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

K3141

Проверил(а):

Афанасьев А. В.

Санкт-Петербург

2024 г.

# Содержание отчета

[**Содержание отчета**](#_heading=h.gjdgxs)

[**Задачи по варианту**](#_heading=h.30j0zll) **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**](#_heading=h.2et92p0)21

# 

# Задачи по варианту

## Задача №1. Сортировка вставкой.

Текст задачи:

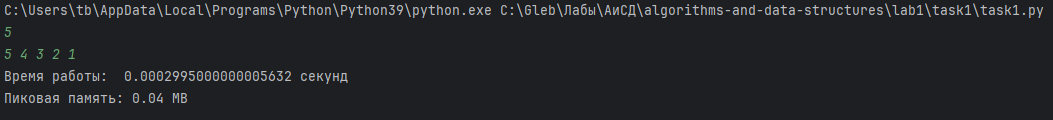


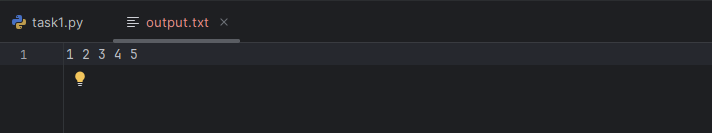
Листинг кода:

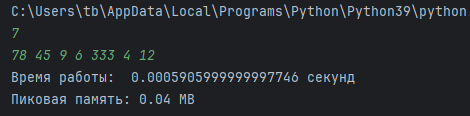
|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  a **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**a**))**  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} MB"**)**  tracemalloc**.***stop***()** |

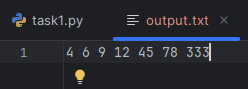
**Текстовое объяснение решения:**

* Импортируем библиотеки time и tracemalloc для подсчета времени работы и затраченной памяти. Использовав 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 а него наши входные данные. start = time.perf\_counter() начинаем отсчет времени работы. Далее, считав из файла input неотсортированный массив и его длинну, мы приступаем к его сортировке. Смысл сортировки вставкой заключается в разбитии 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 ц нас получился отсортированный по возрастанию массив. Заканчиваем подсчет времени, записываем данные в output и выводим значения затраченного времени и памяти.

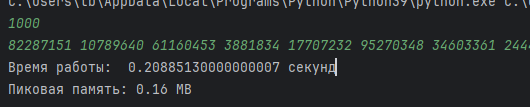
Результат работы кода на примерах задачи:  


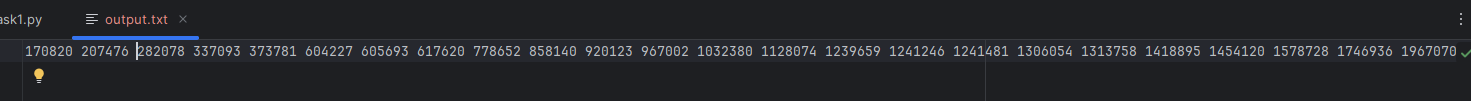


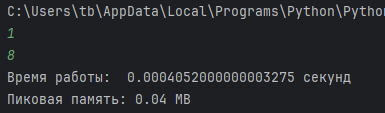


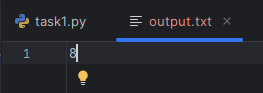


Результат работы кода на максимальных и минимальных значениях:









|  |  |  |
| --- | --- | --- |
|  | 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 | 0.20885130000000007 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.**

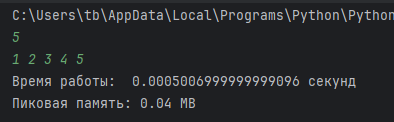
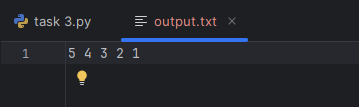
Листинг кода:

|  |
| --- |
| **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} MB"**)**  tracemalloc**.***stop***()** |

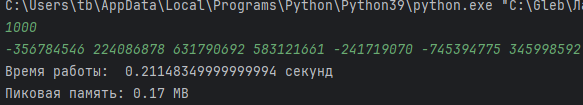
Текстовое объяснение решения:

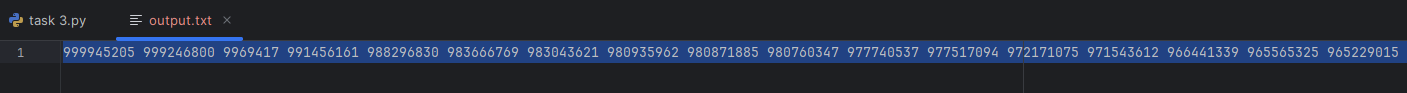
Задача почти полностью аналогична предыдущей. Разница заключается в том, что мы меняем местами члены массива, если предыдущий меньше предыдущего. Также при проверке надо поменять знак.

Результат работы кода на примерах задачи:

Результат работы кода на максимальных и минимальных значениях:





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

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.**

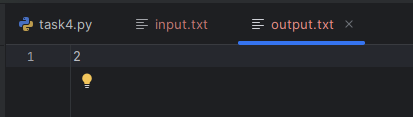
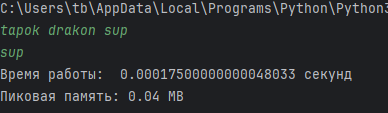
Листинг кода:

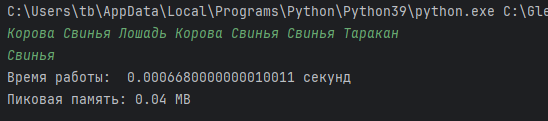
|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open(**"input.txt"**,** "w"**)** **as** f**:**  a **=** **input().***split***()**  n **=** **input().***strip***()**  f**.***write***(**" "**.***join***(**a**))**  f**.***write***(**"\n"**)**  f**.***write***(**n**)**  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} MB"**)**  tracemalloc**.***stop()* |

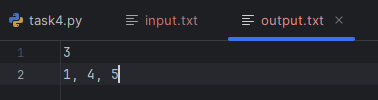
Текстовое объяснение решения:

Перед решением задачи необходимо избавить 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 списки 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







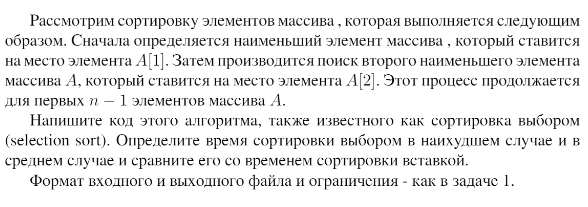
|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0005006999999999096 секунд | 0.04 MB |
| Пример 1 | 0.0006680000000010011 секунд | 0.04 MB |
| Пример 2 |  |  |
| Верхняя граница диапазона значений входных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

Вывод о задаче:

Данная задача учит нас применять методы линейного поиска на практике.

**Задача № 5. Сортировка выбором.**

Текст задачи:

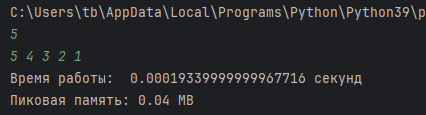


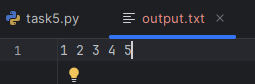
Листинг кода:

|  |
| --- |
| **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} MB"**)**  tracemalloc**.***stop***()** |

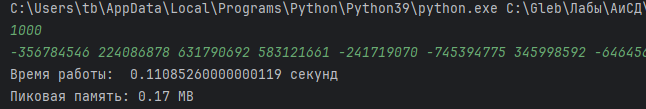
Текстовое объяснение решения:  
Алгоритм сортировки выбором похож на алгоритм сортировки вставкой тем, что у нас так же есть отсортированная и неотсортированная части. Однако при данном методе мы заменяем число минимальным (либо максимальным) числом из неотсортированной части.

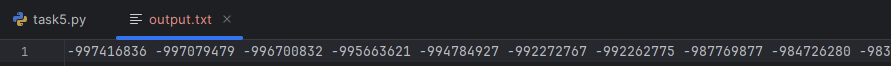
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:

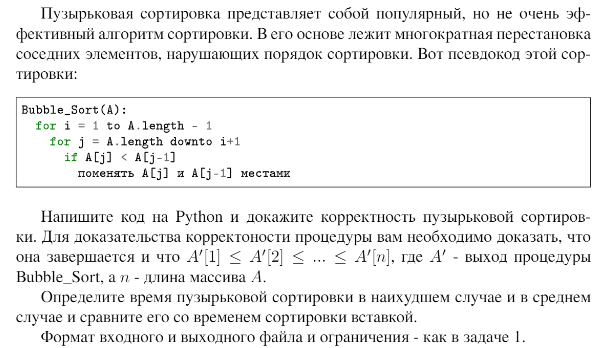




|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00018129999999993984 секунд | 0.04 MB |
| Пример 1 | 0. 0.0002775999999995449 | 0.04 MB |
| Пример 2 |  |  |
| Верхняя граница диапазона значений входных данных 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:  
****

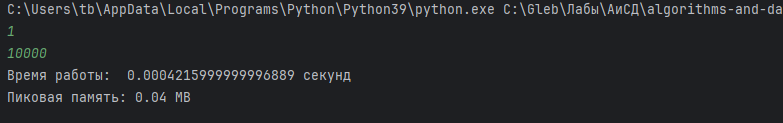
Листинг кода:

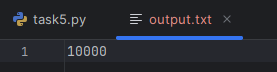
|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  a **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**a**))**  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***()** |

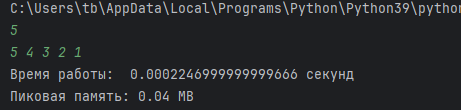
Текстовое объяснение решения:

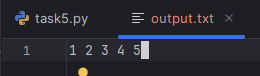
Данная сортировка крайне неэффективна. В ней сортировка массива происходит за счет постоянной замены каждого числа, которое больше своего соседа справа (при сортировке по возрастанию) с этим самым соседом. С изменением шага цикла, мы бы так добрали до конца массива, постоянно меняя числа местами, а затем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

Результат работы кода на примерах задачи:

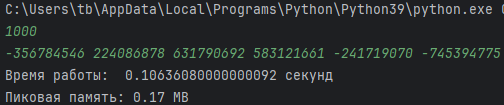


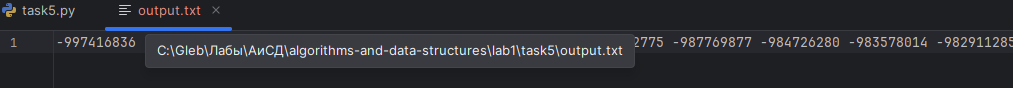






Результаты работы кода на max:

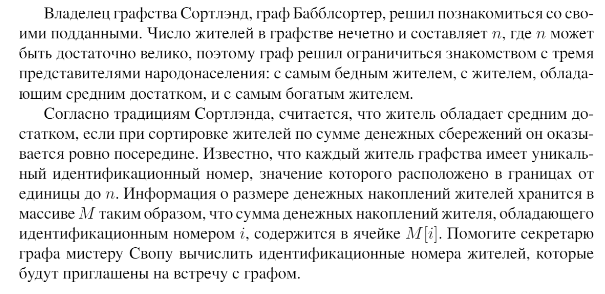




|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0004215999999996889 секунд | 0.04 MB |
| Пример из задачи | 0.000493099999999913 секунд | 0,04 MB |
| Пример из задачи |  |  |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.10636080000000092 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:  


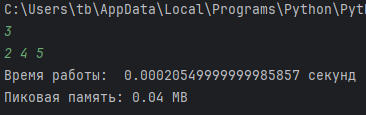
Листинг кода:

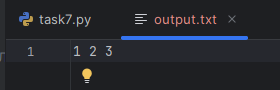
|  |
| --- |
| **from** time **import** **\***  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  a **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**a**))**  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***()** |

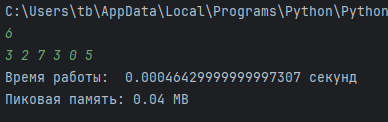
Текстовое объяснение решения:

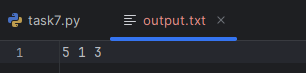
В данной задаче нам надо вывести самое маленькое, среднее и самое большое числа массива. Для сортировки я использую алгоритм сортировки вставкой из 1го задания. Так как в задаче необходимо вывести номера этих людей (чисел) относительно изначального (неотсортированного) списка, мы скопируем изначальный 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:



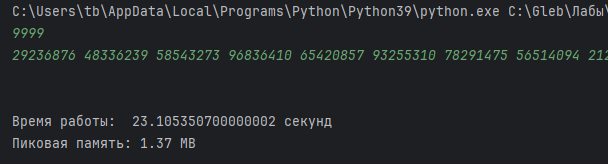


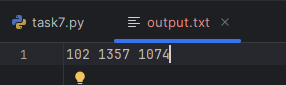




Результат работы кода на максимальных и минимальных значениях:

Max:



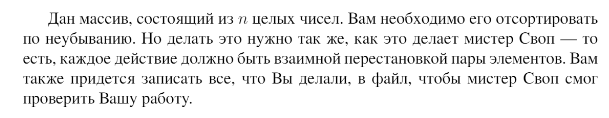


|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0004224999999999923 секунд | 0.04 MB |
| Пример из задачи | 0.0003503000000000256 секунд | 0.04 MB |
| Пример из задачи |  |  |
| Верхняя граница диапазона значений входных данных из текста задачи | 23.105350700000002 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:



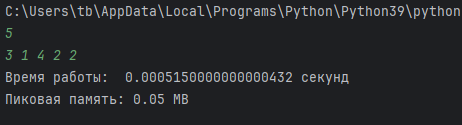
Листинг кода:

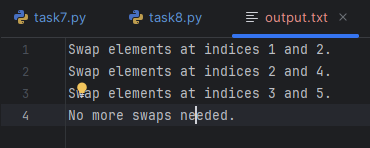
|  |
| --- |
| **from** time **import** **\***  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  a **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**a**))**  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} MB"**)**  tracemalloc**.***stop***()** |

Текстовое объяснение:

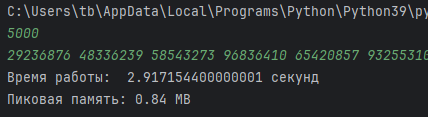
Данный алгоритм, подобно пузырьковой сортировке меняет соседние элементы массива, если они находятся в неправильном порядке, однако лишь до момента, когда они не окажутся на своей правильной позиции.

Результат работы:





На максимальных значениях:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0005543999999999549 секунд | 0.05 MB |
| Пример из задачи | 0.0006260999999998518 секунд | 0.05 MB |
| Пример из задачи |  |  |
| Верхняя граница диапазона значений входных данных из текста задачи | 2.917154400000001 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:



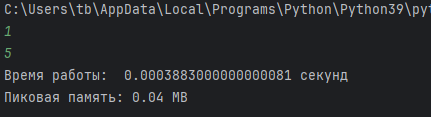
Листинг задачи:

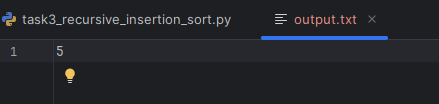
|  |
| --- |
| **import** time  **from** time **import** perf\_counter  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n1 **=** **input()**  a1 **=** **input().***split***()**  f**.***write***(**n1**)**  f**.***write***(**"\n"**)**  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***()** |

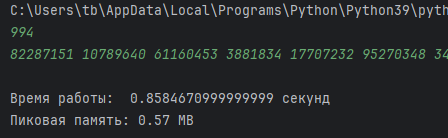
Текстовое объяснение:

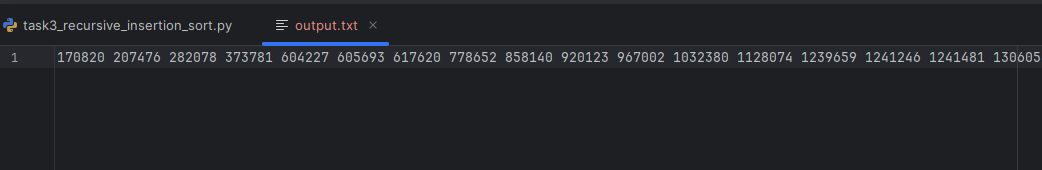
Это рекурсивный вариант сортировки вставкой. В функцию подаем два значения: кол-во элементов в массиве, который надо отсортировать и сам массив. Закручиваем нашу рекурсию повторным вызовом с новым значением n = n – 1. Благодаря данному действию после того, как n достигнет значения 1, функции начнут вызывать друг друга в обратном порядке и мы получим поочередный вызов функций со значениями n = 1, 2, 3 … 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:









|  |  |  |
| --- | --- | --- |
|  | 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 MB |
| Пример из задачи |  |  |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.8584670999999999 секунд | 0.57 MB |

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.