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 15

Выполнил:

Левахин Лев Александрович

К3140

Проверил:

--

Санкт-Петербург

2024 г.

# Содержание отчета

[**Содержание отчета**](#_heading=h.gjdgxs) **2**

[**Задачи по варианту**](#_heading=h.30j0zll) **3**

[Задача №1. Сортировка вставкой](#_heading=h.1fob9te)

[Задача №2. Сортировка вставкой +](#_heading=h.1fob9te)

[Задача №3. Сортировка вставкой по убыванию](#_heading=h.1fob9te)

[Задача №4. Линейный поиск](#_heading=h.1fob9te)

[Задача №5. Сортировка выбором](#_heading=h.1fob9te)

[**Дополнительные задачи**](#_heading=h.3znysh7) **4**

[Задача №6. Пузырьковая сортировка](#_heading=h.2et92p0)

[Задача №7. Знакомство с жителями Сортлэнда](#_heading=h.1fob9te)

[**Вывод**](#_heading=h.tyjcwt) **5**

# Задачи по варианту

**[Задача №1. Сортировка вставкой](#_heading=h.1fob9te)**

Листинг кода.

*"""  
Сортировка вставкой базовая  
"""*file\_in = open("../txtfiles/input.txt")  
file\_out = open("../txtfiles/output.txt", "w")  
  
n = int(file\_in.readline()) #Количество элементов  
lst = list(map(int, file\_in.readline().split())) #Список с элементами  
  
def insertion\_sort(n:int, lst:list) ->str:  
 for i in range(1, n):  
 key = lst[i]  
 j = i-1  
 while (j>=0) and (lst[j]>key):  
 lst[j+1] = lst[j]  
 j -=1  
 lst[j+1] = key  
 lst = [str(el) for el in lst]  
 return " ".join(lst)  
  
file\_out.write(insertion\_sort(n, lst))

Текстовое объяснение решения.

Считываем файлы входа и выхода. Принимаем заданные значения из файла входа. Создаём функцию сортировки вставкой, которая принимает на вход количество элементов массива и сам список. Проходимся по элементам с 1 до количества элементов. Переменной ключу задаём значение текущего элемента, переменной j задаём значение предыдущего. Пока предыдущий элемент больше или равен 0 и этот элемент больше, чем ключ(текущий), будем выполнять условие. Следующий элемент списка от j приравниваем к предыдущему и убавляем j. Таким образом сортируя массив. Приравниваем элемент от j к исходному элементу – ключу. Преобразовываем каждый элемент списка в строку, чтобы вернуть строку значений через пробел, с помощью метода join. Записываем результат в файл.

Результат работы кода на примерах из текста задачи:

(скрины input output файлов)

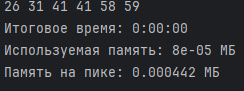


Тесты к задаче:

Листинг кода:

from lab1.task1.src.task1 import insertion\_sort  
import datetime  
import tracemalloc  
  
file\_in = open("../txtfiles/input.txt")  
  
n = int(file\_in.readline()) #Количество элементов  
lst = list(map(int, file\_in.readline().split())) #Список с элементами  
  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
print(insertion\_sort(n, lst)) # Выводим результат отработанной функции  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ") #Выводим время работы в мегабайтах

Скрины работы тестов:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Пример из задачи | 0ч:00м:00с | 0.000442 Мб |

Вывод по задаче:

1. Сортировка вставкой не очень быстро работает, но довольна проста и понятна.

[**Задача №2. Сортировка вставкой +**](#_heading=h.1fob9te)

Листинг кода.

*"""  
Сортировка вставкой +, дополнительно выводит номер, на который был поставлен элемент при обработке  
"""*file\_in = open("../txtfiles/input.txt")  
file\_out = open("../txtfiles/output.txt", "w")  
  
n = int(file\_in.readline()) # Количество элементов  
lst = list(map(int, file\_in.readline().split())) # Список с элементами  
  
def insertion\_sort(n:int, lst:list) -> str:  
 indexes = "1 "  
 for i in range(1, n):  
 key = lst[i]  
 j = i-1  
 while (j>=0) and (lst[j]>key):  
 lst[j+1] = lst[j]  
 j -=1  
 lst[j+1] = key  
 indexes+=str(j+1+1)+" "  
  
 lst = [str(el) for el in lst]  
 res = indexes+"\n"+ " ".join(lst)  
 return res  
  
file\_out.write(insertion\_sort(n, lst))

Текстовое объяснение решения.

Считываем файлы входа и выхода. Принимаем заданные значения из файла входа. Создаём функцию сортировки вставкой, которая принимает на вход количество элементов массива и сам список. Создаём строку куда будем записывать индексы. Так как первый элемент остаётся на своём месте, сразу запишем в эту строку единицу. Проходимся по элементам с 1 до количества элементов. Переменной ключу задаём значение текущего элемента, переменной j задаём значение предыдущего. Пока предыдущий элемент больше или равен 0 и этот элемент больше, чем ключ(текущий), будем выполнять условие. Следующий элемент списка от j приравниваем к предыдущему и убавляем j. Таким образом сортируя массив. Добавляем к строке индексов число, на которое поставили элемент списка. Приравниваем элемент от j к исходному элементу – ключу. Создаём переменную рез, в которую записываем строку индексы, переходим на новую строку, преобразовываем каждый элемент списка в строку, чтобы вернуть строку значений через пробел, с помощью метода join. Записываем результат в файл.

Результат работы кода на примерах из текста задачи:

(скрины input output файлов)

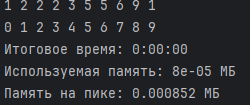


Тесты к задаче:

Листинг кода:

from lab1.task2.src.task2 import insertion\_sort  
import datetime  
import tracemalloc  
  
file\_in = open("../txtfiles/input.txt")  
  
n = int(file\_in.readline()) #Количество элементов  
lst = list(map(int, file\_in.readline().split())) #Список с элементами  
  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
print(insertion\_sort(n, lst)) # Выводим результат отработанной функции  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ") #Выводим время работы в мегабайтах

Скрины работы тестов:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Пример из задачи | 0:00:00 | 0.000852 Мб |

Вывод по задаче:

1) На основе сортировки вставкой можно реализовать её более сложный и интересный прототип.

[**Задача №3. Сортировка вставкой по убыванию**](#_heading=h.1fob9te)

Листинг кода.

*"""  
Сортировка вставкой в невозрастающем(убывающем) порядке  
"""*file\_in = open("../txtfiles/input.txt")  
file\_out = open("../txtfiles/output.txt", "w")  
  
n = int(file\_in.readline()) # Количество элементов  
lst = list(map(int, file\_in.readline().split())) # Список с элементами  
  
def insertion\_sort(n:int, lst:list) ->str:  
 for i in range(1, n):  
 key = lst[i]  
 j = i-1  
 while (j>=0) and (lst[j]<key): # Поменялся знак неравенства, теперь нам подходит случай, когда предыдущий элемент меньше чем ключ  
 lst[j+1] = lst[j]  
 j -=1  
 lst[j+1] = key  
 lst = [str(el) for el in lst]  
 return " ".join(lst)  
  
file\_out.write(insertion\_sort(n, lst))

Текстовое объяснение решения.

Считываем файлы входа и выхода. Принимаем заданные значения из файла входа. Создаём функцию сортировки вставкой, которая принимает на вход количество элементов массива и сам список. Проходимся по элементам с 1 до количества элементов. Переменной ключу задаём значение текущего элемента, переменной j задаём значение предыдущего. Пока предыдущий элемент больше или равен 0 и этот элемент меньше, чем ключ(текущий), будем выполнять условие. Следующий элемент списка от j приравниваем к предыдущему и убавляем j. Таким образом сортируя массив. Приравниваем элемент от j к исходному элементу – ключу. Преобразовываем каждый элемент списка в строку, чтобы вернуть строку значений через пробел, с помощью метода join. Записываем результат в файл.

Результат работы кода на примерах из текста задачи:

(скрины input output файлов)

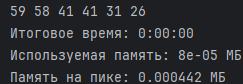


Тесты к задаче:

Листинг кода:

from lab1.task3.src.task3 import insertion\_sort  
import datetime  
import tracemalloc  
  
file\_in = open("../txtfiles/input.txt")  
  
n = int(file\_in.readline()) #Количество элементов  
lst = list(map(int, file\_in.readline().split())) #Список с элементами  
  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
print(insertion\_sort(n, lst)) # Выводим результат отработанной функции  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ") #Выводим время работы в мегабайтах

Скрины работы тестов:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Пример из задачи | 0:00:00 | 0.000442 Мб |

Вывод по задаче:

1. Поменяв всего лишь один символ, можно из сортировки вставкой по возрастанию сделать сортировку вставкой по убыванию.

[**Задача №4. Линейный поиск**](#_heading=h.1fob9te)

Листинг кода.

*"""  
Линейный поиск  
"""*file\_in = open("../txtfiles/input.txt")  
file\_out = open("../txtfiles/output.txt", "w")  
  
a = list(map(int, file\_in.readline().split()))  
v = int(file\_in.readline())  
  
def line\_search(lst: list, find\_el: int):  
 len\_lst = len(lst)  
 cnt\_el = 0  
 ind\_list = []  
 for i in range(len\_lst):  
 if lst[i] == find\_el:  
 cnt\_el+=1  
 ind\_list.append(i)  
 if len(ind\_list) >1:  
 return f"Cnt v: {len(ind\_list)}\nIndex's v: {", ".join([str(el) for el in ind\_list])}"  
 elif len(ind\_list) == 1:  
 return ind\_list[0]  
 return -1  
  
file\_out.write(str(line\_search(a, v)))

Текстовое объяснение решения.

Считываем файлы входа и выхода. Принимаем заданные значения из файла входа. Создаём функцию линейного поиска, которая принимает на вход список с элементами и значение элемента, индекс которого нужно найти. Создаём переменную с длиной списка, счётчик элементов, список с индексом значений. Проходимся по числам с 0 до количества элементов. Если находим нужный элемент – то добавляем к счётчику 1 и добавляем аргумент в список. Если длина этого списка больше одного, будем записывать в файл все элементы и их индексы. Если же длина равна 1, то возвращаем только сам элемент. Если такого элемента нет в строке, возвращаем -1. Записываем результат в файл.

Результат работы кода на примерах из текста задачи:

(скрины input output файлов)





Результат работы кода на собственном примере:

1)





2)





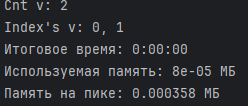
Тесты к задаче:

Листинг кода:

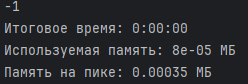
from lab1.task4.src.task4 import line\_search  
import datetime  
import tracemalloc  
  
file\_in = open("../txtfiles/input.txt")  
  
a = list(map(int, file\_in.readline().split()))  
v = int(file\_in.readline())  
  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
print(line\_search(a, v)) # Выводим результат отработанной функции  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ") #Выводим время работы в мегабайтах

Скрины работы тестов:

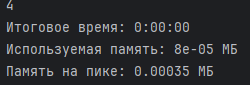
0)Из задачи



1)



2)



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Пример из задачи | 0:00:00 | 0.000358 Мб |
| Пример 1 | 0:00:00 | 0.00035 Мб |
| Пример 2 | 0:00:00 | 0.00035 Мб |

Вывод по задаче:

1. С помощью условных операторов и разных условий задачи можно сильно её усложнить.
2. Чтобы легко узнать индекс элемента, можно перебирать не сами элементы списка, а только числа, а проверять уже список от индекса.

[**Задача №5. Сортировка выбором**](#_heading=h.1fob9te)

Листинг кода.

*"""  
Сортировка выбором  
"""*file\_in = open("../txtfiles/input.txt")  
file\_out = open("../txtfiles/output.txt", "w")  
  
n = int(file\_in.readline()) # Количество элементов  
lst = list(map(int, file\_in.readline().split())) # Список с элементами  
  
def selection\_sort(n: int, lst: list) -> str:  
 for i in range(n-1):  
 key = lst[i] # Выбираем минимальный элемент  
 min\_ind = i  
 for j in range(i+1, n):  
 if key>lst[j]:  
 key = lst[j]  
 min\_ind = j  
 if min\_ind != i: # Обмен значениями  
 lst[i], lst[min\_ind] = lst[min\_ind], lst[i]  
 lst = [str(el) for el in lst]  
 return " ".join(lst)

file\_out.write(selection\_sort(n, lst))

Текстовое объяснение решения.

Считываем файлы входа и выхода. Принимаем заданные значения из файла входа. Создаём функцию сортировки выбором, которая принимает на вход количество элементов массива и сам список. Проходимся по всем элементам в списке до n-1(потому что будем сравнивать со следующим). Выбираем минимальный элемент, по умолчанию равный первому вхождению, также запоминаем его индекс. Проходимся по элементам от i до количества элементов. Находим минимальный элемент и обновляем его и его индекс соответственно. Далее производим обмен значений между первым и минимальным элементами. Так сортируем весь список. Преобразовываем каждый элемент списка в строку, чтобы вернуть строку значений через пробел, с помощью метода join. Записываем результат в файл.

Результат работы кода на примерах из текста задачи:

(скрины input output файлов)

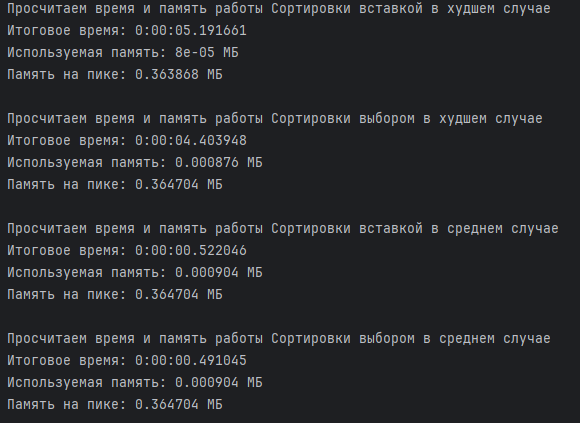


Тесты к задаче:

Листинг кода:

from lab1.task1.src.task1 import insertion\_sort  
from lab1.task5.src.task5 import selection\_sort  
import datetime  
import tracemalloc  
import random  
  
lst\_hud = [random.randint(1, 100\_000) for i in range(6\_000)]  
n\_hud = 6\_000  
  
lst\_sr = [random.randint(1, 10\_000) for j in range(2\_000)]  
n\_sr = 2\_000  
  
print("Просчитаем время и память работы Сортировки вставкой в худшем случае")  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
insertion\_sort(n\_hud, lst\_hud)  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ\n") #Выводим время работы в мегабайтах  
  
  
  
print("Просчитаем время и память работы Сортировки выбором в худшем случае")  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
selection\_sort(n\_hud, lst\_hud)  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ\n") #Выводим время работы в мегабайтах  
  
  
  
print("Просчитаем время и память работы Сортировки вставкой в среднем случае")  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
insertion\_sort(n\_sr, lst\_sr)  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ\n") #Выводим время работы в мегабайтах  
  
  
  
print("Просчитаем время и память работы Сортировки выбором в среднем случае")  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
selection\_sort(n\_sr, lst\_sr)  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ\n") #Выводим время работы в мегабайтах

Скрины работы тестов(Сравнение работы двух сортировок на средних и максимальных значениях):



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти (Мб) |
| Сортировка вставкой средние значения | 5мс | 0.36 |
| Сортировка выбором средние значения | 0.49 мс | 0.36 |
| Сортировка вставкой максимальные значения | 5с | 0.38 |
| Сортировка выбором максимальные значения | 4с | 0.37 |

Вывод по задаче:

1. Сортировка выбором работает быстрее сортировки вставкой.
2. Интересно сравнивать две сортировки, которые работают разными методами.
3. Одну и ту же задачу отсортировать массив можно решить множеством разных способов.

# Дополнительные задачи

**[Задача №6. Пузырьковая сортировка](#_heading=h.2et92p0)**

Листинг кода.

*"""  
Пузырьковая сортировка  
"""*file\_in = open("../txtfiles/input.txt")  
file\_out = open("../txtfiles/output.txt", "w")  
  
n = int(file\_in.readline()) #Количество элементов  
lst = list(map(int, file\_in.readline().split())) #Список с элементами  
  
def bubble\_sort(n:int, lst:list) ->str:  
 for i in range(n-1):  
 for j in range(i+1, n):  
 if lst[j] <= lst[i]:  
 lst[j], lst[i] = lst[i], lst[j]  
 lst = [str(el) for el in lst]  
 return " ".join(lst)  
  
  
file\_out.write(bubble\_sort(n, lst))

Текстовое объяснение решения.

Считываем файлы входа и выхода. Принимаем заданные значения из файла входа. Создаём функцию сортировки пузырьком, которая принимает на вход количество элементов массива и сам список. Проходимся по числам до количества элементов - 1. Создаём вложенный цикл от цифры на единицу больше и до количества элементов. Проверяем меньше ли следующий элемент или равен. Если да, меняем их местами. Преобразовываем каждый элемент списка в строку, чтобы вернуть строку значений через пробел, с помощью метода join. Записываем результат в файл.

Результат работы кода на примерах из текста задачи:

(скрины input output файлов)



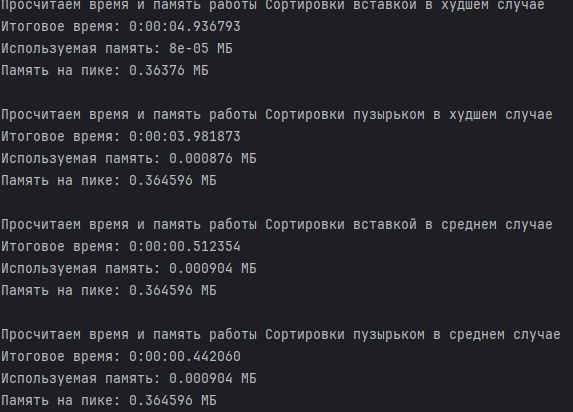
Тесты к задаче:

Листинг кода:

from lab1.task1.src.task1 import insertion\_sort  
from lab1.task6.src.task6 import bubble\_sort  
import datetime  
import tracemalloc  
import random  
  
lst\_hud = [random.randint(1, 100\_000) for i in range(6\_000)]  
n\_hud = 6\_000  
  
lst\_sr = [random.randint(1, 10\_000) for j in range(2\_000)]  
n\_sr = 2\_000  
  
print("Просчитаем время и память работы Сортировки вставкой в худшем случае")  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
insertion\_sort(n\_hud, lst\_hud)  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ\n") #Выводим время работы в мегабайтах  
  
  
  
print("Просчитаем время и память работы Сортировки пузырьком в худшем случае")  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
bubble\_sort(n\_hud, lst\_hud)  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ\n") #Выводим время работы в мегабайтах  
  
  
  
print("Просчитаем время и память работы Сортировки вставкой в среднем случае")  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
insertion\_sort(n\_sr, lst\_sr)  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ\n") #Выводим время работы в мегабайтах  
  
  
  
print("Просчитаем время и память работы Сортировки пузырьком в среднем случае")  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
bubble\_sort(n\_sr, lst\_sr)  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ\n") #Выводим время работы в мегабайтах

работы в мегабайтах

Скрины работы тестов:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти (Мб) |
| Сортировка вставкой средние значения | 5мс | 0.37 |
| Сортировка пузырьком средние значения | 4 мс | 0.37 |
| Сортировка вставкой максимальные значения | 5с | 0.37 |
| Сортировка пузырьком максимальные значения | 4с | 0.37 |

Вывод по задаче:

1. Вложенный цикл существенно повышает время работы алгоритма.
2. Такой вид решения первый, что приходит в голову, когда речь идёт о сортировке.

[**Задача №7. Знакомство с жителями Сортлэнда**](#_heading=h.1fob9te)

Листинг кода.

*"""  
В графстве Сортленда нужно найти 3х человек  
- Самого бедного  
- Среднего достатка  
- Самого богатого  
"""*file\_in = open("../txtfiles/input.txt")  
file\_out = open("../txtfiles/output.txt", "w")  
  
n = int(file\_in.readline()) # Количество жителей  
m = list(map(float, file\_in.readline().split())) # Список с состоянием житиелей, индексы жителей i+1  
  
def Sortland(n: int, m: list) -> str:  
 win\_list = []  
  
 win\_list.append(m.index(min(m)) + 1)  
 m\_sort = sorted(m)  
 win\_list.append(m.index(m\_sort[len(m\_sort) // 2]) + 1)  
 win\_list.append(m.index(max(m)) + 1)  
  
 win\_list = [str(el) for el in win\_list]  
 return " ".join(win\_list)  
  
file\_out.write(Sortland(n,m))

Текстовое объяснение решения.

Считываем файлы входа и выхода. Принимаем заданные значения из файла входа. Создаём функцию Сортлэнд, которая принимает на вход количество элементов массива и сам список. Создаём список победителей, добавляем в него индекс минимального элемента оригинального списка (+1 чтобы совпадало с нумерацией). Далее сортируем список по возрастанию, добавляем в список победителей жителя со средним достатком. Для этого находим индекс среднего по заработку человека, целочисленно деля длину отсортированного списка на 2. Добавляем индекс максимального элемента. Таким образом, получаем 3х победителей. Преобразовываем каждый элемент списка в строку, чтобы вернуть строку значений через пробел, с помощью метода join. Записываем результат в файл.

Результат работы кода на примерах из текста задачи:

(скрины input output файлов)

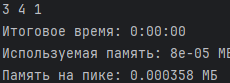


Тесты к задаче:

Листинг кода:

from lab1.task7.src.task7 import Sortland  
import datetime  
import tracemalloc  
  
file\_in = open("../txtfiles/input.txt")  
  
n = int(file\_in.readline()) #Количество элементов  
lst = list(map(float, file\_in.readline().split())) #Список с элементами  
  
tracemalloc.start() # Запускаем счётчик памяти  
start\_time = datetime.datetime.now() # Запускаем счётчик времени  
  
print(Sortland(n, lst)) # Выводим результат отработанной функции  
  
finish\_time = datetime.datetime.now() # Измеряем время конца работы  
print("Итоговое время:",finish\_time - start\_time) # Выводим итоговое время  
  
current, peak = tracemalloc.get\_traced\_memory() # Присваеваем двум переменным память, используемую сейчас, и на пике  
print(f"Используемая память: {current / 10\*\*6} МБ\nПамять на пике: {peak / 10\*\*6} МБ") #Выводим время работы в мегабайтах

Скрины работы тестов:



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Пример из задачи | 0:00:00 | 0.000358 Мб |

Вывод по задаче:

1. Задачу можно решить лёгким способом, не создавая собственную сортировку.

# Вывод

Существует множество видов сортировок, как самые оптимизированные, так и самые неоптимизированные. С ними можно совершать огромное количество действий и решать множество задач, также подстраивая их под свои цели. Они отличаются между собой по времени выполнения и затрачиваемой памяти. Чтобы использовать много видов сортировок и подбирать их под свои нужды, нужно понимать логику работы таких сортировок, о которых идёт речь в данной лабораторной. Я постарался использовать все свои знания, чтобы грамотно написать и применить их.