САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе № 2

по курсу «Алгоритмы и структуры данных»

Тема: Сортировка слиянием. Метод декомпозиции

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 слиянием

Задача №2. Сортировка слиянием+ 6

Задача №9. Метод Штрассена для умножения матриц 17

**Дополнительные задачи:**

Задача №3. Число инверсий 17

Задача №4. Бинарный поиск 19

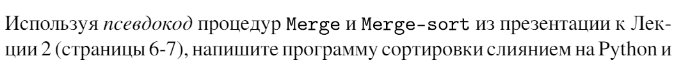
[**Вывод**](#_heading=h.2et92p0)21

# 

# Задачи по варианту

## Задача №1. Сортировка слиянием.

Текст задачи:



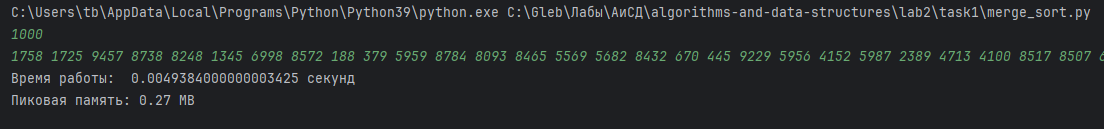


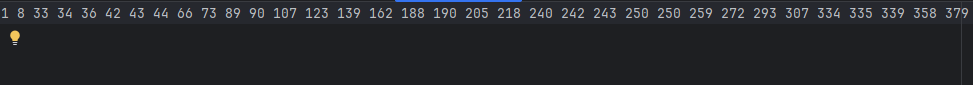
Листинг кода:

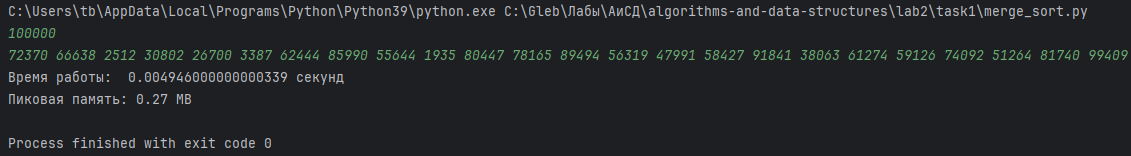
|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  a **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**a**))**  start **=** time**.***perf\_counter***()**  **def** merge**(**L**,** R**):**  res **=** **[]**  i**,** j **=** 0**,** 0  n**,** m **=** **len(**L**),** **len(**R**)**  **while** i **<** n **and** j **<** m**:**  **if** L**[**i**]** **<** R**[**j**]:**  res**.***append***(**L**[**i**])**  i **+=** 1  **else:**  res**.***append***(**R**[**j**])**  j **+=** 1  **while** j **<** m**:**  res**.***append***(**R**[**j**])**  j **+=** 1  **while** i **<** n**:**  res**.***append***(**L**[**i**])**  i **+=** 1  **return** res  **def** merge\_sort**(**A**):**  **if** **len(**A**)** **<=** 1**:**  **return** A  q **=** **len(**A**)** **//** 2  left **=** merge\_sort**(**A**[:**q**])**  right **=** merge\_sort**(**A**[**q**:])**  **return** merge**(**left**,** right**)**  **with** **open(**"input.txt"**,** "r"**)** **as** f**:**  n **=** **int(**f**.***readline***())**  A1 **=** f**.***readline***().***split***()**  A **=** **[int(**x**)** **for** x **in** A1**]**  res **=** merge\_sort**(**A**)**  **with** **open(**"output.txt"**,** "w"**)** **as** f**:**  res **=** **[str(**x**)** **for** x **in** res**]**  s **=** " "**.***join***(**res**)**  f**.***write***(**s**)**  end **=** time**.***perf\_counter***()**  **print(**"Время работы: "**,** end **-** start**,** "секунд"**)**  current**,** peak **=** tracemalloc**.***get\_traced\_memory***()**  **print(**f"Пиковая память: {peak **/** 2**\*\***20:.2f} MB"**)**  tracemalloc**.***stop***()** |

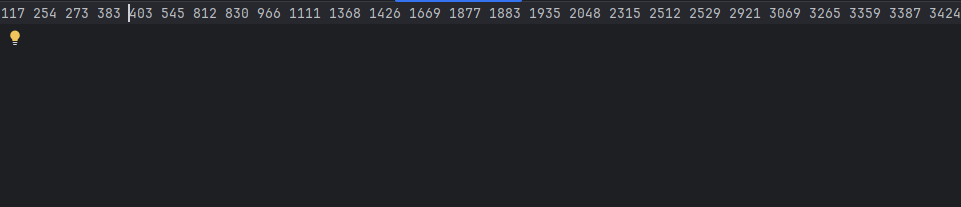
**Текстовое объяснение решения:**

* Сначала разберем принцип работы функции merge: ей на вход подаются 2 части массива, которые неободимо сравнить и объединить в 1 массив в правильном порядке. Затем мы инициализируем вспомогательный массив для хранения будущего результата (res = []), указатели (i, j = 0, 0) и находим длинны подмассиво (n, m = len(L), len(R)). Затем в цикле while пока мы не дойдем до конца хотя бы одного подмассива, мы будем сравнивать их элементы и записывать подходящий в новый массив res, после нахождения подходящего элемента, указатель в этом массиве увеличивается на 1. Затем в еще двух циклов while мы проверяем прошли ли указатели каждый подмассив полностью и если нет – мы дописываем элементы из недопройденного массива в результирующий массив без проверки т.к каждый подмассив уже отсортирован. Возвращаем res. Функцию merge вызывает функция mergt\_sort, давайте разберем ее: Сначла устанавливаем, что условием выхода из рекурсии будет то, что массив, который полуает функция будет содержать 1 элемент. Затем находим индекс середины массива через целочисленное деление. Затем рекурсивно вызываем функцию left = merge\_sort(A[:q]), подавая в аргументе только левую часть массива, а затем сразу же выполняем тот же алгоритм для правой части: right = merge\_sort(A[q:]). Таким образом после достижения максимальной глубины рекурсии, нам останется лишь объединить все подмассивы которые мы получили: return merge(left, right)

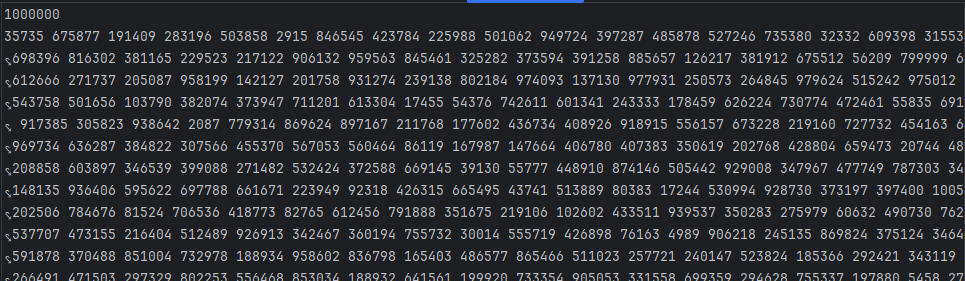
Результат работы кода на примерах задачи:  


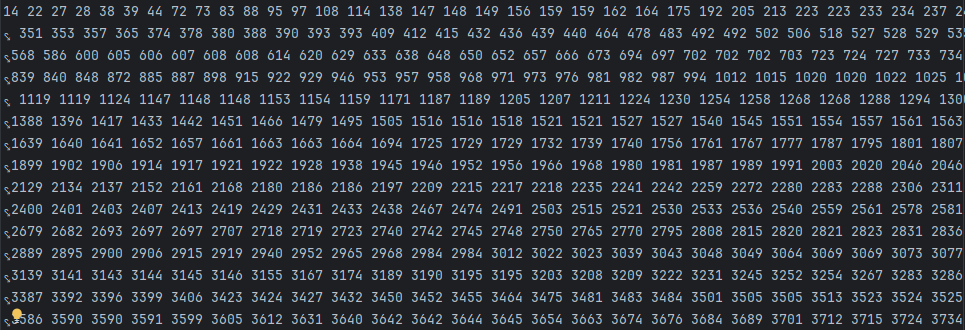


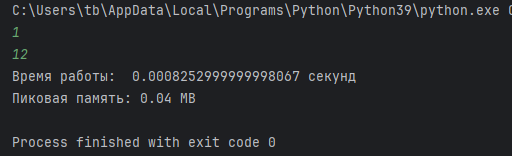


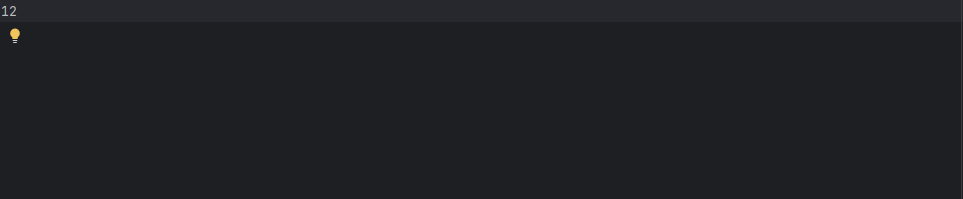


Результат работы кода на максимальных и минимальных значениях:









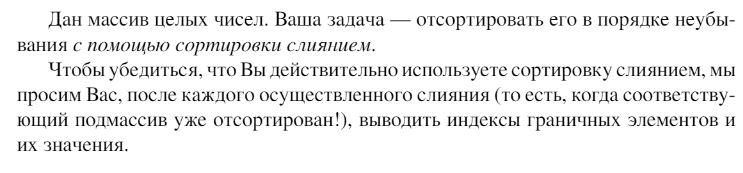
|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0008252999999998067 секунд | 0.04 MB |
| Пример 1 | 0.00027679999999996596 секунд | 0.04 MB |
| Пример 2 | 0.00026571257242557242 секунд | 0.04 MB |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.9125724999999996 секунд | 34.92 MB |

Вывод о задаче:

Данная задача знакомит нас с относительно эффективным алгоритмом сортировки.

**Задача № 2. Сортировка слиянием +**

Текст задачи:



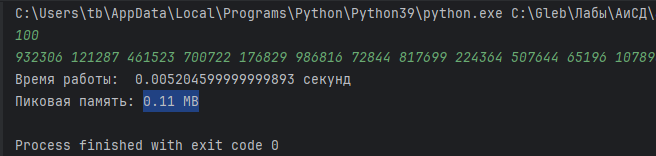
Листинг кода:

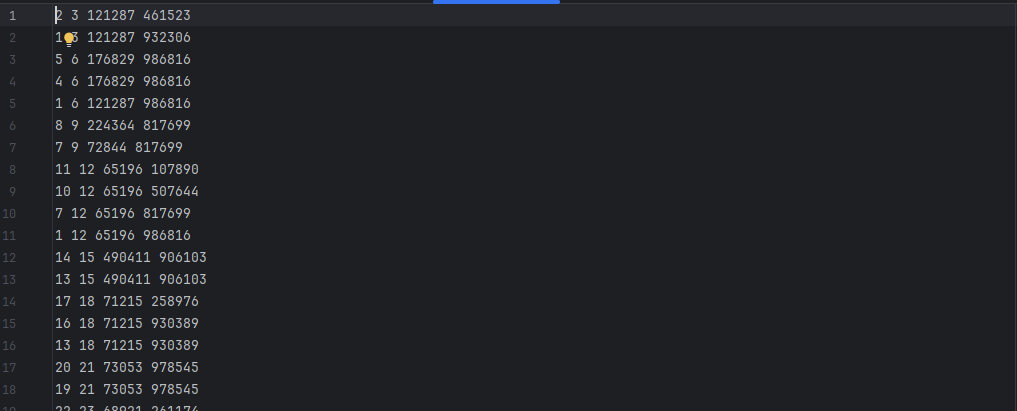
|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  a **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**a**))**  start **=** time**.***perf\_counter***()**  **def** merge\_sort**(**arr**):**  **if** **len(**arr**)** **<=** 1**:**  **return** arr  mid **=** **len(**arr**)** **//** 2  left\_half **=** merge\_sort**(dict(list(**arr**.***items***())[:**mid**]))**  right\_half **=** merge\_sort**(dict(list(**arr**.***items***())[**mid**:]))**  **return** merge**(**left\_half**,** right\_half**)**  **def** merge**(**left**,** right**):**  sorted\_array **=** **[]**  i **=** j **=** 0  l1 **=** **min(list(**left**.***keys***()))**  r1 **=** **max(list(**right**.***keys***()))**  **while** i **<** **len(**left**)** **and** j **<** **len(**right**):**  **if** **list(**left**.***values***())[**i**]** **<** **list(**right**.***values***())[**j**]:**  sorted\_array**.***append***([list(**left**.***keys***())[**i**],** **list(**left**.***values***())[**i**]])**  i **+=** 1  **else:**  sorted\_array**.***append***([list(**right**.***keys***())[**j**],** **list(**right**.***values***())[**j**]])**  j **+=** 1  **while** i **<** **len(**left**):**  sorted\_array**.***append***([list(**left**.***keys***())[**i**],** **list(**left**.***values***())[**i**]])**  i **+=** 1  **while** j **<** **len(**right**):**  sorted\_array**.***append***([list(**right**.***keys***())[**j**],** **list(**right**.***values***())[**j**]])**  j **+=** 1  f**.***write***(str(**l1**)** **+** ' ' **+** **str(**r1**)** **+** ' ' **+** **str(**sorted\_array**[**0**][**1**])** **+** ' ' **+** **str(**sorted\_array**[-**1**][**1**]))**  f**.***write***(**"\n"**)**  **return** **dict(**sorted\_array**)**  **with** **open(**"input.txt"**,** "r"**)** **as** f**:**  n **=** **int(**f**.***readline***())**  arr1 **=** f**.***readline***().***split***()**  arr **=** **[int(**x**)** **for** x **in** arr1**]**  **with** **open(**"output.txt"**,** "w"**)** **as** f**:**  inp\_arr **=** **[]**  **for** i **in** **range(**1**,** **len(**arr**)+**1**):**  inp\_arr**.***append***([**i**,** arr**[**i**-**1**]])**  dict\_arr **=** **dict(**inp\_arr**)**  sorted\_arr **=** merge\_sort**(**dict\_arr**)**  res **=** sorted\_arr**.***values***()**  res **=** **[str(**x**)** **for** x **in** sorted\_arr**.***values***()]**  f**.***write***(**" "**.***join***(**res**))** |

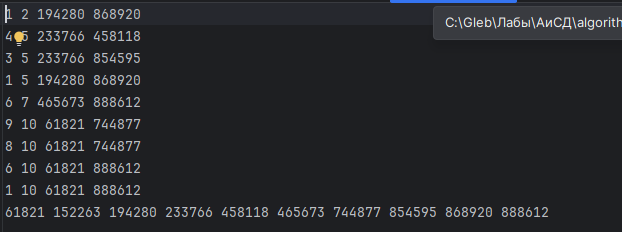
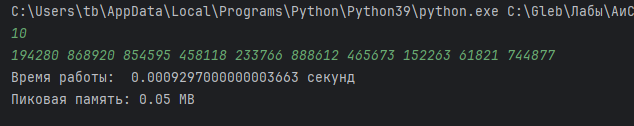
Текстовое объяснение решения:

Так как данная задача – модификация предыдущей, я объясню только то, в чем она отличается. Начнем с вызова: После считывания данных из файла input мы создаем буферный массив inp\_arr = [], а затем в цикле for i in range(1, len(arr)+1) заполняем его массивам, состоящими из 2х элементов: индекса (начиная с 1) и значения исходного массива. Далее из получившейся конструкции мы создаем словарь dict\_arr, в котором теперь все значния исходного массива будут пронумерованы. Затем мы вузываем функцию merge\_sort(), которая и выполняет все вычисления, а затем записываем данный в файл output. Рассмотрим сначала функцию merge т.к она вызывается функцией merge\_sort(). Находим сначала минимальный ключ в словаре l1 = min(list(left.keys())), а затем максимальный r1 = max(list(right.keys())), она нам понадобятся для записи границ слияния. Затем аналогично прошлой задаче мы проверяем все значения подмассивов, однако в данной задаче, в результирующий массив мы записываем не только подходящие значения, но и их номера. Затем после того, как мы отсортировали подмассивы, записываем полученные результаты в файл output и выводим отсортированный массив, как словарь. Функция merge\_sort аналогична функции в прошлой задаче, с поправкой на использование словарей.

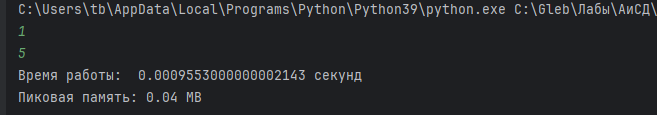
Результат работы кода на примерах задачи:

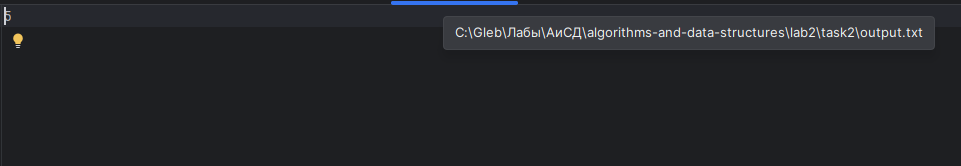


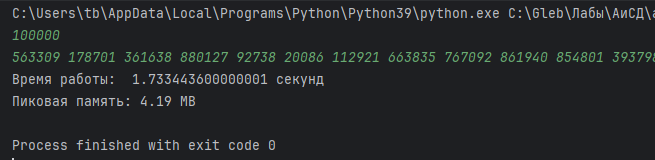


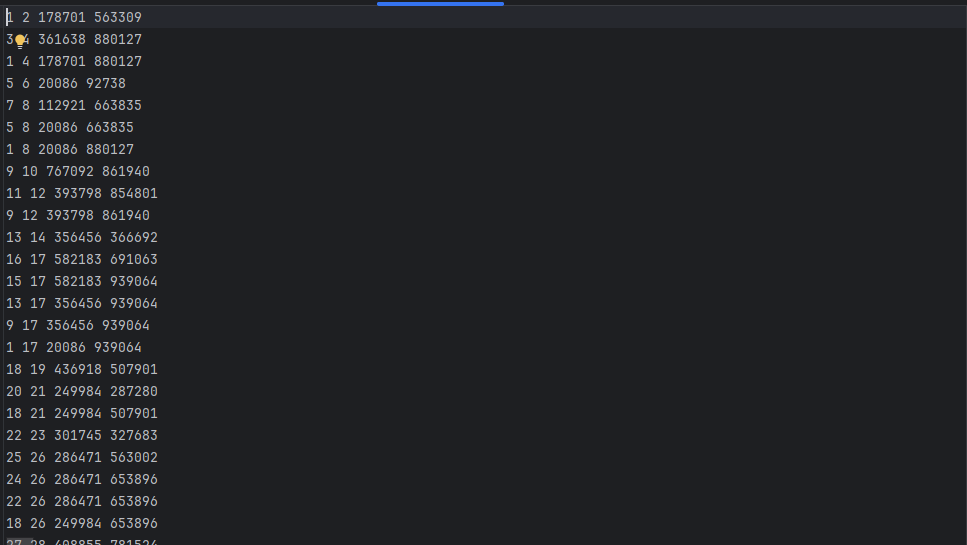


Результат работы кода на максимальных и минимальных значениях:









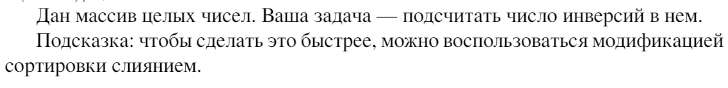
|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0005006999999999096 секунд | 0.04 MB |
| Пример 1 | 0.005204599999999893 секунд | 0.11 MB |
| Пример 2 | 0.0009297000000003663 секунд | 0.05 MB |
| Верхняя граница диапазона значений входных данных из текста задачи | 1.7544864999999996 секунд | 4.19 MB |

Вывод о задаче:

Данаая задача показывает, что устаявщиеся алгоритмы могут быть модифицированы для достижения новых целей.

**Задача № 3. Число инверсий**

Текст задачи:



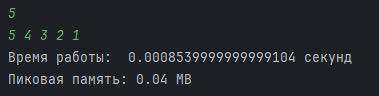
Листинг кода:

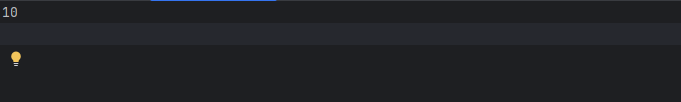
|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  a **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**a**))**  start **=** time**.***perf\_counter***()**  **def** merge\_and\_count**(**arr**,** left**,** mid**,** right**):**  left\_part **=** arr**[**left**:**mid **+** 1**]**  right\_part **=** arr**[**mid **+** 1**:**right **+** 1**]**  i **=** j **=** 0  inversions **=** 0  temp **=** **[]**  **while** i **<** **len(**left\_part**)** **and** j **<** **len(**right\_part**):**  **if** left\_part**[**i**]** **<=** right\_part**[**j**]:**  temp**.***append***(**left\_part**[**i**])**  i **+=** 1  **else:**  temp**.***append***(**right\_part**[**j**])**  j **+=** 1  inversions **+=** **len(**left\_part**)** **-** i  **while** i **<** **len(**left\_part**):**  temp**.***append***(**left\_part**[**i**])**  i **+=** 1  **while** j **<** **len(**right\_part**):**  temp**.***append***(**right\_part**[**j**])**  j **+=** 1  **for** i**,** val **in** **enumerate(**temp**):**  arr**[**left **+** i**]** **=** val  **return** inversions  **def** merge\_sort\_and\_count**(**arr**,** left**,** right**):**  inversions **=** 0  **if** left **<** right**:**  mid **=** **(**left **+** right**)** **//** 2  inversions **+=** merge\_sort\_and\_count**(**arr**,** left**,** mid**)**  inversions **+=** merge\_sort\_and\_count**(**arr**,** mid **+** 1**,** right**)**  inversions **+=** merge\_and\_count**(**arr**,** left**,** mid**,** right**)**  **return** inversions  **with** **open(**"input.txt"**,** "r"**)** **as** f**:**  n **=** **int(**f**.***readline***())**  arr **=** **list(map(int,** f**.***readline***().***split***()))**  inversions **=** merge\_sort\_and\_count**(**arr**,** 0**,** n **-** 1**)**  **with** **open(**"output.txt"**,** "w"**)** **as** f**:**  f**.***write***(str(**inversions**)** **+** "\n"**)** |

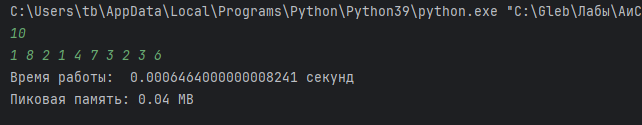
Текстовое объяснение решения:

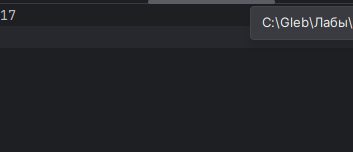
Данная задача так же является модификацией первой, соответственно ее принципы работу будут в целом похожи. Рассмотрим функцию merge(). Она работает подхоже на предыдущие варианты, однако в данной задаче в случае если элемент из левой части в сравнении с элементом из правой части меньше, то мы увеличиваем число инверсий на inversions += len(left\_part) – i т.к все предыдуще элементы левой чассти тоже будут создавать инверсии (мы сортируем массив слева направо) . Также отличием будет цикл for i, val in enumerate(temp), который переопределяет значения исходного массива. Также давайте разберем функцию

Результат работы кода на примерах из текста задачи:

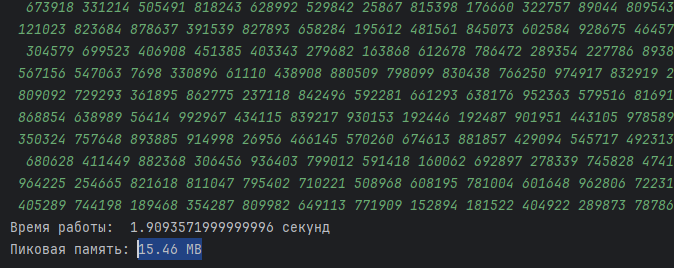


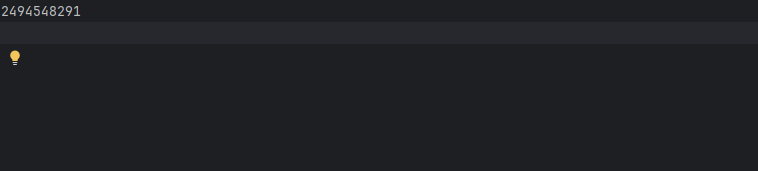


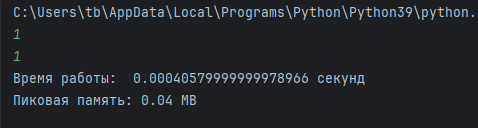


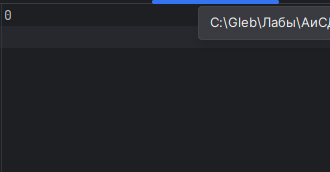


Результаты работы алгоритма на максимальных и минимальных значениях:









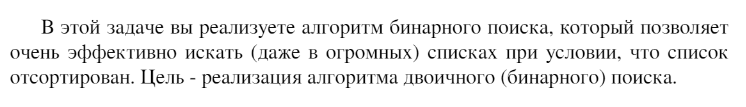
|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00040579999999978966 секунд | 0.04 MB |
| Пример 1 | 0.0007782999999994544 секунд | 0.04 MB |
| Пример 2 | 0.0006464000000008241 секунд | 0.04 MB |
| Верхняя граница диапазона значений | 1.9093571999999996 секунд | 15.46 MB |

Вывод о задаче:

Данная задача учит нас нестандратно подходить к уже привычным алгоритмам и методам для достижения новых целей.

**Задача № 4. Бинарный поиск.**

Текст задачи:

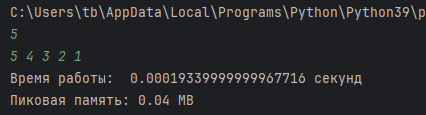


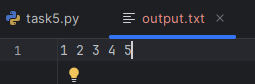
Листинг кода:

|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  arr **=** **input().***split***()**  m **=** **input()**  brr **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**arr**))**  f**.***write***(**"\n"**)**  f**.***write***(**m**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**brr**))**  **def** merge\_sort**(**arr**):**  **if** **len(**arr**)** **<=** 1**:**  **return** arr  mid **=** **len(**arr**)** **//** 2  left\_half **=** merge\_sort**(dict(list(**arr**.***items***())[:**mid**]))**  right\_half **=** merge\_sort**(dict(list(**arr**.***items***())[**mid**:]))**  **return** merge**(**left\_half**,** right\_half**)**  **def** merge**(**left**,** right**):**  sorted\_array **=** **[]**  i **=** j **=** 0  **while** i **<** **len(**left**)** **and** j **<** **len(**right**):**  **if** **list(**left**.***values***())[**i**]** **<** **list(**right**.***values***())[**j**]:**  sorted\_array**.***append***([list(**left**.***keys***())[**i**],** **list(**left**.***values***())[**i**]])**  i **+=** 1  **else:**  sorted\_array**.***append***([list(**right**.***keys***())[**j**],** **list(**right**.***values***())[**j**]])**  j **+=** 1  **while** i **<** **len(**left**):**  sorted\_array**.***append***([list(**left**.***keys***())[**i**],** **list(**left**.***values***())[**i**]])**  i **+=** 1  **while** j **<** **len(**right**):**  sorted\_array**.***append***([list(**right**.***keys***())[**j**],** **list(**right**.***values***())[**j**]])**  j **+=** 1  **return** **dict(**sorted\_array**)**  **def** binary\_search**(**arr\_dict\_sorted**,** n**,** b**):**  **if** n **==** 0**:**  **return** **-**1  mid **=** n **//** 2  **if** **list(**arr\_dict\_sorted**.***values***())[**mid**]** **==** b**:**  **return** **list(**arr\_dict\_sorted**.***keys***())[**mid**]**  **if** **list(**arr\_dict\_sorted**.***values***())[**mid**]** **>** b**:**  res **=** binary\_search**(dict(list(**arr\_dict\_sorted**.***items***())[:**mid**]),** **len(dict(list(**arr\_dict\_sorted**.***items***())[:**mid**])),** b**)**  **if** **list(**arr\_dict\_sorted**.***values***())[**mid**]** **<** b**:**  res **=** binary\_search**(dict(list(**arr\_dict\_sorted**.***items***())[**mid **+** 1**:]),** **len(dict(list(**arr\_dict\_sorted**.***items***())[**mid **+** 1**:])),** b**)**  **return** res  **with** **open(**"input.txt"**,** "r"**)** **as** f**:**  n **=** **int(**f**.***readline***())**  a1 **=** f**.***readline***().***split***()**  arr **=** **[int(**x**)** **for** x **in** a1**]**  m **=** **int(**f**.***readline***())**  b1 **=** f**.***readline***().***split***()**  brr **=** **[int(**x**)** **for** x **in** b1**]**  arr\_for\_dict **=** **[]**  **for** i **in** **range(len(**arr**)):**  arr\_for\_dict**.***append***([**i**,** arr**[**i**]])**  arr\_dict **=** **dict(**arr\_for\_dict**)**  arr\_dict\_sorted **=** merge\_sort**(**arr\_dict**)**  res **=** **[]**  **for** b **in** brr**:**  res**.***append***(**binary\_search**(**arr\_dict\_sorted**,** n**,** b**))**  **if** res**[-**1**]** **==** **None:**  res**[-**1**]** **=** **-**1  **print(**res**)** |

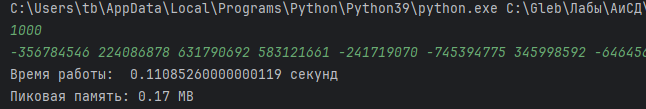
Текстовое объяснение решения:  
Функции merge и merge\_sort мы пропустим т.к они идентичны одноименным функциям во второй задаче и используются лишь для сортировки.

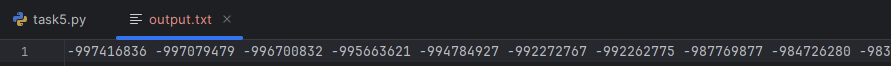
Результат работы кода на примерах из текста задачи:





Результат работы кода на максимальных и минимальных значениях:

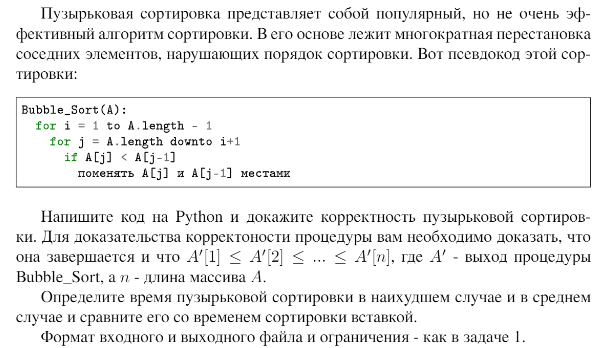




|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00018129999999993984 секунд | 0.04 MB |
| Пример 1 | 0. 0.0002775999999995449 | 0.04 MB |
| Пример 2 |  |  |
| Верхняя граница диапазона значений входных данных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

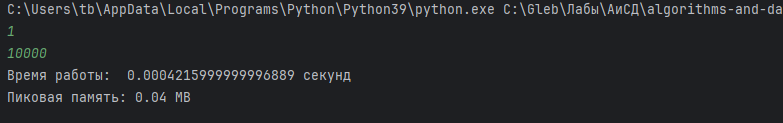
Листинг кода:

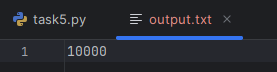
|  |
| --- |
| **import** time  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  a **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**a**))**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

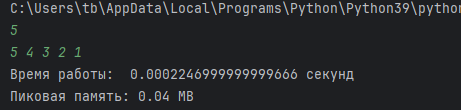
Текстовое объяснение решения:

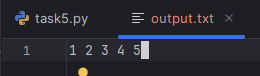
Данная сортировка крайне неэффективна. В ней сортировка массива происходит за счет постоянной замены каждого числа, которое больше своего соседа справа (при сортировке по возрастанию) с этим самым соседом. С изменением шага цикла, мы бы так добрали до конца массива, постоянно меняя числа местами, а затем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

Результат работы кода на примерах задачи:

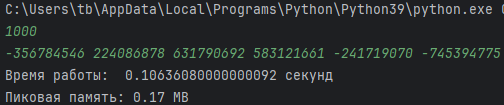


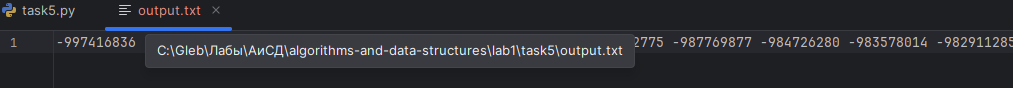






Результаты работы кода на max:

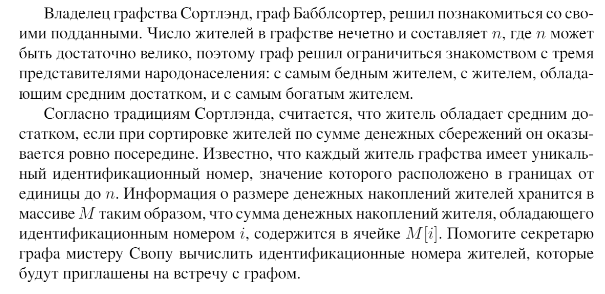




|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0004215999999996889 секунд | 0.04 MB |
| Пример из задачи | 0.000493099999999913 секунд | 0,04 MB |
| Пример из задачи |  |  |
| Верхняя граница диапазона значений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


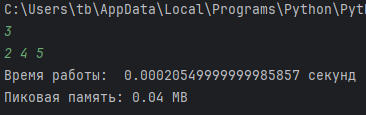
Листинг кода:

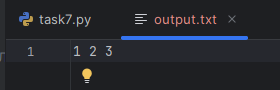
|  |
| --- |
| **from** time **import** **\***  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  a **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**a**))**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

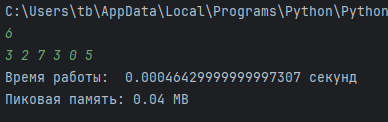
Текстовое объяснение решения:

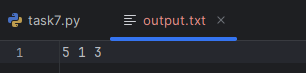
В данной задаче нам надо вывести самое маленькое, среднее и самое большое числа массива. Для сортировки я использую алгоритм сортировки вставкой из 1го задания. Так как в задаче необходимо вывести номера этих людей (чисел) относительно изначального (неотсортированного) списка, мы скопируем изначальный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



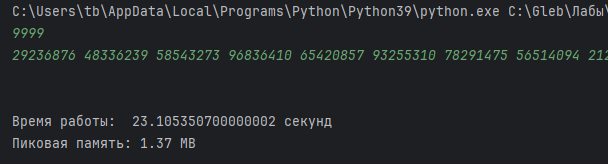


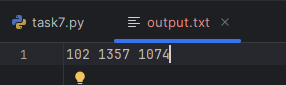




Результат работы кода на максимальных и минимальных значениях:

Max:



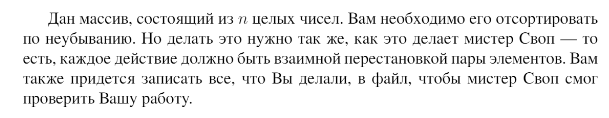


|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0004224999999999923 секунд | 0.04 MB |
| Пример из задачи | 0.0003503000000000256 секунд | 0.04 MB |
| Пример из задачи |  |  |
| Верхняя граница диапазона значений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



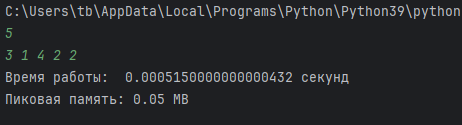
Листинг кода:

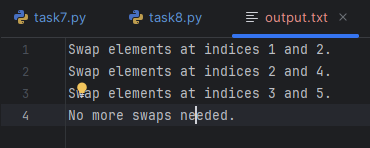
|  |
| --- |
| **from** time **import** **\***  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n **=** **input()**  a **=** **input().***split***()**  f**.***write***(**n**)**  f**.***write***(**"\n"**)**  f**.***write***(**" "**.***join***(**a**))**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

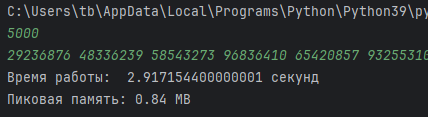
Данный алгоритм, подобно пузырьковой сортировке меняет соседние элементы массива, если они находятся в неправильном порядке, однако лишь до момента, когда они не окажутся на своей правильной позиции.

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:



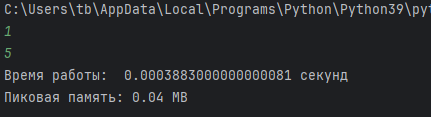
Листинг задачи:

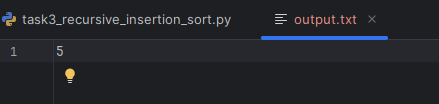
|  |
| --- |
| **import** time  **from** time **import** perf\_counter  **import** tracemalloc  tracemalloc**.***start***()**  **with** **open** **(**"input.txt"**,** "w"**)** **as** f**:**  n1 **=** **input()**  a1 **=** **input().***split***()**  f**.***write***(**n1**)**  f**.***write***(**"\n"**)**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

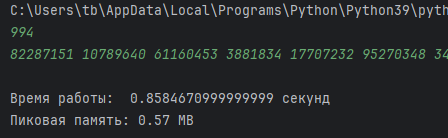
Текстовое объяснение:

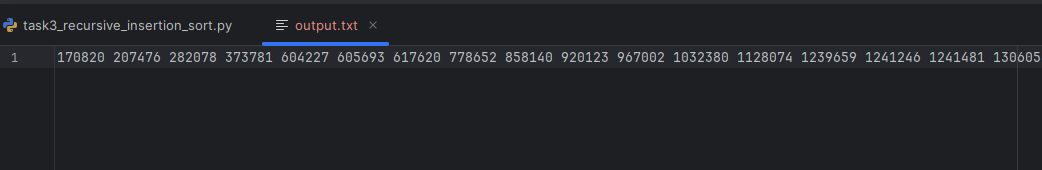
Это рекурсивный вариант сортировки вставкой. В функцию подаем два значения: кол-во элементов в массиве, который надо отсортировать и сам массив. Закручиваем нашу рекурсию повторным вызовом с новым значением n = n – 1. Благодаря данному действию после того, как n достигнет значения 1, функции начнут вызывать 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.