

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:
Кузнецов А.Г.
К3140

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка вставкой	3
Задача №2. Сортировка вставкой +	4
Задача №3. Сортировка вставкой по убыванию	5
Задача №4. Линейный поиск	6
Задача №5. Сортировка выбором	7
Задача №6. Пузырьковая сортировка	8
Вывод	9

Задачи по варианту

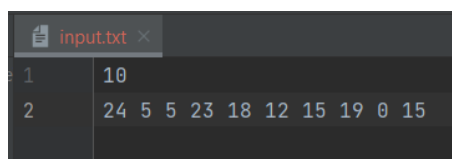
Задача №1. Сортировка вставкой

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

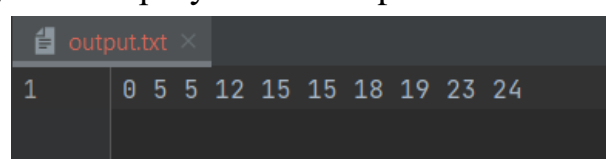
- Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \leq n \leq 10^3$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9 .
- Формат выходного файла (output.txt). Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
- Ограничение по времени. 2сек. • Ограничение по памяти. 256 мб.

```
def Sort(a):
    for i in range(1, n):
        temp = a[i]
        if -10**9 <= a[i] <= 10**9: # проверка числа, чтобы оно не
            превосходило по модулю 10**9
                j = i - 1
                while j >= 0 and temp < a[j]:
                    a[j + 1] = a[j]
                    j -= 1
                a[j + 1] = temp
            else:
                return 'Error'
    return a
with open('input.txt', 'r') as f:
    n=int(f.readline())
    a=f.readline().split()
    aint = list(map(int, a))
if (1 <= n <= 10**3): # проверка первого числа (кол-во чисел), чтобы оно не
    превосходило 10**3
        with open('output.txt', 'w') as f:
            f.write(' '.join(map(str, Sort(aint))))
```

Получаем на вход данные из input.txt и используем их в функции Sort, в которой происходит сортировка вставкой, а именно мы начинаем со второго значения в массиве и проверяем меньше ли оно чем предыдущие значения, если да, то меняем их местами с помощью вставки. После сортировки массива записываем полученный результат в output.txt



1	10
2	24 5 5 23 18 12 15 19 0 15



1	0 5 5 12 15 15 18 19 23 24
---	----------------------------

input.txt	
1	6
2	31 41 59 26 41 58
output.txt	
1	26 31 41 41 58 59

Вывод по задаче: В ходе работы над первой задачей был реализован процесс сортировки вставкой заданного массива

Задача №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.

```
def Sort(a):
    indx[0]=1
    for i in range(1, n):
        temp = a[i]
        if -10**9 <= a[i] <= 10**9: # проверка числа, чтобы оно не
# превосходило по модулю 10**9
            j = i - 1
            while j >= 0 and temp < a[j]:
                a[j + 1] = a[j]
                j -= 1
            a[j + 1] = temp
            indx[i] = (j + 2)
        else:
            return 'Error'
    # Переводим из массивов в строки с отступом в виде пробела
    indx_w = ' '.join(map(str, indx))
    aint_w = ' '.join(map(str, aint))
    return (indx_w + '\n' + aint_w)
with open('input.txt', 'r') as f:
    n=int(f.readline())
    a=f.readline().split()
    aint = list(map(int, a))
indx=[0]*n
if (1 <= n <= 10**3): # проверка первого числа (кол-во чисел), чтобы оно не
# превосходило 10**3
    with open('output.txt', 'w') as f:
        f.write(Sort(aint))
```

Получаем на вход данные из input.txt и используем их в функции Sort, в которой происходит сортировка вставкой, а именно мы начинаем со второго значения в массиве и проверяем меньше ли оно чем предыдущие значение, если да, то меняем их местами с помощью вставки. Дополнительно находим новый индекс элемента массива после обработки. После сортировки массива записываем полученный результат в output.txt

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

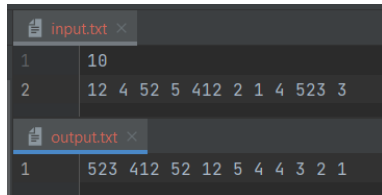
Вывод по задаче: В ходе работы над второй задачей был реализован процесс сортировки вставкой заданного массива и дополнительно был произведен подсчет нового индекса элемента массива после обработки.

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.

```
def Sort(a):
    for i in range(1, n):
        temp = a[i]
        if -10**9 <= a[i] <= 10**9: # проверка числа, чтобы оно не
# превосходило по модулю 10**9
            j = i - 1
            while j >= 0 and temp >= a[j]:
                a[j + 1] = a[j]
                j -= 1
            a[j + 1] = temp
        else:
            return 'Error'
    return a
with open('input.txt', 'r') as f:
    n=int(f.readline())
    a=f.readline().split()
    aint = list(map(int, a))
if (1 <= n <= 10**3): # проверка первого числа (кол-во чисел), чтобы оно не
# превосходило 10**3
    with open('output.txt', 'w') as f:
        f.write(' '.join(map(str, Sort(aint))))
```

Получаем на вход данные из input.txt и используем их в функции Sort, в которой происходит сортировка вставкой, а именно мы начинаем со второго значения в массиве и проверяем больше или равно ли оно чем предыдущие значения, если да, то меняем их местами с помощью вставки. Дополнительно находим новый индекс элемента массива после обработки. После сортировки массива записываем полученный результат в output.txt



input.txt	
1	10
2	12 4 52 5 412 2 1 4 523 3

output.txt	
1	523 412 52 12 5 4 4 3 2 1

Вывод: В ходе работы над третьей задачей был реализован процесс сортировки вставкой заданного массива по убыванию

Задача №4. Линейный поиск

Рассмотрим задачу поиска. • Формат входного файла. Последовательность из n чисел $A = a_1, a_2, \dots, a_n$ 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

```
def Linear_sort(a, v):
    k=0
    for i in range (len(a)):
        if a[i] == v:
            k+=1
            indx.append(i)
    if len(indx)!=0:
        res = ' '.join(map(str, indx))
        return str(k) + "\n" + str(res)

with open('input.txt','r') as f:
    a = list(map(int, f.readline().split()))
    v = f.readline()
indx=[]
if (1 <= len(a) <= 10**3):
    if v!="":
        with open('output.txt', 'w') as f:
```

```

        f.write(str(Linear_sort(a, int(v))))
    else:
        with open('output.txt', 'w') as f:
            f.write(str(0) + "\n" + str(-1))

```

Получаем значения из файла input.txt и значение V, если оно есть. Затем в функции Linear_sort ищем какие значения в списке идентичные с V и записываем их индексы и количество. Если же нет значения V, то выводим количество 0 и индекс -1. Полученные данные записываем в output.txt

input.txt	1	-1 0 31 41 121 59 26 31 -58 15 4 -1 0 31 41
	2	31
output.txt	1	3
	2	2 7 13

input.txt	1	-1 0 31 41 121 59 26 31 -58 15 4 -1 0 31 41
	2	
output.txt	1	0
	2	-1

Вывод: В ходе работы над четвертой задачей был реализован поиск идентичных значений V в списке и вывод их количества и порядкового номера

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.

```

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

```

```

with open('input.txt', 'r') as f:
    a = list(map(int, f.readline().split()))
if 1 <= len(a) <= 10**3:
    with open('output.txt', 'w') as f:
        f.write(' '.join(map(str, selection_sort(a))))

```

Получаем числа из файла input.txt, а затем обрабатываем массив с помощью пузырьковой сортировки в функции selection_sort и в конце работы функции получаем отсортированный список чисел. Далее записываем результат в файл output.txt

input.txt
1 -1 0 31 41 121 59 26 234 -58 15 4

output.txt
1 -58 -1 0 4 15 26 31 41 59 121 234

Сравнение времени выполнения	Сортировка выбором	Сортировка вставкой
Наихудший случай	0.0725135000247974 секунд	0.06722690002061427 секунд
Средний случай	0.06582509999861941 секунд	0.05375660001300275 секунд

Вывод: В ходе работы над пятой задачей был использован метод сортировки выбором, а также было проведено сравнения этого метода с сортировкой вставкой. С помощью полученных результатов, мы убедились, что сортировка выбором хуже, чем сортировка вставкой

Задача №6. Пузырьковая сортировка

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.

```

def bubble_sort(a):
    for i in range(len(a) - 1):
        for j in range(len(a) - i - 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:
    a = list(map(int, f.readline().split()))
if 1 <= len(a) <= 10**3:
    with open('output.txt', 'w') as f:
        f.write(' '.join(map(str, bubble_sort(a))))

```

Получаем числа из файла input.txt, а затем обрабатываем массив с помощью пузырьковой сортировки в функции bubble_sort и в конце работы функции получаем отсортированный список чисел. Далее записываем результат в файл output.txt

input.txt	
1	31 41 59 26 41 58 15
output.txt	
1	15 26 31 41 41 58 59

Сравнение времени выполнения	Пузырьковая сортировка	Сортировка вставкой
Наихудший случай	0.0794123999949079 секунд	0.06722690002061427 секунд
Средний случай	0.0657237000123132 секунд	0.05375660001300275 секунд

Вывод: В ходе работы над шестой задачей был использован метод сортировки пузырьком, а также было проведено сравнения этого метода с сортировкой вставкой. С помощью полученных результатов, мы убедились, что сортировка пузырьком хуже, чем сортировка вставкой

Вывод

В ходе лабораторной работы была проведена работа с такими сортировками как: сортировка вставкой, выбором, пузырьковая, а также был реализован линейный поиск в списке