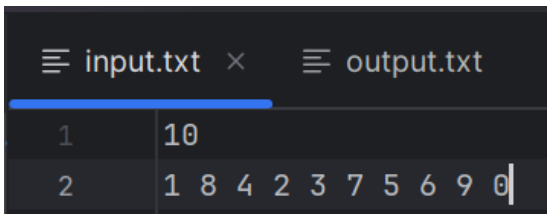
2) insertion_sort():

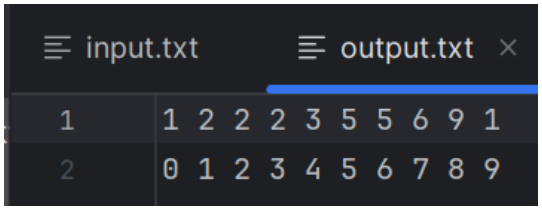
Получает на вход длину списка и сам список. Создает список lst_ind, заполненный единицами, который будет содержать индексы. Циклом for начинает бежать по нему от 1 до n-го индекса. В переменную elem записываем текущий элемент и создаем j, содержащую индекс предыдущего элемента. Запускаем цикл while, который в случае, если предыдущий элемент больше текущего, сдвинет больший вперед. В конце сдвигов элементу, который был сдвинут последним, выдаем сохраненное в elem значение. Затем мы проверяем частный случай с i=1. Если это не частный случай, то записываем в lst_ind на место с индексом i индекс выбранного элемента + 1. И так продолжаем до конца списка, после чего возвращаем кортеж из нужных нам списков.

Тесты:

1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.
2) Импортируем нашу функцию main() и с помощью встроенной библиотеки tracemalloc замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

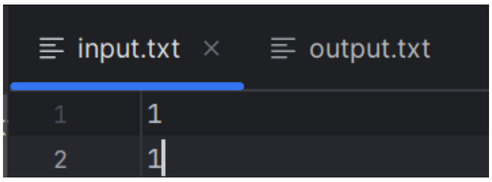
Результат работы кода на примерах из текста задачи:

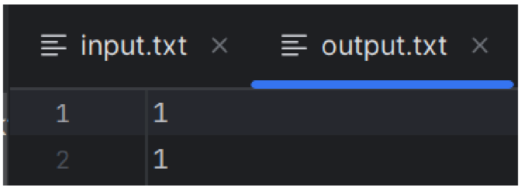


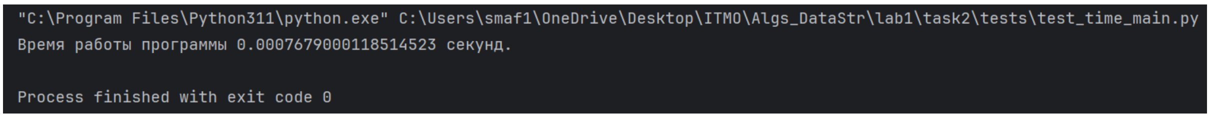


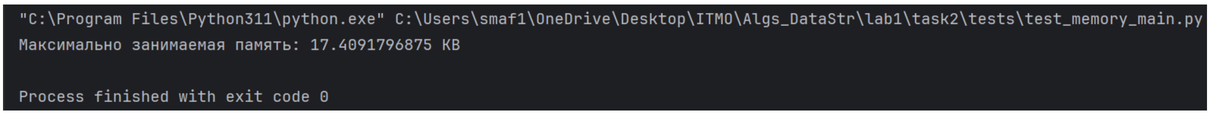
Результат работы кода на максимальных и минимальных значениях:

1) Минимальные значения:

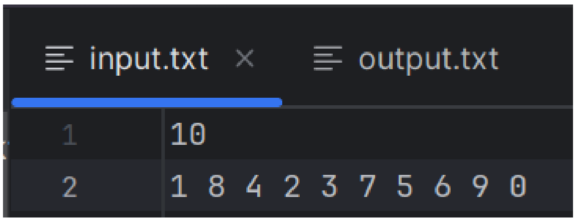


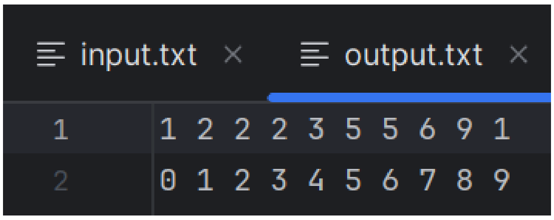


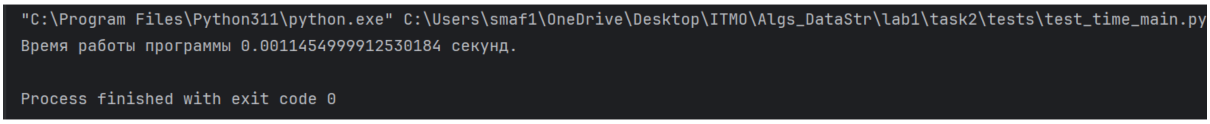


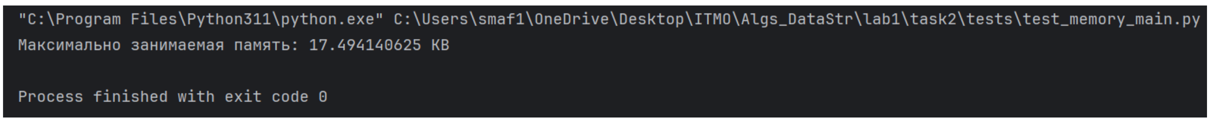


2) Значения из примеров:

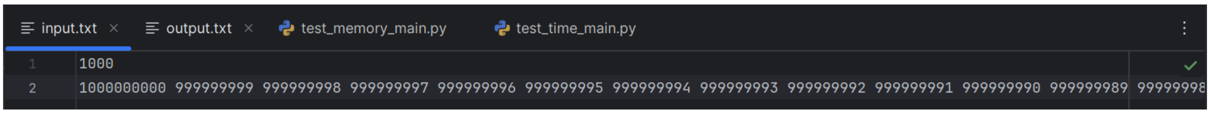


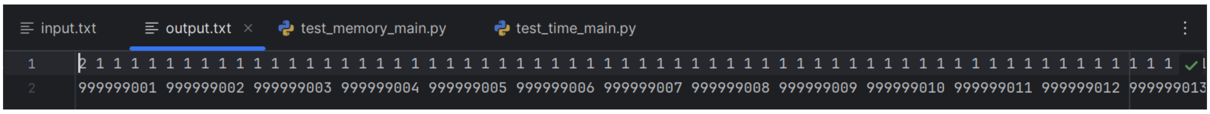


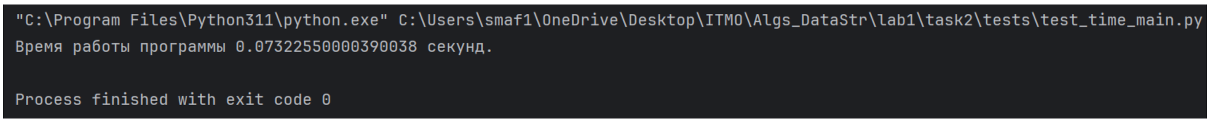


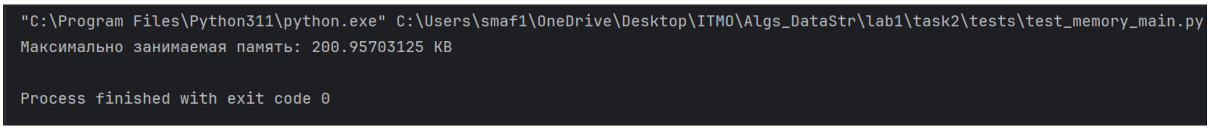


3) Максимальные значения:









	Время выполнения, с	Затраты памяти, KB
Нижняя граница диапазона значений входных данных из текста задачи	0.000768	17.409
Пример из задачи	0.001145	17.494

Верхняя граница диапазона значений входных данных из текста задачи	0.073226	200.957

Вывод по задаче: Сортировка вставкой является довольно затратной сортировкой по времени и памяти.

Задача №4. Линейный поиск

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.

Листинг кода

```
def linear_search(elem, lst):
    n = len(lst)
    cnt=0
    count = -1
    out_lst = []
    for i in range(n):
        if elem==lst[i]:
            cnt+=1
            out_lst.append(i)
    if cnt>0:
        count = cnt
    return (count, out_lst)

def main():
    file_inp = open("input.txt")
    lst = file_inp.readline().split()
    elem = file_inp.readline().strip()
    file_inp.close()

    if len(lst)>1000:
        quit("Входные данные не соответствуют условию задачи!")

    lst_ans = linear_search(elem, lst)

    count, lst = str(lst_ans[0])+'\n', ', '.join(list(map(str, lst_ans[1])))

    file_out = open("output.txt", 'w')
    file_out.write(count)
    file_out.write(lst)
    file_out.close()

main()
```

Тесты:

1) Тест на время работы:

```
from lab1.task4.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))
```

2) Тест на занимаемую память:

```
from lab1.task4.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

Текстовое объяснение решения.

Реализовано 2 функции:

1) main():

Открывает файл input.txt и считывает из него elem – элемент, который нужно искать и список (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). В переменную lst_ans записывает результат работы функции linear_search(). Затем разделяет этот кортеж на 2 элемента: count (сколько раз встретился элемент) в который попадает 1-ый элемент кортежа, превращенный в строку с добавлением ‘\n’, и out_lst – список индексов на которых стоит этот элемент, превращенный в строку с разделением запятой между ними с помощью join(). Тем самым формирует 2 выходных сообщения, которые далее записывает в файл output.txt.

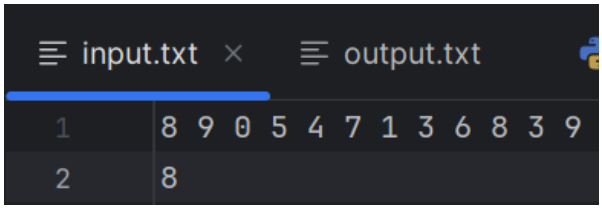
2) linear_search():

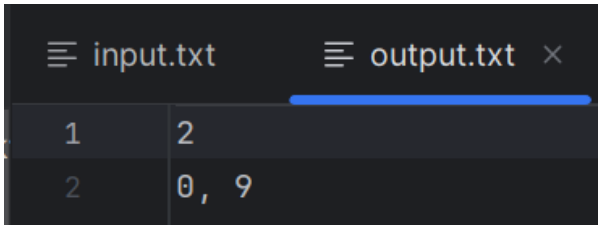
Получает на вход элемент и сам список. В переменную n записывает его длину. Создает переменные cnt – счётчик повторений и count – количество повторений для вывода, а также out_lst – список, который будет содержать индексы элемента в списке. Циклом for проходим по элементам списка и проверяем, совпадают ли они с нашим. Если да, то счетчик cnt увеличиваем на 1 и записываем индекс в out_lst. И так продолжаем до конца списка, после чего проверяем счетчик. Если он изменился, то перезаписываем его значение в count. Выводим кортеж из count и out_lst.

Тесты:

1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.
2) Импортируем нашу функцию main() и с помощью встроенной библиотеки tracemalloc замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

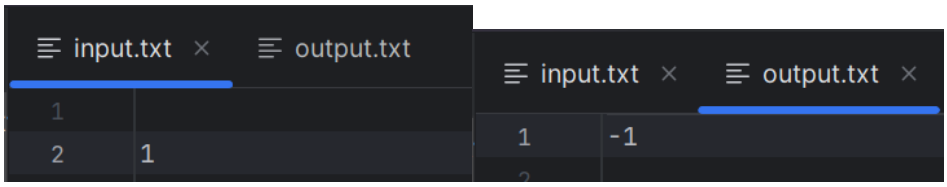
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:

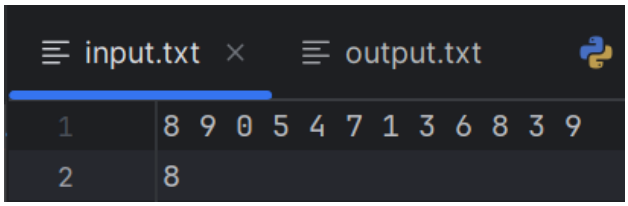
1) Минимальные значения:

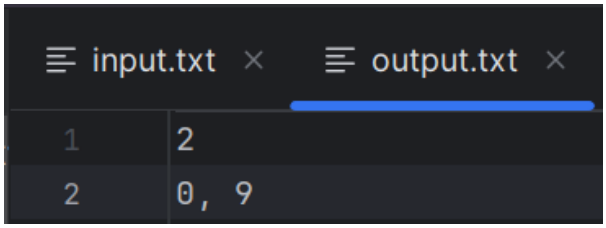



```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task4\tests\test_time_main.py
Время работы программы 0.0007249999907799065 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task4\tests\test_memory_main.py
Максимально занимаемая память: 17.501953125 KB

Process finished with exit code 0
```

2) Значения из примера:





```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task4\tests\test_time_main.py
Время работы программы 0.0008889999880921096 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task4\tests\test_memory_main.py
Максимально занимаемая память: 17.5244140625 KB

Process finished with exit code 0
```

3) Максимальные значения:

input.txt | output.txt | test_memory_main.py | test_time_main.py | main.py

```
1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
1000
```

input.txt | output.txt | test_memory_main.py | test_time_main.py | main.py

```
1000
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task4\tests\test_time_main.py
Время работы программы 0.001282200013520196 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task4\tests\test_memory_main.py
Максимально занимаемая память: 167.3095703125 KB

Process finished with exit code 0
```

	Время выполнения, с	Затраты памяти, KB
Нижняя граница диапазона значений входных данных из текста задачи	0.000725	17.502
Пример из задачи	0.000889	17.524
Верхняя граница диапазона значений входных данных из текста задачи	0.001282	167.309

Вывод по задаче: Линейный поиск работает не так быстро и занимает довольно много памяти на больших данных.

Задача №5. Сортировка выбором

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.

Листинг кода

```
def selection_sort(n, lst):
    for i in range(n-1):
        m=10000000000
        ind = -1
        for j in range(i,n):
            if lst[j]<m:
                m=lst[j]
                ind=j
        lst[i], lst[ind] = lst[ind], lst[i]
    return lst



def main():
    file_inp = open("input.txt")
    n = int(file_inp.readline())
    lst = list(map(int, file_inp.readline().split()))
    file_inp.close()

    if len(lst)!= n:
        quit("Указанная длина списка не соответствует действительной!")

    if n>1000 or n<1 or max(lst)>10**9:
        quit("Входные данные не соответствуют условию задачи!")

    lst_ans = selection_sort(n, lst)

    lst = ' '.join(list(map(str, selection_sort(n, lst))))

    file_out = open("output.txt", 'w')
    file_out.write(lst)
    file_out.close()

main()
```

Тесты:

1) Тест на время работы:

```
from lab1.task5.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))
```

2) Тест на занимаемую память:

```
from lab1.task5.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

Текстовое объяснение решения:
Реализовано 2 функции:

1) main():

Открывает файл input.txt и считывает из него n – количество элементов и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается) и перезаписывает переменную lst_ans, положив туда список, отсортированный функцией selection_sort(). Затем с помощью метода join() и обращения элементов списка в строку формирует выходное сообщение, которое далее записывает в файл output.txt.

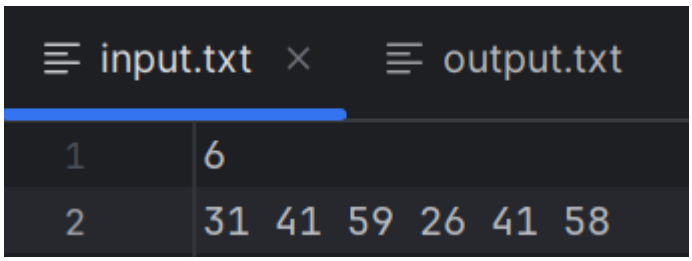
2) selection_sort():

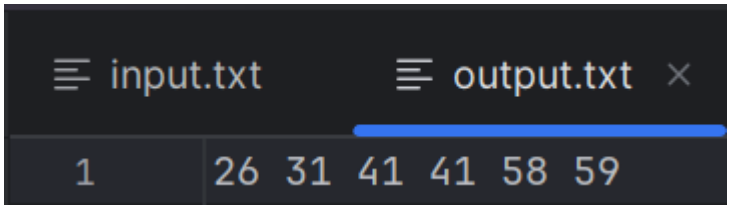
Получает на вход длину списка и сам список. Циклом for начинает бежать по нему от 0 до (n-1)-го индекса. В переменную m записывает максимальное число из доступных нам ограничений, чтобы потом там содержался минимум списка. Создает переменную ind с временным значением -1. В ней далее будет содержаться индекс элемента, с которым мы будем менять местами на i-ый элемент. Запускаем цикл for от i до n. Если нам встречается элемент с индексом j, который меньше нашего минимума, то перезаписываем минимум и записываем в ind индекс j. После прохождения внутреннего цикла меняем элементы с индексами i и j местами. И так продолжаем до конца списка, после чего возвращаем его.

Тесты:

1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.
2) Импортируем нашу функцию main() и с помощью встроенной библиотеки tracemalloc замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

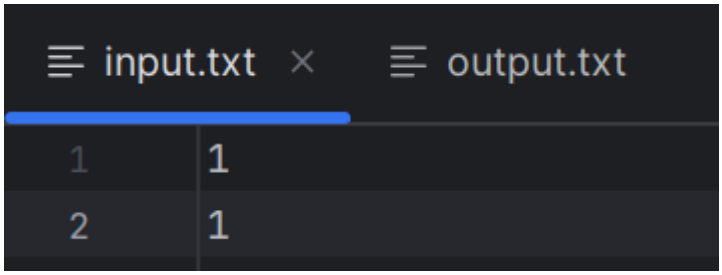
Результат работы кода на примерах из текста задачи:

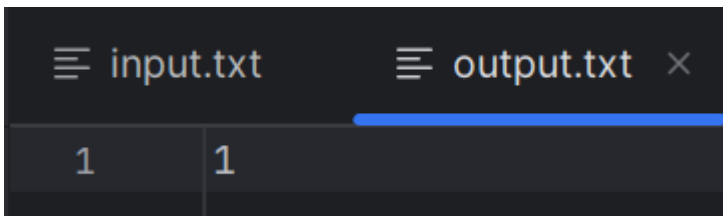




Результат работы кода на максимальных и минимальных значениях:

1) Минимальные значения:

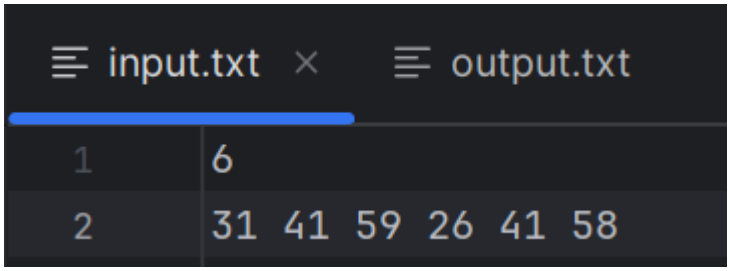


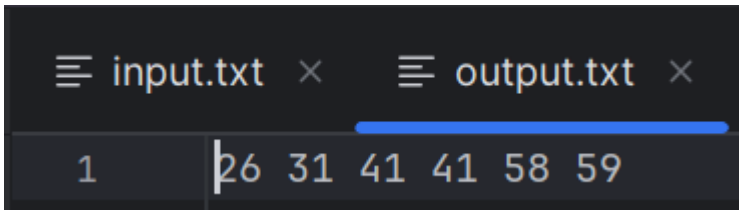



```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task5\tests\test_time_main.py
Время работы программы 0.0007431999983964488 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task5\tests\test_memory_main.py
Максимально занимаемая память: 17.4091796875 KB

Process finished with exit code 0
```

2) Значения из примера:

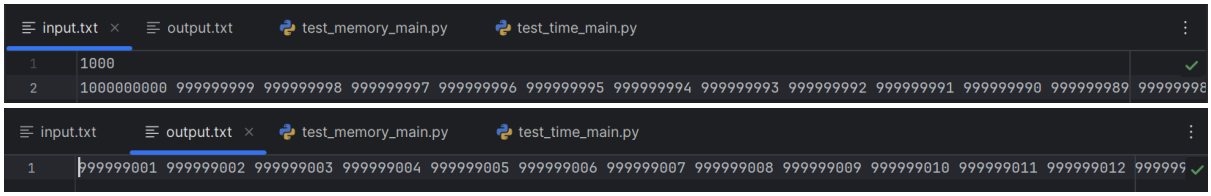





```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task5\tests\test_time_main.py
Время работы программы 0.0008177999989129603 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task5\tests\test_memory_main.py
Максимально занимаемая память: 17.4892578125 KB

Process finished with exit code 0
```

3) Максимальные значения:




```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task5\tests\test_time_main.py
Время работы программы 0.04637919999368023 секунд.

Process finished with exit code 0
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\Algs_DataStr\lab1\task5\tests\test_memory_main.py
Максимально занимаемая память: 112.1044921875 KB

Process finished with exit code 0
```

	Время выполнения, с	Затраты памяти, КВ
Нижняя граница диапазона значений входных данных из текста задачи	0.000743	17.409
Пример из задачи	0.000818	17.489
Верхняя граница диапазона значений входных данных из текста задачи	0.046379	112.104

Вывод по задаче: Сортировка выбором является довольно затратной по времени и занимаемой памяти.

Вывод

Сортировки сложностью O(n**2) (вставками, пузырьковая, выбором) и Линейный поиск, являются довольно затратными по времени работы и занимаемой памяти.