

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: Тема работы
Вариант 18

Выполнила:
Петрова М. В
К3139

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка вставкой [N баллов]	3
Задача №5. Сортировка выбором [N баллов]	5
Задача №6. Пузырьковая сортировка [N баллов]	8
Дополнительные задачи	10
Задача №2. Сортировка вставкой+ [N баллов]	10
Задача №7. Знакомство с жителями Сортлэнда [N баллов]	13
Задача №10. Палиндром [N баллов]	16
Вывод	

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.

```
import time
start = time.perf_counter()

with open('input.txt') as f:
    ind_list = []
    n = int(f.readline())
    array = list(map(int, f.readline().split()))
    for i in range(1, n):
        x = array[i]
        j = i

        while j > 0 and array[j - 1] > x:
            array[j] = array[j - 1]
            j -= 1

        array[j] = x

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

```
stop = time.perf_counter()
print("time: %s ms" % (stop - start))
```

Текстовое описание решения

1. Импортируем библиотеку time и засекаем время начала.
2. Чтение данных из файла
3. Сортируем массив вставками
 - 3.1 Итерируется по индексам массива, начиная с 1 до n-1.
 - 3.2 Сохраняем текущий элемент в x.
 - 3.3 Инициализируем j значением i для поиска позиции вставки.
 - 3.4 Проверяем, если предыдущий элемент больше текущего, то сдвигаем элемент влево.
 - 3.5 Вставляем x в найденную позицию.
 - 3.6 Уменьшаем индекс j.
4. Записываем отсортированные данные в файл:
5. Считаем итоговое время работы программы

Тест	Время выполнения микросек
10 1 8 4 2 3 7 5 6 9 0	0.0010099999999999997
15 1 8 4 2 3 7 5 6 9 0 -10**9 10**9 23123 4532 2	0.0058221

Вывод по задаче:

Программа считывает из файла число n и n чисел, которые надо отсортировать.

Сортировка проходит вставками. Результат записывается в файл. Также программа замеряет и выводит время выполнения сортировки.

input

```
15
```

```
1 8 4 2 3 7 5 6 9 0 -1_000_000_000 1_000_000_000 23123 4532 2
```

output

```
-10000000000 0 1 2 2 3 4 5 6 7 8 9 4532 23123 10000000000
```

```
time: 0.0058251 ms
```

input

```
10
```

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

output

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

```
time: 0.001009999999999997 ms
```

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.

```
import time
def selection_sort(n, array):
    for i in range(n):
        min_ind = i
        for j in range(i + 1, n):
            if array[j] < array[min_ind]:
```

```

        min_ind = j

        array[i], array[min_ind] = array[min_ind],
array[i]
    return array

start = time.perf_counter()
with open('input.txt') as f:
    n = int(f.readline())
    array = list(map(int, f.readline().split()))
    result = selection_sort(n, array)

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

stop = time.perf_counter()
print("time: %s ms" % (stop - start))

```

Текстовое описание решения

1. Импортируем библиотеку time и засекаем время начала.
2. Функция selection_sort() сортирует массив выбором
 - 2.1 Проходим по всем элементам массива for i in range(n):
 - 2.2 Предполагаем, что текущий элемент минимальный
 - 2.3 Ищем индекс минимального элемента в оставшейся части массива
 - 2.4 Обновляем индекс минимального элемента
 - 2.5 Меняем местами текущий элемент и найденный минимальный #
 - 2.6 Возвращаем отсортированный массив
3. Чтение данных из файла
4. Сортируем массив вставками
 - 3.1 Итерируется по индексам массива, начиная с 1 до n-1.
 - 3.2 Сохраняем текущий элемент в x.
 - 3.3 Инициализируем j значением i для поиска позиции вставки.
 - 3.4 Проверяем, если предыдущий элемент больше текущего, то сдвигаем элемент влево.
 - 3.5 Вставляем x в найденную позицию.
 - 3.6 Уменьшаем индекс j.
5. Записываем отсортированные данные в файл:

6. Считаем итоговое время работы программы

Тест	Время выполнения микросек
10 1 8 4 2 3 7 5 6 9 0	0.0005318000000000024
15 1 8 4 2 3 7 5 6 9 0 -10**9 10**9 23123 4532 2	0.0023782999999999964

input

```
15
1 8 4 2 3 7 5 6 9 0 -1_000_000_000 1_000_000_000 23123 4532 2
```

output

```
-10000000000 0 1 2 2 3 4 5 6 7 8 9 4532 23123 10000000000
```

```
time: 0.0005318000000000024 ms
```

input

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

output

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

```
time: 0.0023782999999999964 ms
```

Вывод по задаче:

Программа считывает из файла число n и n чисел, которые надо отсортировать.

Сортировка проходит выбором. Результат записывается в файл. Также программа замеряет и выводит время выполнения сортировки.

Задача №6. Пузырьковая сортировка

6 задача. 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 корректности процедуры вам необходимо доказать, что она завершается и что $A'[1] \leq A'[2] \leq \dots \leq A'[n]$, 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.

```
import time  
def bubble_sort(n, array):  
    for i in range(n):  
        flag = False  
        for j in range(0, n - i - 1):  
            if array[j] > array[j + 1]:  
                array[j], array[j + 1] = array[j +  
1], array[j]  
                flag = True  
        if flag == False:  
            break  
    return array  
  
start = time.perf_counter()  
with open('input.txt') as f:  
    n = int(f.readline())  
    array = list(map(int, f.readline().split()))  
    result = bubble_sort(n, array)  
  
with open('output.txt', 'w') as f:  
    f.write(' '.join(map(str, result)))
```



```
stop = time.perf_counter()
print("time: %s ms" % (stop - start))
```

Текстовое описание решения

1. Импортируем библиотеку time и засекаем время начала.
2. Функция bubble_sort() сортирует массив пузырьком
 - 2.1 Проходим по всем элементам массива
 - 2.2 Флаг для отслеживания изменений
 - 2.3 Проходим по массиву до последнего неотсортированного элемента
 - 2.4 Если текущий элемент больше следующего, меняем их местами
 - Устанавливаем флаг, если произошла смена
 - 2.5 Если не было изменений, массив уже отсортирован
 - 2.6 Возвращаем отсортированный массив
3. Чтение данных из файла
4. вызываем функцию сортировки
5. Записываем отсортированные данные в файл:
6. Считаем итоговое время работы программы

Вывод по задаче:

Программа считывает из файла число n и n чисел, которые надо отсортировать.

Проходит пузырьковая сортировка. Результат записывается в файл. Также программа замеряет и выводит время выполнения сортировки.

Тест	Время выполнения микросек
11 20 16 13 12 11 10 9 8 7 6 5	0.0058353999999999975
15 1 8 4 2 3 7 5 6 9 0 -10**9 10**9 23123 4532 2	0.0045902999999999985

input

```
11
20 16 13 12 11 10 9 8 7 6 5
```

output

```
5 6 7 8 9 10 11 12 13 16 20
```

```
time: 0.005835399999999975 ms
```

input

```
15
1 8 4 2 3 7 5 6 9 0 -1_000_000_000 1_000_000_000 23123 4532 2
```

output

```
-1000000000 0 1 2 2 3 4 5 6 7 8 9 4532 23123 1000000000
```

```
time: 0.004590299999999985 ms
```

Дополнительные задачи

Задача №2. Сортировка вставками+

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

```
import time
start = time.perf_counter()
with open('input.txt') as f:
    ind_list = [1]
    n = int(f.readline())
```

```

array = list(map(int, f.readline().split()))
for i in range(1, n):
    x = array[i]
    j = i

    while j > 0 and array[j - 1] > x:
        array[j] = array[j - 1]
        j -= 1

    ind_list.append(j + 1)
    array[j] = x

with open('output.txt', 'w') as f:
    f.write(' '.join(map(str, ind_list)) + "\n")
    f.write(' '.join(map(str, array)))

stop = time.perf_counter()
print("time: %s ms" % (stop - start))

```

1. Запись времени начала выполнения
2. Создаем список для хранения индексов вставки
3. Читаем данные из файла
4. Сортировка массив вставками
5. Сохраняем текущий элемент $j = i$
6. Инициализируем индекс для поиска позиции вставки
7. Сдвигаем элементы, чтобы найти правильное место для x
8. Вставляем x на правильную позицию. Сохраняем индекс вставки.
9. Записываем индексы и отсортированный массив в файл
10. Вывод времени

Тест	Время выполнения микросек
10 1 8 4 2 3 7 5 6 9 0	0.0031879000000000005

input

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

output

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

```
time: 0.0031879000000000005 ms
```

Вывод по задаче:

Программа считывает из файла число n и n чисел, которые надо отсортировать.

Сортировка проходит выбором, одновременно запоминая индексы вставки. Результат записывается в файл. Также программа замеряет и выводит время выполнения сортировки.

Задача №7. Знакомство с жителями Сортлэнда

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

- Пример:

input.txt	output.txt
5 10.00 8.70 0.01 5.00 3.00	3 4 1

Если отсортировать жителей по их достатку, получится следующий массив:

$$[0.01, 3][3.00, 5][5.00, 4][8.70, 2][10.00, 1]$$

Здесь каждый житель указан в квадратных скобках, первое число — его достаток, второе число — его идентификационный номер. Таким образом, самый бедный житель имеет номер 3, самый богатый — номер 1, а средний — номер 4.

- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.

```

import time
start = time.perf_counter()

with open('input.txt') as f:
    ind_list = []
    n = int(f.readline())
    data = f.readline().split()
    array = []
    for i in range(len(data)):
        array.append([i + 1, float(data[i])])

    for i in range(1, n):
        x = array[i]
        j = i

        while j > 0 and array[j - 1][1] > x[1]:
            array[j] = array[j - 1]
            j -= 1

        array[j] = x
        lenn = len(array)

with open('output.txt', 'w') as f:
    f.write(str(array[0][0]) + " " +
str(array[lenn//2][0]) + " " + str(array[-1][0]))

stop = time.perf_counter()
print("time: %s ms" % (stop - start))

```

Текстовое описание решения

1. Импортируем библиотеку time и засекаем время начала.
2. Чтение данных из файла
3. создание двумерного массива с индексами
4. Сортируем массив вставками запоминая индексы
 - 4.1 Итерируется по индексам массива, начиная с 1 до n-1.
 - 4.2 Сохраняем текущий элемент в x.
 - 4.3 Инициализируем j значением i для поиска позиции вставки.

- 4.4 Проверяем, если предыдущий элемент больше текущего, то сдвигаем элемент влево.
- 4.5 Вставляем x в найденную позицию.
- 4.6 Уменьшаем индекс j.
5. Записываем нулевой, средний и последний элемент в файл
6. Считаем итоговое время работы программы

input

```
5
10.00 8.70 0.01 5.00 3.00
```

output

```
3 4 1
```

```
time: 0.0031879000000000005 ms
```

```
time: 0.0031879000000000005 ms
```

Тест	Время выполнения микросек
5 10.00 8.70 0.01 5.00 3.00	0.0031879000000000005

Вывод по задаче:

Программа считывает из файла число n и n чисел, которые надо отсортировать.

Сортировка проходит выбором, исходные индексы сохраняются. Результат нужных индексов записывается в файл. Также программа замеряет и выводит время выполнения сортировки.

Задача №10. Палиндром

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 искомым палиндром.

```
import time
start = time.perf_counter()
with open('input.txt') as f:
    n = int(f.readline())
    s = f.readline()
    sl = {}
    for el in s:
        if el not in sl:
            sl[el] = 1
        else:
            sl[el] += 1
    answer = ""
    letter = ""
    for k, v in sorted(sl.items()):
        if v % 2 == 0:
            while v > 0:
                answer = answer[:len(answer) // 2] +
k + k + answer[len(answer)//2:]
                v -= 2
            else:
                if letter == "":
                    letter = k
            answer = answer[:len(answer) // 2] + letter +
answer[len(answer)//2:]
with open('output.txt', 'w') as f:
    f.write(answer)
```



```
stop = time.perf_counter()
print("time: %s ms" % (stop - start))
```

текстовое объяснение

1. Импортируется модуль time для измерения времени выполнения.
2. Читаем данные из файла и сохраняем в словарь, подсчитывая количество каждой буквы
3. Составляется палиндрома

3.1 Инициализируются пустая строка answer и пустая строка letter.

3.2 Для каждого символа и его количества из отсортированного словаря:

Если количество символа четное, добавляются пары символов в середину строки answer

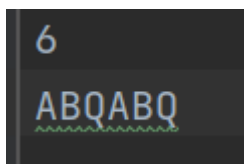
Если количество нечетное и letter еще не задан, запоминается этот символ как letter

3.3 Если найден символ с нечетным количеством, он добавляется в центр строки answer.

4. Записываем результат в файл output

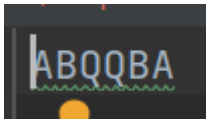
Тест	Время выполнения микросек
6 ABQABQ	0.004969000000000001
6 ABCDEF	0.006438699999999999

input

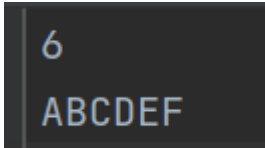


```
6
ABQABQ
```

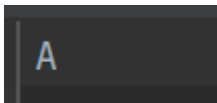
output



input



output



Вывод по задаче

Задача заключается в формировании палиндрома из заданной строки, прочитанной из файла. Код подсчитывает количество каждого символа, добавляет символы с четным количеством в центр палиндрома, а один символ с нечетным количеством помещает в середину. Результат записывается в выходной файл, а также измеряется время выполнения процесса.

Вывод по лабораторной

Изучили различные способы сортировки: вставками, выбором и пузырьком, а также их вариации.