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: Сортировки.
Вариант 9

Выполнила:
Гашимов И.Ф.
К3139

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Сперва начнем с обязательный задач для моего номера: 9, а именно **1, 3, 6.**

1. 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.

Ниже показано как работает данный способ:

6 5 3 1 8 7 2 4

Используем код процедуры Insertion-sort, в данном коде в цикле for мы указываем количество элементов через len(arr)

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
```

```
            arr[j + 1] = key
    return arr
```

Здесь написал 3 проверки:

1. Если количество элементов не в диапазоне от 1 до 1000, то код прекращается

```
if 1 <= n <= 1000:
    pass
else:
    print('[Ошибка] Количество элементов должно быть от 1 до
1000!')
    exit(1)
```

2. Проверяет количество введенных данных и сравнивает их с заданными в input.txt, если они не совпадают, то код закрывается к кодом ошибки 1

```
if len(unsorted_base) != n:
    print('[Ошибка] Количество элементов и введенное число
элементов не совпадает!')
    exit(1)
```

3. Проверка чисел, если какое-то число больше 10**9 по модулю, то код прекращается с ошибкой 1

```
for i in unsorted_base:
    if abs(i) > 10**9:
        print('[Ошибка] Числа должны быть не больше 10**9 по
модулю!')
        exit(1)
```

Открываем input.txt, n присваиваем количество элементов, а unsorted_base присваиваем лист с элементами

```
with open('input.txt', 'r') as f:
    n = int(f.readline())
    unsorted_base = list(map(int,f.readline().split()))
```

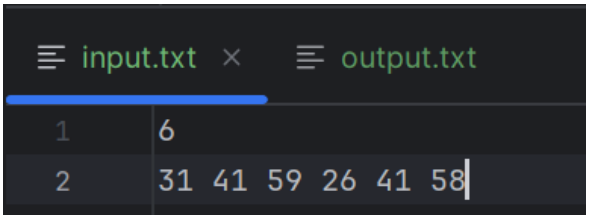
Затем уже с помощью функции сортируем данные в новую переменную

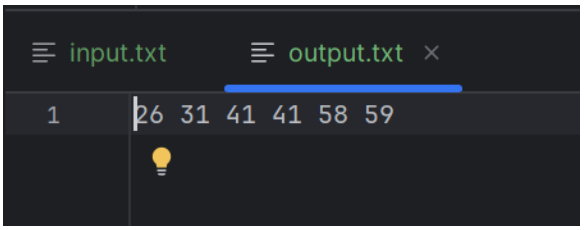
```
sorted_base = insertion_sort(unsorted_base)
```

Записываем новый массив в output.txt, заранее сделав str с разделителем “ “

```
with open('output.txt', 'w') as f:
    f.write(' '.join(map(str,sorted_base)))
```

в input.txt задаем данные, затем получаем output.txt





Время и память исполнения кода:

```
Время сортировки: 0.000137 секунд
Использование памяти: 0.003906250000 МБ
```

Полный код:

```
import time, os, psutil
def get_memory_usage():
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / (1024 ** 2)

initial_memory = get_memory_usage()
start_time = time.perf_counter()
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

with open('input.txt', 'r') as f:
```

```
    n = int(f.readline())
    unsorted_base = list(map(int,f.readline().split()))
#ПРОВЕРКИ
if 1 <= n <= 1000:
    pass
else:
    print('[Ошибка] Количество элементов должно быть от 1 до
1000!')
    exit(2)

if len(unsorted_base) != n:
    print('[Ошибка] Количество элементов и введенное число
элементов не совпадает!')
    exit(3)

for i in unsorted_base:
    if abs(4) > 10**9:
        print('[Ошибка] Числа должны быть не больше 10**9 по
модулю!')
        exit(1)
#ПРОВЕРКИ

sorted_base = insertion_sort(unsorted_base)

end_time = time.perf_counter()
final_memory = get_memory_usage()
time = end_time - start_time
memory_used = final_memory - initial_memory
print(f"Время сортировки: {time:.6f} секунд")
print(f"Использование памяти: {memory_used:.12f} МБ")



with open('output.txt', 'w') as f:
    f.write(' '.join(map(str,sorted_base)))
```

3.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?

Здесь мы используем такой же код как и в 1. , но отличаются функциями insertion sort.

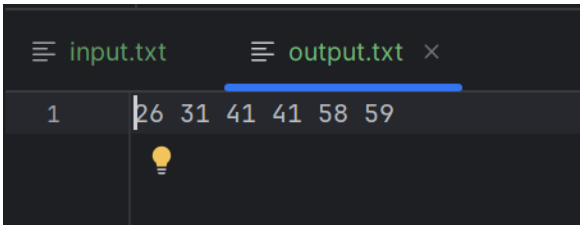
Функция **insertion_sort_recursive** реализует алгоритм сортировки вставками в рекурсивной форме.

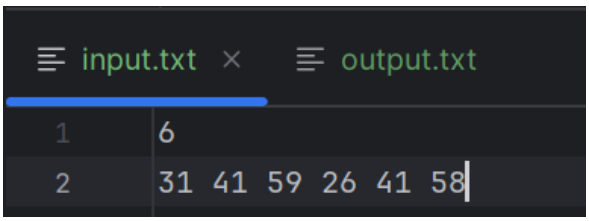
- Если **n** меньше или равно 1, функция возвращает массив, так как массив с одним элементом уже считается отсортированным.

- Функция рекурсивно вызывает саму себя для первых **n-1** элементов массива. Это предполагает, что эти **n-1** элементов уже отсортированы.
- для key(последнего) элемента ищется подходящее место
- После завершения всех рекурсивных вызовов, функция возвращает отсортированный массив.

```
def insertion_sort_recursive(arr, n):
    if n <= 1:
        return arr

    insertion_sort_recursive(arr, n - 1)

    key = arr[n - 1]
    j = n - 2

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

    arr[j + 1] = key
    return arr
```

в input.txt задаем данные, затем получаем output.txt





Время исполнения программы и использование памяти:

```
Время сортировки: 0.000137 секунд
Использование памяти: 0.00390625 МБ
```

Полный код:

```
import time, os, psutil
def get_memory_usage():
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / (1024 ** 2)

initial_memory = get_memory_usage()
start_time = time.perf_counter()
def insertion_sort_recursive(arr, n):
    if n <= 1:
        return arr

    insertion_sort_recursive(arr, n - 1)

    key = arr[n - 1]
    j = n - 2

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

    arr[j + 1] = key
    return arr
with open('input.txt', 'r') as f:
    n = int(f.readline())
    unsorted_base = list(map(int,f.readline().split()))
#ПРОВЕРКИ
if 1 <= n <= 1000:
    pass
else:
    print('[Ошибка] Количество элементов должно быть от 1 до
1000!')
    exit(2)
```

```
if len(unsorted_base) != n:
   print('[Ошибка] Количество элементов и введенное число
элементов не совпадает!')
   exit(3)

for i in unsorted_base:
   if abs(4) > 10**9:
       print('[Ошибка] Числа должны быть не больше 10**9 по
модулю!')
       exit(1)
#ПРОВЕРКИ

sorted_base =
insertion_sort_recursive(unsorted_base,len(unsorted_base))

end_time = time.perf_counter()
final_memory = get_memory_usage()
time = end_time - start_time
memory_used = final_memory - initial_memory
print(f"Время сортировки: {time:.6f} секунд")
print(f"Использование памяти: {memory_used:.8f} МБ")



with open('output.txt', 'w') as f:
   f.write(' '.join(map(str,sorted_base)))
```

6.Пузырьковая сортировка

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:

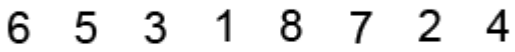
```
Bubble_Sort(A):
  for i = 1 to A.length - 1
    for j = A.length downto i+1
      if A[j] < A[j-1]
        поменять A[j] и A[j-1] местами
```

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.

Вот как выглядит метод пузырька



Вот функция **bubble_sort(a)** в Python формате

```python
def bubble_sort(a):
    for i in range(n - 1):
        flag = False
        for j in range(n - 1 - i):
```

```
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
                flag = True
        if not flag:
            break
    return a
```

Код идентичен 1. сортировке, отличается самая функция сортировки, которая дана выше. так же использовал флаг, чтобы остановить цикл если отсортировано

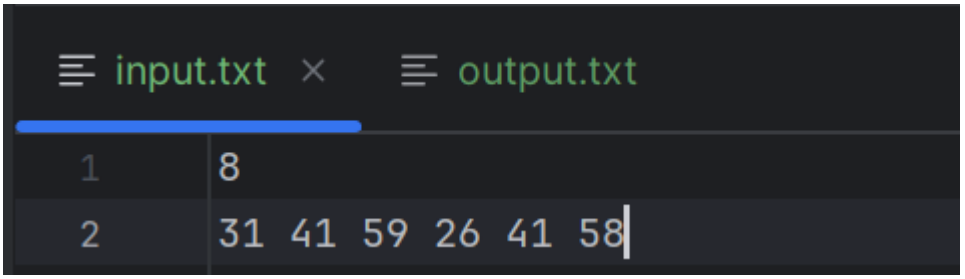
Полный код:

```
import time, os, psutil

def get_memory_usage():
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / (1024 ** 2)

initial_memory = get_memory_usage()
start_time = time.perf_counter()

def bubble_sort(a):
    for i in range(n - 1):
        flag = False
        for j in range(n - 1 - i):
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
                flag = True
        if not flag:
            break
    return a


with open('input.txt', 'r') as f:
    n = int(f.readline())
    unsorted_base = list(map(int,f.readline().split()))
#ПРОВЕРКИ
if 1 <= n <= 1000:
    pass
else:
    print('[Ошибка] Количество элементов должно быть от 1 до
1000!')
    exit(2)

if len(unsorted_base) != n:
    print('[Ошибка] Количество элементов и введенное число
элементов не совпадает!')
```

```
    exit(3)

for i in unsorted_base:
    if abs(4) > 10**9:
        print('[Ошибка] Числа должны быть не больше 10**9 по
модулю!')
        exit(1)
#ПРОВЕРКИ

sorted_base = bubble_sort(unsorted_base)

end_time = time.perf_counter()
final_memory = get_memory_usage()
time = end_time - start_time
memory_used = final_memory - initial_memory
print(f"Время сортировки: {time:.6f} секунд")
print(f"Использование памяти: {memory_used:.8f} МБ")



with open('output.txt', 'w') as f:
    f.write(' '.join(map(str,sorted_base)))
```

Доказательство верности сортировки:

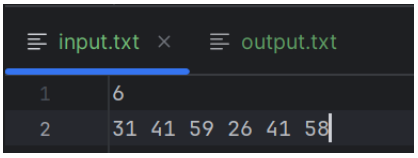
Снизу задаем сперва неправильные данные для input.txt ((Неправильное количество(n)))

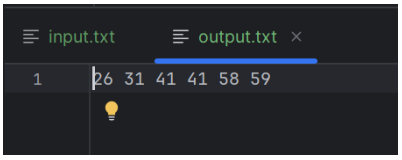


получаем

```
C:\Users\Colorful\AppData\Local\Microsoft\WindowsApps\python3.11.exe C:\Users\Colorful\PycharmProjects\algosi\src\lab1\task6\bubble_sort.p
[Ошибка] Количество элементов и введенное число элементов не совпадает!
```

Задаем правильные данные в input.txt:





Время и память:

Как видим время и память используется больше, чем в предыдущих

```
Время сортировки: 0.000145 секунд
Использование памяти: 0.01171875 МБ
```

Полный код:

```
import time, os, psutil

def get_memory_usage():
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / (1024 ** 2)

initial_memory = get_memory_usage()
start_time = time.perf_counter()

def bubble_sort(a):
    for i in range(1, len(a)):
        for j in range(len(a) - 1, i - 1, -1):
            if a[j] < a[j - 1]:
                a[j], a[j - 1] = a[j - 1], a[j]
    return a

with open('input.txt', 'r') as f:
    n = int(f.readline())
    unsorted_base = list(map(int,f.readline().split()))
#ПРОВЕРКИ
if 1 <= n <= 1000:
    pass
else:
    print('[Ошибка] Количество элементов должно быть от 1 до
1000!')
    exit(2)

if len(unsorted_base) != n:
    print('[Ошибка] Количество элементов и введенное число
элементов не совпадает!')
    exit(3)

for i in unsorted_base:
    if abs(4) > 10**9:
        print('[Ошибка] Числа должны быть не больше 10**9 по
модулю!')
        exit(1)
```

```python
#ПРОВЕРКИ

sorted_base = bubble_sort(unsorted_base)

end_time = time.perf_counter()
final_memory = get_memory_usage()
time = end_time - start_time
memory_used = final_memory - initial_memory
print(f"Время сортировки: {time:.6f} секунд")
print(f"Использование памяти: {memory_used:.8f} МБ")



with open('output.txt', 'w') as f:
   f.write(' '.join(map(str,sorted_base)))
```

2. 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]

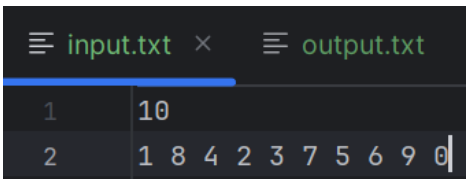
Изменил функцию так, чтобы она в конце def давала индекс измененного числа:

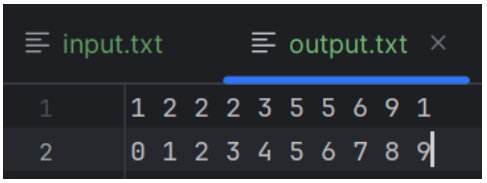
- создал в начале список состоящий из нулей в n’ном количестве
- дальше в конце кода добавил sp_ind[i] = j + 1, т.к. мы начинали цикл с 1
- добавил к каждому числу 1, т.к. нумерация начинается с числа 1
- затем получаем сам список и их индекса

```
def insertion_sort_plus(arr):
    sp_ind = [0] * len(arr)
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
        sp_ind[i] = j + 1
    sp_ind = [index+1 for index in sp_ind]
```

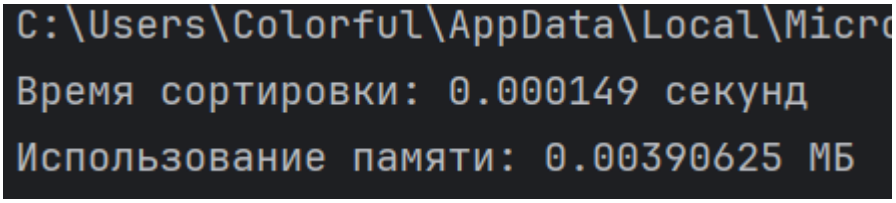
```
    return arr, sp_ind
```

input и output соответственно:





Время и память:



Полный код(одинаковы все, кроме функций, в первом объяснил код):

```
import time, os, psutil
def get_memory_usage():
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / (1024 ** 2)

initial_memory = get_memory_usage()
start_time = time.perf_counter()
def insertion_sort_plus(arr):
    sp_ind = [0] * len(arr)
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
        sp_ind[i] = j + 1
    sp_ind = [index+1 for index in sp_ind]
    return arr, sp_ind

with open('input.txt', 'r') as f:
    n = int(f.readline())
    unsorted_base = list(map(int,f.readline().split()))
#ПРОВЕРКИ
if 1 <= n <= 1000:
    pass
else:
```

```
    print('[Ошибка] Количество элементов должно быть от 1 до
1000!')
    exit(2)

if len(unsorted_base) != n:
    print('[Ошибка] Количество элементов и введенное число
элементов не совпадает!')
    exit(3)

for i in unsorted_base:
    if abs(4) > 10**9:
        print('[Ошибка] Числа должны быть не больше 10**9 по
модулю!')
        exit(1)
#ПРОВЕРКИ

sorted_base, indexes = insertion_sort_plus(unsorted_base)

end_time = time.perf_counter()
final_memory = get_memory_usage()
time = end_time - start_time
memory_used = final_memory - initial_memory
print(f"Время сортировки: {time:.6f} секунд")
print(f"Использование памяти: {memory_used:.8f} МБ")


with open('output.txt', 'w') as f:
    f.write(' '.join(map(str,indexes))+'\n')
    f.write(' '.join(map(str,sorted_base)))
```

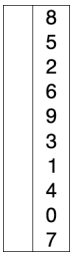
5.Сортировка выбором

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.



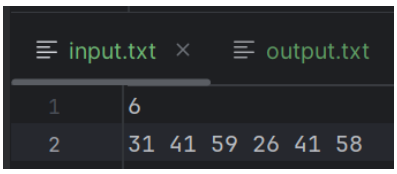
Вот пример работы данной сортировки

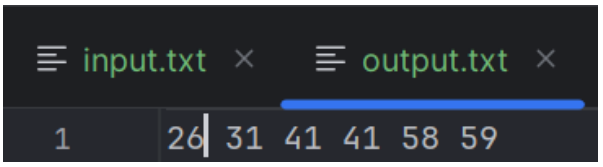
Код идентичен с 1. кроме функции сортировки, снизу можно увидеть ее:

- можно заметить что цикл задан до n -1, т.к. после такой сортировки в конце гарантированно останется самый большой элемент

```
def selection_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        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]
    return arr
```

input и output соответственно:





Время и память:

```
Время сортировки: 0.000151 секунд
Использование памяти: 0.00390625 МБ
```

Полный код(идентичен первому, только функция отличается):

```
import time, os, psutil
def get_memory_usage():
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / (1024 ** 2)

initial_memory = get_memory_usage()
start_time = time.perf_counter()
def selection_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        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]
    return arr

with open('input.txt', 'r') as f:
    n = int(f.readline())
    unsorted_base = list(map(int,f.readline().split()))
#ПРОВЕРКИ
if 1 <= n <= 1000:
    pass
else:
    print('[Ошибка] Количество элементов должно быть от 1 до
1000!')
    exit(2)

if len(unsorted_base) != n:
    print('[Ошибка] Количество элементов и введенное число
элементов не совпадает!')
    exit(3)

for i in unsorted_base:
```

```
   if abs(4) > 10**9:
       print('[Ошибка] Числа должны быть не больше 10**9 по
модулю!')
       exit(1)
#ПРОВЕРКИ

sorted_base = selection_sort(unsorted_base)

end_time = time.perf_counter()
final_memory = get_memory_usage()
time = end_time - start_time
memory_used = final_memory - initial_memory
print(f"Время сортировки: {time:.6f} секунд")
print(f"Использование памяти: {memory_used:.8f} МБ")



with open('output.txt', 'w') as f:
   f.write(' '.join(map(str,sorted_base)))
```

7.Жители Сортленда

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 ($3 \leq n \leq 9999$, 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.

Использую сортировку с первой задачи:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr
```

- Делаем копию M Через M[:] и присваиваем ее к переменной M_old
- Делаем сортировку M
- затем присваиваем к переменным, минимальное, среднее и самое большое значение
- затем прибавляем 1 т.к. нумерация индекса должна идти с 1
- передаем значения в output.txt

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

with open('input.txt', 'r') as f:
    n = int(f.readline())
    M = list(map(float, f.readline().split()))

M_old = M[:]
M = insertion_sort(M)

min_man = M[0]
middle_man = M[n // 2]
max_man = M[-1]

min_index = M_old.index(min_man) + 1
middle_index = M_old.index(middle_man) + 1
max_index = M_old.index(max_man) + 1

with open('output.txt', 'w') as f:
    f.write(f"{min_index} {middle_index} {max_index}")
```

Полный код:

```
import time, os, psutil

def get_memory_usage():
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / (1024 ** 2)

initial_memory = get_memory_usage()
start_time = time.perf_counter()

def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

with open('input.txt', 'r') as f:
    n = int(f.readline())
    M = list(map(float, f.readline().split()))

M_old = M[:]
M = insertion_sort(M)

min_man = M[0]
middle_man = M[n // 2]
max_man = M[-1]

min_index = M_old.index(min_man) + 1
middle_index = M_old.index(middle_man) + 1
max_index = M_old.index(max_man) + 1

with open('output.txt', 'w') as f:
    f.write(f"{min_index} {middle_index} {max_index}")

end_time = time.perf_counter()
```

```
final_memory = get_memory_usage()

elapsed_time = end_time - start_time
memory_used = final_memory - initial_memory

print(f"Время сортировки: {elapsed_time:.6f} секунд")
print(f"Использование памяти: {memory_used:.8f} МБ")
```