

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 8

Выполнил:
Макунина Арина
К32421

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Задачи по варианту	3
Задача №1. Сортировка вставкой	3
Задача №3. Сортировка вставкой по убыванию	5
Задача №5. Сортировка выбором	7
Вывод	9

Задачи по варианту

Задача №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.

Первая часть задачи решается достаточно примитивно: мы используем представленный нам метод сортировки вставкой, который описывается достаточно банально: очередной элемент $A[j]$ перемещается в начало списка до тех пор, пока он меньше предыдущего элемента.

```
def InsertionSort(A):
    for j in range(2, len(A)):
        key = A[j]
        i = j - 1
        while (i > 0 and A[i] > key):
            A[i + 1] = A[i]
            i = i - 1
        A[i + 1] = key
    return A

A = [31, 41, 59, 26, 41, 58]
print(InsertionSort(A))
```

Вторая часть задачи потребовала видоизменить метод, но смысл сортировки вставкой остался прежним: элементы сравниваются по очереди с теми, которые стоят слева от него, и до тех пор, пока они не больше его, они не переставляются. Затем, когда найден элемент слева, равный или меньший, чем элемент, элемент вставляется в указанное место с помощью

insert(), при этом он удаляется со своего изначального места с помощью pop().

```
def InsertionSort(A, n):
    for j in range(1, n):
        key = j
        while (A[j] < A[key - 1]):
            key = key - 1
        A.insert(max(key, 0), A[j]) # вставляем
элемент по индексу
        A.pop(j + 1) # получаем элемент по индексу,
удаляя его из последовательности
    return A

f = open('input.txt')
n = int(f.readline())
A = list(map(int, f.readline().split()))
f.close()
w = open('output.txt', 'w')
w.write(' '.join(list(map(str, InsertionSort(A,
n)))))
w.close()
```

Результат работы кода:

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

Результат работы кода на максимальных и минимальных значениях:

input.txt	output.txt
1 8	1 8

input.txt	output.txt
1000 5 8 3 1 9 11 2 3 9	1 1 2 3 3 5 8 9 9 10 11 11 12 13 14 15 16 17 18 19 20 21

	Время выполнения, с	Максимальные затраты памяти, МБ
Нижняя граница диапазона значений	0.0012743000000000006	0.017287254333496094

ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ		
Пример из текста задачи	0.0010901999999999995	0.01736164093017578
Верхняя граница диапазона значений ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ	0.014482000000000009	0.10016250610351562

Вывод по задаче: количество времени на работу данного алгоритма полностью зависит от входных данных. Больше времени потребуется на сортировку массива, в котором значения абсолютно не упорядочены. Данный метод, на мой взгляд, хорош для небольших массивов с частично отсортированными элементами.

Задача №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? – да!

Решение аналогично сортировке вставкой из задачи №1. Для того, чтобы массив был корректно отсортирован в порядке убывания, необходимо «перевернуть» его, что реализовано в процедуре Swap, где мы используем функцию reverse, которая меняет местами элементы списка.

```
def InsertionSort(A, n):
    for j in range(1, n):
        key = j
        while (A[j] < A[key - 1]):
            key = key - 1
        A.insert(max(key, 0), A[j]) # вставляем
элемент по индексу
        A.pop(j + 1) # получаем элемент по индексу,
удаляя его из последовательности
    return Swap(A)
```

```
def Swap(A):
    A.reverse() # меняем местами элементы массива
    return A

f = open('input.txt')
n = int(f.readline())
A = list(map(int, f.readline().split()))
f.close()
w = open('output.txt', 'w')
w.write(' '.join(list(map(str, InsertionSort(A,
n)))))
w.close()
```

Результат работы кода:

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

Результат работы кода на максимальных и минимальных значениях:

input.txt ×	output.txt ×
1	1
4	4

input.txt ×	output.txt ×
1000	1
1 2 3 4 5 6 7 8 9	1000 999 998 997 996 995 994 993 992 991 990 989 988 987

	Время выполнения, с	Максимальные затраты памяти, МБ
Нижняя граница диапазона значений входных данных из текста задачи	0.0021261000000000058	0.017416954040527344
Пример из задачи	0.0012992999999999894	0.01749134063720703
Верхняя граница диапазона значений	0.008779299999999999	0.10029029846191406

ВХОДНЫХ ДАННЫХ ИЗ текста задачи		
------------------------------------	--	--

Вывод по задаче: аналогичен задаче №1. Использование внутренних функций не усложняет алгоритм и минимально влияет на его оптимальность.

Задача №5. 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.

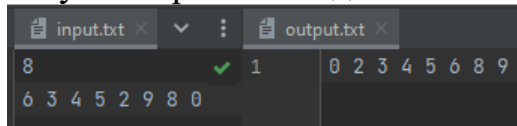
В программном методе сортировки выбором для начала мы ищем локальный минимум, меняем его местами с первым элементом в подмассиве с индексом j и продолжаем выполнять цикл for до полной сортировки массива через подмассивы. Таким образом, сортировка выбором основана на постоянном поиске минимального элемента массива.

```
def SelectionSort(A, n):
    for i in range(n):
        min = i
        for j in range(i + 1, n):
            if A[min] > A[j]:
                min = j
        A[i], A[min] = A[min], A[i]
    return A

f = open('input.txt')
n = int(f.readline())
A = list(map(int, f.readline().split()))
f.close()
w = open('output.txt', 'w')
w.write(' '.join(list(map(str, SelectionSort(A,
```

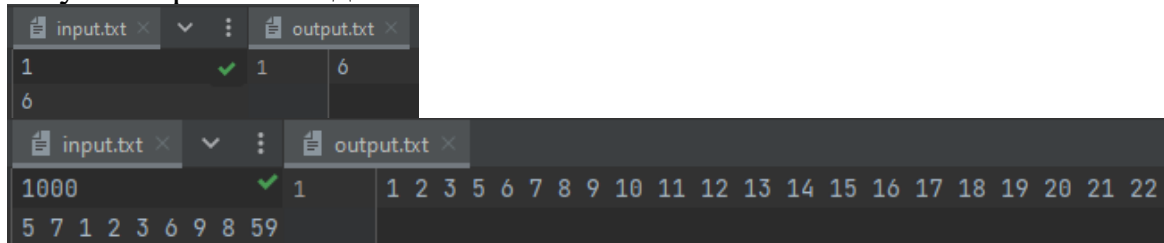
```
n) ) ) )
w.close()
```

Результат работы кода:



```
input.txt x  v  :  output.txt x
8              1  0 2 3 4 5 6 8 9
6 3 4 5 2 9 8 0
```

Результат работы кода на максимальных и минимальных значениях:



```
input.txt x  v  :  output.txt x
1              1  6
6

input.txt x  v  :  output.txt x
1000           1  1 2 3 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
5 7 1 2 3 6 9 8 59
```

	Время выполнения, с	Максимальные затраты памяти, МБ
Нижняя граница диапазона значений входных данных из текста задачи	0.00135530000000000037	0.017287254333496094
Пример из задачи	0.00161600000000000063	0.01736164093017578
Верхняя граница диапазона значений входных данных из текста задачи	0.3479951	0.10016250610351562

Вывод по задаче: необходимо рассматривать последовательность как две части (отсортированные и неотсортированные элементы), но это достаточно быстрый и экономный процесс, как мы убедились на тесте с 1000 элементов в массиве.

Вывод

В ходе выполнения данной лабораторной работы я познакомилась и реализовала на Python два вида простых сортировок: вставкой и выбором. Получилось весьма успешно, потому что в оценке скорости выполнения и эффективности использования памяти при компиляции алгоритмы уложились в норму.