

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: 7

Выполнил:

Дегтярь Г.С

К3141

Проверил(а):

Афанасьев А. В.

Санкт-Петербург

2024 г.

Содержание отчета

Содержание отчета

Задачи по варианту	2
Задача №1. Сортировка вставкой.	3
Задача №3. Сортировка вставкой по убыванию.	6
Задача №4. Линейный поиск.	7
Задача №5. Сортировка выбором.	9
Задача №6. Пузырьковая сортировка.	12
Задача №7. Знакомство с жителями Сортлэнда.	14
Задача №8. Секретарь Своп.	17
Дополнительные задачи:	
Задача №3.1. Сортировка выбором через рекурсию	19
Вывод	21

Задачи по варианту

Задача №1. Сортировка вставкой.

Текст задачи:

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

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

with open ("input.txt", "w") as f:
    n = input()
    a = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(a))

start = time.perf_counter()

with open ("input.txt", "r") as f:
    n = int(f.readline())
    a = f.readline().split()
    a = [int(x) for x in a]
    for i in range(1, n, 1):
        for j in range(0, i, 1):
            if a[i] < a[i - 1]:
                if a[j] > a[i]:
                    p = a[j]
                    a[j] = a[i]
                    a[i] = p

    for q in range(n - 1):
        if a[q + 1] < a[q]:
            print("error: invalid sort")

end = time.perf_counter()

with open ("output.txt", "w") as f:
    a = [str(x) for x in a]
    f.write(" ".join(a))

print("Время работы: ", end - start, "секунд")

current, peak = tracemalloc.get_traced_memory()

print(f"Пиковая память: {peak / 2**20:.2f} МВ")

tracemalloc.stop()
```

Текстовое объяснение решения:

- Импортируем библиотеки `time` и `tracemalloc` для подсчета времени работы и затраченной памяти. Используя функцию `tracemalloc.start()` мы начинаем подсчет памяти. `with open ("input.txt", "w") as f:` Данный блок будет повторяться в каждой задаче, в нем мы лишь создаем файл `input` и записываем в него наши входные данные. `start = time.perf_counter()` начинаем отсчет времени работы. Далее, считав из файла `input` неотсортированный массив и его длину, мы приступаем к его сортировке. Смысл сортировки вставкой заключается в разбиении массива на 2 части: отсортированную и неотсортированную, а затем подстановке значений из неотсортированной части в отсортированную на подходящее место. С помощью цикла `for i in range(1, n, 1):` мы сравниваем значение предыдущего элемента массива с последующим и если предыдущий элемент больше, то мы попадаем в цикл `for j in range(0, i, 1):`, который ищет первый элемент в отсортированной (левой) части массива, который будет больше текущего элемента `a[i]` и в случае, если находит, меняет его значение с `a[i]`. Таким образом мы проходимся по всему массиву. Далее в цикле `for q in range(n - 1):` мы проверяем действительно ли получился отсортированный по возрастанию массив. Заканчиваем подсчет времени, записываем данные в `output` и выводим значения затраченного времени и памяти.

Результат работы кода на примерах задачи:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\61eb\Лабы\АлгД\algorithms-and-data-structures\lab1\task1\task1.py
5
5 4 3 2 1
Время работы: 0.0002995000000005632 секунд
Пиковая память: 0.04 MB
```

task1.py output.txt ×

1	1	2	3	4	5

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python
7
78 45 9 6 333 4 12
Время работы: 0.0005905999999997746 секунд
Пиковая память: 0.04 MB
```

task1.py output.txt ×

1	4	6	9	12	45	78	333

Результат работы кода на максимальных и минимальных значениях:

```

C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\
1000
82287151 10789640 61160453 3881834 17707232 95270348 34603361 244
Время работы: 0.208851300000000007 секунд
Пиковая память: 0.16 MB

C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\
1
8
Время работы: 0.0004052000000003275 секунд
Пиковая память: 0.04 MB

task1.py output.txt x
1 8

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004284999999999428 секунд	0.04 MB
Пример 1	0.00027679999999996596 секунд	0.04 MB
Пример 2		
Верхняя граница диапазона значений входных данных из текста задачи	0.208851300000000007 секунд	0.16 MB

Вывод о задаче:

Данная задача знакомит нас с методом сортировки вставкой и учит использовать его.

Задача № 3. Сортировка вставкой по убыванию.

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

with open ("input.txt", "w") as f:
    n = input()
    a = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(a))

start = time.perf_counter()

with open ("input.txt", "r") as f:
    n = int(f.readline())
    a = f.readline().split()
    for i in range(1, n, 1):
        for j in range(0, i, 1):
            if a[i] > a[i - 1]:
                if a[j] < a[i]:
                    a[j], a[i] = a[i], a[j]

end = time.perf_counter()

for q in range(n - 1):
    if a[q + 1] > a[q]:
        print("error: invalid sort")

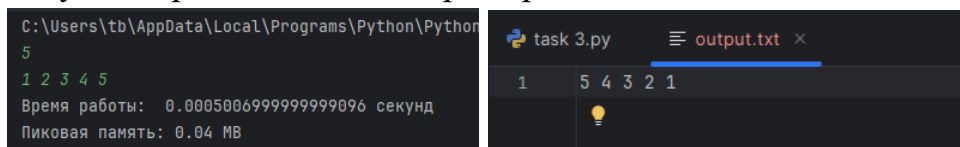
with open ("output.txt", "w") as f:
    f.write(" ".join(a))

print("Время работы: ", end - start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} МВ")
tracemalloc.stop()
```

Текстовое объяснение решения:

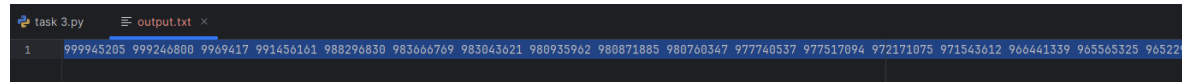
Задача почти полностью аналогична предыдущей. Разница заключается в том, что мы меняем местами члены массива, если предыдущий меньше предыдущего. Также при проверке надо поменять знак.

Результат работы кода на примерах задачи:



Результат работы кода на максимальных и минимальных значениях:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe "C:\Gleb\Л
1000
-356784546 224086878 631790692 583121661 -241719070 -745394775 345998592
Время работы: 0.21148349999999994 секунд
Пиковая память: 0.17 МВ
```



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005006999999999096 секунд	0.04 МВ
Пример 1	0.00027679999999996596 секунд	0.04 МВ
Пример 2		
Верхняя граница диапазона значений входных данных из текста задачи	0.19530899999999995 секунд	0.17 МВ

Вывод о задаче:

Данная задача учит нас адаптировать методы сортировки для конкретных целей и использовать рекурсивные функции для создания алгоритмов сортировки.

Задача № 4. Линейный поиск.

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

with open("input.txt", "w") as f:
    a = input().split()
    n = input().strip()
    f.write(" ".join(a))
    f.write("\n")
    f.write(n)

start = time.perf_counter()

with open("input.txt", "r") as f:
    a = f.readline().split()
    n = f.readline().strip()
    k = 0
```

```

b = []
for i in range(len(a)):
    if a[i] == n:
        k += 1
        b.append(str(i))

end = time.perf_counter()

with open("output.txt", "w") as f:
    if k == 0:
        f.write("-1")
    elif k == 1:
        f.write(b[0])
    elif k > 1:
        f.write(str(k))
        f.write("\n")
        f.write(", ".join(b))

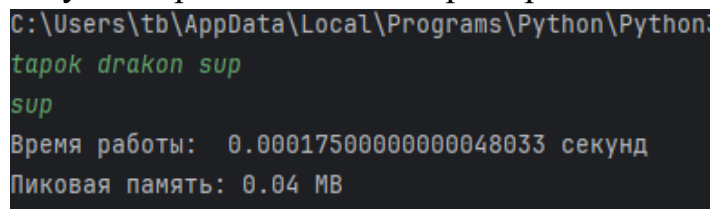
print("Время работы: ", end - start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} МВ")
tracemalloc.stop()

```

Текстовое объяснение решения:

Перед решением задачи необходимо избавиться от лишних символов, которые могут возникнуть во время исполнения метода `readline()` или `input()`, поэтому при изменении переменной `n`, добавляем функцию `strip()`, которая убирает все лишнее из нашей строки. Решение данной задачи заключается в том, что мы проходим по всему заданному списку и находим количество вхождений нужного нам слова, а также сохраняем индексы тех элементов массива, которые соответствуют нужному нам результату. Далее прописываем частные случаи вывода и выводим результат.

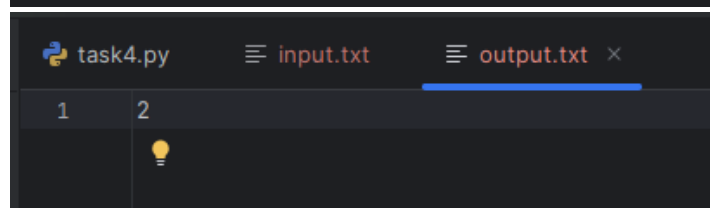
Результат работы кода на примерах из текста задачи



```

C:\Users\tb\AppData\Local\Programs\Python\Python3
tapok drakon sup
sup
Время работы: 0.00017500000000048033 секунд
Пиковая память: 0.04 МВ

```

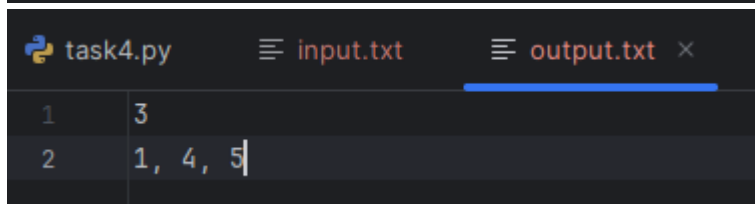


```

task4.py  input.txt  output.txt x
1 2
  
```



```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\6le
Корова Свинья Лошадь Корова Свинья Свинья Таракан
Свинья
Время работы: 0.0006680000000010011 секунд
Пиковая память: 0.04 МВ
```



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005006999999999096 секунд	0.04 МВ
Пример 1	0.0006680000000010011 секунд	0.04 МВ
Пример 2		
Верхняя граница диапазона значений входных данных из текста задачи		

Вывод о задаче:

Данная задача учит нас применять методы линейного поиска на практике.

Задача № 5. Сортировка выбором.

Текст задачи:

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:

```
import time
import tracemalloc

tracemalloc.start()
```

```

with open ("input.txt", "w") as f:
    n = input()
    a = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(a))

start = time.perf_counter()

with open ("input.txt", "r") as f:
    n = int(f.readline())
    a = f.readline().split()
    a = [int(x) for x in a]
    for i in range(0, n - 1, 1):
        minimal = 10 ** 9
        for j in range (i, n, 1):
            if a[j] < minimal:
                minimal = a[j]
                min_ind = j
        a[i], a[min_ind] = a[min_ind], a[i]

end = time.perf_counter()

for q in range(n - 1):
    if a[q + 1] < a[q]:
        print("error: invalid sort")

with open ("output.txt", "w") as f:
    a = [str(x) for x in a]
    f.write(" ".join(a))

print("Время работы: ", end - start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} МВ")
tracemalloc.stop()

```

Текстовое объяснение решения:

Алгоритм сортировки выбором похож на алгоритм сортировки вставкой тем, что у нас так же есть отсортированная и неотсортированная части. Однако при данном методе мы заменяем число минимальным (либо максимальным) числом из неотсортированной части.

Результат работы кода на примерах из текста задачи:

```

C:\Users\tb\AppData\Local\Programs\Python\Python39\p
5
5 4 3 2 1
Время работы: 0.00019339999999967716 секунд
Пиковая память: 0.04 МВ

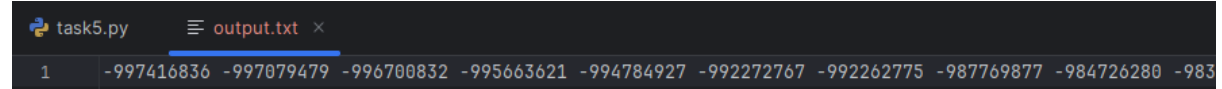
```

task5.py output.txt ×

1 1 2 3 4 5

Результат работы кода на максимальных и минимальных значениях:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\6leb\Лабы\АиСД\
1000
-356784546 224086878 631790692 583121661 -241719070 -745394775 345998592 -646450
Время работы: 0.11085260000000119 секунд
Пиковая память: 0.17 MB
```



The screenshot shows a Python IDE with a file named 'task5.py' and an 'output.txt' window. The output.txt window displays a list of 10 numbers: 1, -997416836, -997079479, -996700832, -995663621, -994784927, -992272767, -992262775, -987769877, -984726280, -983. The console output shows the execution of the code, resulting in the same list of numbers, followed by the execution time (0.11085260000000119 seconds) and peak memory usage (0.17 MB).

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00018129999999993984 секунд	0.04 MB
Пример 1	0. 0.00027759999999995449	0.04 MB
Пример 2		
Верхняя граница диапазона значений входных данных из текста задачи	0.11085260000000119 секунд	0.17 MB

Вывод по задаче: Данная задача знакомит нас с методом сортировки выбором и учит использовать его.

Задание № 6. Пузырьковая сортировка.

Текст задачи:

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.

Листинг кода:

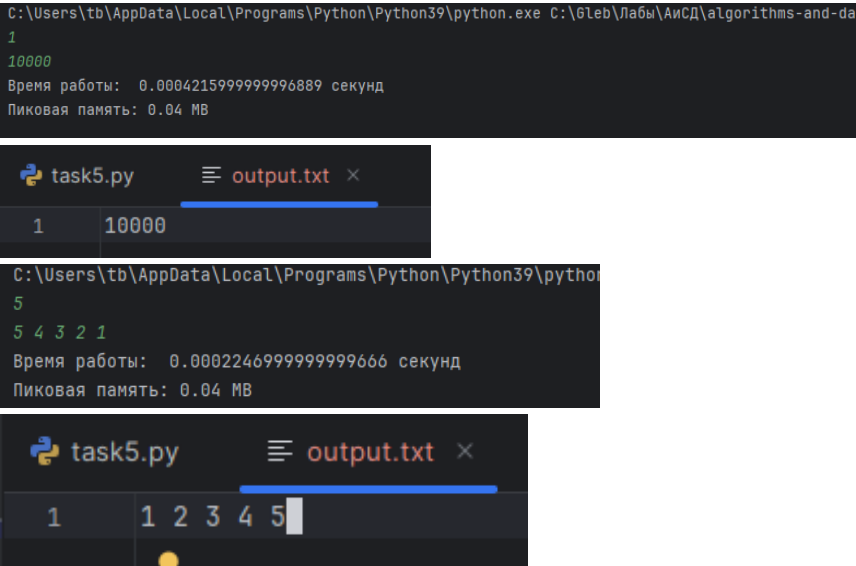
```
import time  
import tracemalloc  
  
tracemalloc.start()  
  
with open ("input.txt", "w") as f:  
    n = input()  
    a = input().split()  
    f.write(n)  
    f.write("\n")  
    f.write(" ".join(a))  
  
start = time.perf_counter()  
  
with open ("input.txt", "r") as f:  
    n = int(f.readline())  
    a = f.readline().split()  
    a = [int(x) for x in a]  
    for i in range(0, n, 1):  
        for j in range(0, n - 1, 1):  
            if a[j] > a[j + 1]:  
                a[j], a[j + 1] = a[j + 1], a[j]  
  
end = time.perf_counter()  
  
for q in range(n - 1):  
    if a[q + 1] < a[q]:  
        print("error: invalid sort")  
  
with open ("output.txt", "w") as f:
```

```
a = [str(x) for x in a]
f.write(" ".join(a))

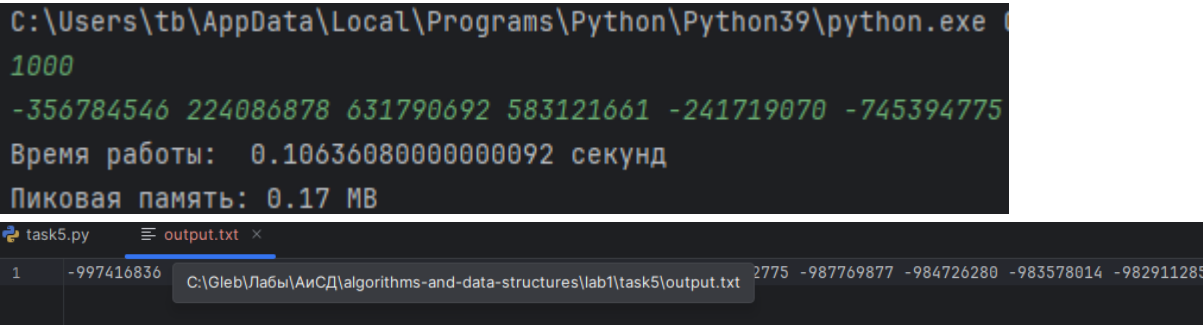
print("Время работы: ", end = start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} MB")
tracemalloc.stop()
```

Текстовое объяснение решения:
Данная сортировка крайне неэффективна. В ней сортировка массива происходит за счет постоянной замены каждого числа, которое больше своего соседа справа (при сортировке по возрастанию) с этим самым соседом. С изменением шага цикла, мы бы так добрали до конца массива, постоянно меняя числа местами, а затем начали то же самое сначала n раз.

Результат работы кода на примерах задачи:



Результаты работы кода на max:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из	0.00042159999999996889 секунд	0.04 MB

текста задачи		
Пример из задачи	0.000493099999999913 секунд	0,04 MB
Пример из задачи		
Верхняя граница диапазона значений входных данных из текста задачи	0.106360800000000092 секунд	0.17 MB

Вывод по задаче:

Данная задача знакомит нас с пузырьковым методом сортировки данных и учит использовать его.

Задача № 7. Знакомство с жителями Сортлэнда.

Текст задачи:

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.

Листинг кода:

```
from time import *
import tracemalloc

tracemalloc.start()

with open ("input.txt", "w") as f:
    n = input()
    a = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(a))

start = perf_counter()

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

```

n = int(f.readline())
a = f.readline().split()
a = [float(x) for x in a]
b = a.copy()
for i in range(1, n, 1):
    for j in range(0, i, 1):
        if a[i] < a[i - 1]:
            if a[j] > a[i]:
                p = a[j]
                a[j] = a[i]
                a[i] = p

end = perf_counter()

for q in range(n - 1):
    if a[q + 1] < a[q]:
        print("error: invalid sort")

with open("output.txt", "w") as f:
    f.write(str(b.index(a[0]) + 1) + " " + str(b.index(a[n // 2]) + 1) +
            " " + str(b.index(a[-1]) + 1))
print("Время работы: ", end - start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} MB")
tracemalloc.stop()

```

Текстовое объяснение решения:

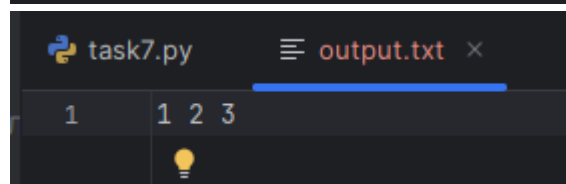
В данной задаче нам надо вывести самое маленькое, среднее и самое большое числа массива. Для сортировки я использую алгоритм сортировки вставкой из 1го задания. Так как в задаче необходимо вывести номера этих людей (чисел) относительно изначального (неотсортированного) списка, мы скопируем изначальный список в переменную `b` и будем выводить индексы относительно нее. Также раз нас просят вывести не индексы, а номера самого бедного, среднего и богатого человека, найдя индекс нужного числа, мы будем увеличивать его на 1, получая тем самым номер нужного человека.

Результат работы кода на примерах из текста задачи:

```

C:\Users\tb\AppData\Local\Programs\Python\Pyt
3
2 4 5
Время работы: 0.00020549999999985857 секунд
Пиковая память: 0.04 MB

```



```
C:\Users\tb\AppData\Local\Programs\Python\Python
6
3 2 7 3 0 5
Время работы: 0.00046429999999997307 секунд
Пиковая память: 0.04 MB
```

```
task7.py  output.txt x
1 5 1 3
```

Результат работы кода на максимальных и минимальных значениях:

Max:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\6leb\Лабы\
9999
29236876 48336239 58543273 96836410 65420857 93255310 78291475 56514094 211
Время работы: 23.1053507000000002 секунд
Пиковая память: 1.37 MB
```

```
task7.py  output.txt x
1 102 1357 1074
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004224999999999923 секунд	0.04 MB
Пример из задачи	0.00035030000000000256 секунд	0.04 MB
Пример из задачи		
Верхняя граница диапазона значений входных данных из текста задачи	23.1053507000000002 секунд	1.37 MB

Вывод о задаче: Данная задача хорошо закрепляет основные методы работы с сортировками в python и учит применять их на практике.

Задание № 8. Секретарь Своп:

Текст задачи:

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.

Листинг кода:

```
from time import *
import tracemalloc

tracemalloc.start()

with open ("input.txt", "w") as f:
    n = input()
    a = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(a))

start = perf_counter()

with open ("input.txt", "r") as f:
    with open ("output.txt", "w") as g:
        n = int(f.readline())
        b = f.readline().split()
        a = [int(x) for x in b]
        for i in range( n - 1):
            m = i
            for j in range(i + 1, n):
                if a[m] > a[j]:
                    m = j
            if m != i:
                p = min(m,i) + 1
                v = max(m,i) + 1
                g.write("Swap elements at indices " + str(p) + " and "
+ str(v) + ".")
                g.write("\n")
                a[m], a[i] = a[i], a[m]
            g.write("No more swaps needed.")

end = perf_counter()

for q in range(n - 1):
    if a[q + 1] < a[q]:
        print("error: invalid sort")

print("Время работы: ", end - start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} МВ")
tracemalloc.stop()
```

Текстовое объяснение:

Данный алгоритм, подобно пузырьковой сортировке меняет соседние элементы массива, если они находятся в неправильном порядке, однако лишь до момента, когда они не окажутся на своей правильной позиции.

Результат работы:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python
5
3 1 4 2 2
Время работы: 0.0005150000000000432 секунд
Пиковая память: 0.05 MB
```

```
task7.py task8.py output.txt x
1 Swap elements at indices 1 and 2.
2 Swap elements at indices 2 and 4.
3 Swap elements at indices 3 and 5.
4 No more swaps needed.
```

На максимальных значениях:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\py
5000
29236876 48336239 58543273 96836410 65420857 93255316
Время работы: 2.9171544000000001 секунд
Пиковая память: 0.84 MB
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005543999999999549 секунд	0.05 MB
Пример из задачи	0.00062609999999998518 секунд	0.05 MB
Пример из задачи		
Верхняя граница диапазона значений входных данных из текста задачи	2.9171544000000001 секунд	0.84 MB

Вывод о задаче:

Данная задача очень интересна тем, что для ее решения необходимо пойти дальше стандартного алгоритма сортировки. Подобный подход развивает адаптивные способности программиста.

Дополнительные Задачи:

Задача №3.1. Сортировка вставкой через рекурсию.

Текст задачи:

Подумайте, можно ли переписать алгоритм сортировки вставкой с использованием рекурсии?

Листинг задачи:

```
import time
from time import perf_counter
import tracemalloc

tracemalloc.start()

with open ("input.txt", "w") as f:
    n1 = input()
    a1 = input().split()
    f.write(n1)
    f.write("\n")
    f.write(" ".join(a1))

start = time.perf_counter()

with open ("input.txt", "r") as f:
    n = int(f.readline())
    arr = f.readline().split()
    arr = [int(x) for x in arr]

    def recursive_insertion_sort(n, arr):
        if n == 1:
            return arr

        arr = recursive_insertion_sort(n - 1, arr)
        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

    a = recursive_insertion_sort(n, arr)
    for q in range(n - 1):
        if a[q + 1] < a[q]:
            print("error: invalid sort")
    a = [str(x) for x in a]

end = perf_counter()

with open ("output.txt", "w") as f:
    f.write(" ".join(a))

print("Время работы: ", end - start, "секунд")
```

```
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} MB")
tracemalloc.stop()
```

Текстовое объяснение:

Это рекурсивный вариант сортировки вставкой. В функцию подаем два значения: кол-во элементов в массиве, который надо отсортировать и сам массив. Закручиваем нашу рекурсию повторным вызовом с новым значением $n = n - 1$. Благодаря данному действию после того, как n достигнет значения 1, функции начнут вызывать друг друга в обратном порядке и мы получим поочередный вызов функций со значениями $n = 1, 2, 3 \dots 7$ и новыми массивами. Затем мы пробегаемся по самому массиву и сортируем его по принципу сортировки вставкой.

Примеры работы программы на максимальных и минимальных числах:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\py
1
5
Время работы: 0.0003883000000000081 секунд
Пиковая память: 0.04 MB
```

```
task3_recursive_insertion_sort.py  output.txt x
1 5
💡
```

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\pyth
994
82287151 10789640 61160453 3881834 17707232 95270348 34
Время работы: 0.8584670999999999 секунд
Пиковая память: 0.57 MB
```

```
task3_recursive_insertion_sort.py  output.txt x
1 170820 207476 282078 373781 604227 605693 617620 778652 858140 920123 967002 1032380 1128074 1239659 1241246 1241481 130605
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0003883000000000081 секунд	0.04 MB

Пример из задачи	0.00032340000000008473 секунд	0.04 МВ
Пример из задачи		
Верхняя граница диапазона значений входных данных из текста задачи	0.8584670999999999 секунд	0.57 МВ

Вывод о задаче:

Данная задача наглядно показывает принципы работы рекурсивных функций и хорошо помогает разобраться в них.

Вывод о лабораторной работе:

Данная лабораторная работа направлена на изучение классических методов сортировки данных в python, тестирования программы, рекурсивных алгоритмов. Данная работа помогла мне освоить перечисленные навыки и вспомнить основные методы работы с языком программирования python.