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 9

Выполнила:
Левчук С.А.
К3141

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета **2**
Задачи по варианту **3**
Задание №1. Сортировка вставкой 3
Задание №3. Сортировка вставкой по убыванию 6
Задание №6. Пузырьковая сортировка 9
Дополнительные задачи **13**
Задание №2. Сортировка вставкой + 13
Задание №4. Линейный поиск 17
Задание №5. Сортировка выбором 20
Вывод **24**

Задачи по варианту

Задание №1. Сортировка вставкой

1 задача. Сортировка вставкой

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

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

Выберите любой набор данных, подходящих по формату, и протестируйте алгоритм.

Решение:

```
def insertion_sort(n, numbers):
    for i in range(1, n):
        for j in range(i-1, -1, -1):
            if numbers[i] < numbers[j]:
                numbers[i], numbers[j] = numbers[j], numbers[i]
                i, j = j, i
    return numbers


with open('input.txt', 'r') as file:
    n = int(file.readline())
    numbers = list(map(int, file.readline().split()))
file.close()

if 1 <= n <= 10**3:
    with open('output.txt', 'w') as file:
        file.write(' '.join(map(str, insertion_sort(n, numbers))))
    file.close()
else:
    print('Введенные данные не соответствуют условию')
```

1. Сначала опишем основную функцию - insertion_sort. На вход она принимает два параметра: n - число элементов массива и numbers - массив элементов. Далее производим сортировку вставкой данного массива: попарно сравниваем элементы и в случае, если правый

меньше левого, меняем их местами, а также делаем это со значениями переменных i и j, содержащих индексы рассматриваемых элементов. После выполнения этого алгоритма функция возвращает отсортированный список.

2. Открываем файл input.txt, содержащий входные данные. Читаем первую строку и приводим к целочисленному виду полученное число (n). Далее читаем вторую строку и преобразуем ее в список целых чисел (numbers). Закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, выполняем функцию insertion_sort(n, numbers), открываем файл output.txt, записываем в него результат работы программы - отсортированный массив numbers, после чего закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Результат работы программы на примере из текста задачи:

Входные данные:

```
6
31 41 59 26 41 58
```

Выходные данные:

```
26 31 41 41 58 59
```

Тесты времени выполнения и затраченной памяти.

1. Пример из текста задачи

```
import time
import psutil
from lab1.task1.src.task1 import insertion_sort

test_nums1 = [31, 41, 59, 26, 41, 58]

start_time1 = time.perf_counter()
test1 = insertion_sort(len(test_nums1), test_nums1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f} Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 8.400005754083395e-06 секунд
Память: 15.23 Мбайт

Process finished with exit code 0
```

2. Минимальное значение

```
import time
```

```
import psutil
from lab1.task1.src.task1 import insertion_sort

test_nums2 = [0]

start_time2 = time.perf_counter()
test2 = insertion_sort(len(test_nums2), test_nums2)
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs
Время: 2.900022082030773e-06 секунд
Память: 14.80 Мбайт

Process finished with exit code 0
```

3. Максимальное значение

```
import time
import psutil
import random
from lab1.task1.src.task1 import insertion_sort

test_nums3 = [random.randint(-10**9, 10**9+1) for _ in range(1,
10**3+1)]

start_time3 = time.perf_counter()
test3 = insertion_sort(len(test_nums3), test_nums3)
print(f'Время: {time.perf_counter() - start_time3} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs
Время: 0.024462300003506243 секунд
Память: 15.13 Мбайт

Process finished with exit code 0
```

Вывод по задаче:

В ходе решения данной задачи мы научились осуществлять сортировку вставкой массива целых чисел и выяснили, каково время ее выполнения и затраты памяти на предельных и средних значениях.

Задание №3. Сортировка вставкой по убыванию

3 задача. Сортировка вставкой по убыванию

Перепишите процедуру Insertion-sort для сортировки в невозрастающем порядке вместо неубывающего с использованием процедуры **Swap**.

Формат входного и выходного файла и ограничения - как в задаче 1.

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

Решение:

```
def swap(a, b):
    c = b
    b = a
    a = c
    return a, b


def insertion_sort_swap(n, numbers):
    for i in range(1, n):
        for j in range(i-1, -1, -1):
            if numbers[i] > numbers[j]:
                        numbers[i], numbers[j] = swap(numbers[i],
numbers[j])
                i, j = j, i
    return numbers


with open('input.txt', 'r') as file:
    n = int(file.readline())
    numbers = list(map(int, file.readline().split()))
file.close()

if 1 <= n <= 10**3:
    with open('output.txt', 'w') as file:
                file.write('  '.join(map(str,  insertion_sort_swap(n,
numbers))))
    file.close()
else:
    print('Введенные данные не соответствуют условию')
```

1. Опишем функцию swap: на вход она получает два параметра - целые числа a и b. Далее мы вводим вспомогательную переменную c, с помощью которой меняем значения a и b местами (то есть a := b, b := a). После чего меняем функцию insertion_sort из задачи 1 таким образом, что теперь мы сдвигаем те числа, что больше, влево, применяя для этого функцию swap.

2. Открываем файл input.txt, содержащий входные данные. Читаем первую строку и приводим к целочисленному виду полученное число (n). Далее читаем вторую строку и преобразуем ее в список целых чисел (numbers). Закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, выполняем функцию insertion_sort_swap(n, numbers), открываем файл output.txt, записываем в него результат работы программы - отсортированный массив numbers, после чего закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Результат работы программы на примере из текста задачи:

Входные данные:

```
6
31 41 59 26 41 58
```

Выходные данные:

```
59 58 41 41 31 26
```

Тесты времени выполнения и затраченной памяти.

1. Пример из текста задачи

```
import time
import psutil
from lab1.task3.src.task3 import insertion_sort_swap

test_nums1 = [31, 41, 59, 26, 41, 58]

start_time1 = time.perf_counter()
test1 = insertion_sort_swap(len(test_nums1), test_nums1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 1.0700023267418146e-05 секунд
Память: 15.25 Мбайт

Process finished with exit code 0
```

2. Минимальное значение

```
import time
import psutil
from lab1.task3.src.task3 import insertion_sort_swap

test_nums2 = [0]
```

```
start_time2 = time.perf_counter()
test2 = insertion_sort_swap(len(test_nums2), test_nums2)
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 2.500019036233425e-06 секунд
Память: 15.00 Мбайт

Process finished with exit code 0
```

3. Максимальное значение

```
import time
import psutil
import random
from lab1.task3.src.task3 import insertion_sort_swap

test_nums3 = [random.randint(-10**9, 10**9+1) for _ in range(1,
10**3+1)]

start_time3 = time.perf_counter()
test3 = insertion_sort_swap(len(test_nums3), test_nums3)
print(f'Время: {time.perf_counter() - start_time3} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 0.0373006000299938 секунд
Память: 14.89 Мбайт

Process finished with exit code 0
```

Вывод по задаче:

В ходе решения данной задачи мы научились осуществлять сортировку вставкой по убыванию с применением дополнительной функции swap массива целых чисел и выяснили, каково время ее выполнения и затраты памяти на предельных и средних значениях.

Задание №6. Пузырьковая сортировка

6 задача. Пузырьковая сортировка

Пузырьковая сортировка представляет собой популярный, но не очень эффективный алгоритм сортировки. В его основе лежит многократная перестановка соседних элементов, нарушающих порядок сортировки. Вот псевдокод этой сортировки:

```
Bubble_Sort(A):
  for i = 1 to A.length - 1
    for j = A.length downto i+1
      if A[j] < A[j-1]
        поменять A[j] и A[j-1] местами
```

Напишите код на Python и докажите корректность пузырьковой сортировки. Для доказательства корректоности процедуры вам необходимо доказать, что она завершается и что $A'[1] \leq A'[2] \leq ... \leq A'[n]$, где A' - выход процедуры Bubble_Sort, а n - длина массива A.

Определите время пузырьковой сортировки в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой.

Формат входного и выходного файла и ограничения - как в задаче 1.

Решение:

```python
def bubble_sort(n, numbers):
    for i in range(n):
        for j in range(i+1, n):
            if numbers[i] > numbers[j]:
                numbers[i], numbers[j] = numbers[j], numbers[i]
    return numbers


def check(numbers):
    for i in range(len(numbers)-1):
        if not(numbers[i] <= numbers[i+1]):
            return False
    return True


with open('input.txt', 'r') as file:
    n = int(file.readline())
    numbers = list(map(int, file.readline().split()))
file.close()

if 1 <= n <= 10**3:
    with open('output.txt', 'w') as file:
        file.write(' '.join(map(str, bubble_sort(n, numbers))))
    file.close()
else:
    print('веденные данные не соответствуют условию')
```

```
print(check(bubble_sort(n, numbers)))
```

1. Сначала опишем основную функцию - bubble_sort. На вход она принимает два параметра: n - число элементов массива и numbers - массив элементов. Далее производим пузырьковую сортировку данного массива: попарно сравниваем элементы и в случае, если правый меньше левого, меняем их местами. После выполнения этого алгоритма функция возвращает отсортированный список.
2. Для доказательства корректной работы алгоритма сортировки создадим функцию check, которая будет попарно сравнивать числа уже отсортированного массива. В случае, если число с меньшим индексом окажется больше числа с большим индексом, функция вернет False, иначе - True.
3. Открываем файл input.txt, содержащий входные данные. Читаем первую строку и приводим к целочисленному виду полученное число (n). Далее читаем вторую строку и преобразуем ее в список целых чисел (numbers). Закрываем файл.
4. Проверяем полученные данные на соответствие условию. Если они подходят, выполняем функцию bubble_sort(n, numbers), открываем файл output.txt, записываем в него результат работы программы - отсортированный массив numbers, после чего закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.
5. С помощью функции check проверяем, что массив отсортирован верно.

Результат работы программы на примере из текста задачи:
Входные данные:

```
6
31 41 59 26 41 58
```

Выходные данные:

```
26 31 41 41 58 59
```

```
C:\Users\User\AppData\Local\Programs\
True

Process finished with exit code 0
```

Тесты времени выполнения и затраченной памяти.

1. Пример из текста задачи

```
import time
import psutil
from lab1.task6.src.task6 import bubble_sort
```

```
test_nums1 = [31, 41, 59, 26, 41, 58]

start_time1 = time.perf_counter()
test1 = bubble_sort(len(test_nums1), test_nums1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 8.600007276982069e-06 секунд
Память: 15.20 Мбайт

Process finished with exit code 0
```

2. Минимальное значение

```
import time
import psutil
from lab1.task6.src.task6 import bubble_sort

test_nums2 = [0]

start_time2 = time.perf_counter()
test2 = bubble_sort(len(test_nums2), test_nums2)
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 4.800036549568176e-06 секунд
Память: 14.93 Мбайт

Process finished with exit code 0
```

3. Максимальное значение

```
import time
import psutil
import random
from lab1.task6.src.task6 import bubble_sort

test_nums2 = [random.randint(-10**9, 10**9+1) for _ in range(1,
10**3+1)]

start_time2 = time.perf_counter()
test2 = bubble_sort(len(test_nums2), test_nums2)
```

```
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 0.02479600004153326 секунд
Память: 14.91 Мбайт

Process finished with exit code 0
```

Сравнение пузырьковой и сортировки вставкой		
	Пузырьковая сортировка	Сортировка вставкой
Время выполнения в лучшем случае	3.00002284348011e-06 секунд	3.200024366378784e-06 секунд
Время выполнения в худшем случае	0.02474299998721108 секунд	0.022426300041843206 секунд

Вывод по задаче:

В ходе решения данной задачи мы научились осуществлять пузырьковую сортировку массива целых чисел, доказывать корректность ее работы и выяснили, каково время ее выполнения и затраты памяти на предельных и средних значениях.

Дополнительные задачи

Задание №2. Сортировка вставкой +

2 задача. Сортировка вставкой +

Измените процедуру Insertion-sort для сортировки таким образом, чтобы в выходном файле отображалось в первой строке n чисел, которые обозначают новый индекс элемента массива после обработки.

- **Формат выходного файла (input.txt).** В первой строке выходного файла выведите n чисел. При этом i-ое число равно индексу, на который, в момент обработки его сортировкой вставками, был перемещен i-ый элемент исходного массива. Индексы нумеруются, начиная с единицы. Между любыми двумя числами должен стоять ровно один пробел.

Пример.

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

В примере сортировка вставками работает следующим образом:

- Первый элемент остается на своем месте, поэтому первое число в ответе — единица. Отсортированная часть массива: [1]
- Второй элемент больше первого, поэтому он тоже остается на своем месте, и второе число в ответе — двойка. [1 8]
- Четверка меньше восьмерки, поэтому занимает второе место. [1 4 8]
- Двойка занимает второе место. [1 2 4 8]
- Тройка занимает третье место. [1 2 3 4 8]
- Семерка занимает пятое место. [1 2 3 4 7 8]
- Пятерка занимает пятое место. [1 2 3 4 5 7 8]
- Шестерка занимает шестое место. [1 2 3 4 5 6 7 8]
- Девятка занимает девятое место. [1 2 3 4 5 6 7 8 9]
- Ноль занимает первое место. [0 1 2 3 4 5 6 7 8 9]

Решение:

```
def insertion_sort(n, numbers):
    indexes = [1]
    for i in range(1, n):
        for j in range(i-1, -1, -1):
            if numbers[i] < numbers[j]:
                numbers[i], numbers[j] = numbers[j], numbers[i]
```

```
                i, j = j, i
        indexes.append(i+1)
    return numbers, indexes


with open('input.txt', 'r') as file:
    n = int(file.readline())
    numbers = list(map(int, file.readline().split()))
file.close()

result_numbers, result_indexes = insertion_sort(n, numbers)

if 1 <= n <= 10**3:
    with open('output.txt', 'w') as file:
        file.write(' '.join(map(str, result_indexes)))
        file.write('\n')
        file.write(' '.join(map(str, result_numbers)))
    file.close()
else:
    print('Введенные данные не соответствуют условию')
```

1. Для решения данной задачи изменим функцию insertion_sort, написанную в ходе выполнения задания 1. На вход она принимает два параметра: n - число элементов массива и numbers - массив элементов. Далее производим сортировку вставкой данного массива: попарно сравниваем элементы и в случае, если правый меньше левого, меняем их местами. При этом создаем список indexes, в который будем записывать новые индексы нужных нам элементов. Изначально в этом списке только один элемент - единица. После выполнения этого алгоритма функция возвращает отсортированный список, а также список индексов.
2. Открываем файл input.txt, содержащий входные данные. Читаем первую строку и приводим к целочисленному виду полученное число (n). Далее читаем вторую строку и преобразуем ее в список целых чисел (numbers). Закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, выполняем функцию bubble_sort(n, numbers), открываем файл output.txt, записываем в него результат работы программы - элементы списка indexes в первой строке и отсортированный массив numbers во второй, после чего закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Результат работы программы на примере из текста задачи:
Входные данные:

```
10
1 8 4 2 3 7 5 6 9 0
```

Выходные данные:

```
1 2 2 2 3 5 5 6 9 1
0 1 2 3 4 5 6 7 8 9
```

Тесты времени выполнения и затраченной памяти.

1. Пример из текста задачи

```
import time
import psutil
from lab1.task2.src.task2 import insertion_sort

test_nums1 = [1, 8, 4, 2, 3, 7, 5, 6, 9, 0]

start_time1 = time.perf_counter()
test1 = insertion_sort(len(test_nums1), test_nums1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 2.4300010409206152e-05 секунд
Память: 14.98 Мбайт

Process finished with exit code 0
```

2. Минимальное значение

```
import time
import psutil
from lab1.task2.src.task2 import insertion_sort

test_nums2 = [0]

start_time2 = time.perf_counter()
test2 = insertion_sort(len(test_nums2), test_nums2)
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 6.600050255656242e-06 секунд
Память: 14.80 Мбайт

Process finished with exit code 0
```

4. Максимальное значение

```
import time
import psutil
import random
from lab1.task2.src.task2 import insertion_sort

test_nums3 = [random.randint(-10**9, 10**9+1) for _ in range(1,
10**3+1)]

start_time3 = time.perf_counter()
test3 = insertion_sort(len(test_nums3), test_nums3)
print(f'Время: {time.perf_counter() - start_time3} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 0.047860999999102205 секунд
Память: 15.05 Мбайт

Process finished with exit code 0
```

Вывод по задаче:

В ходе решения данной задачи мы научились осуществлять сортировку вставкой массива целых чисел, выяснили, каково время ее выполнения и затраты памяти на предельных и средних значениях и разработали алгоритм по сохранению данных о новых индексах элементов при их обработке.

Задание №4. Линейный поиск

4 задача. Линейный поиск

Рассмотрим задачу поиска.

- **Формат входного файла**. Последовательность из n чисел $A = a_1, a_2, \ldots, a_n$ в первой строке, числа разделены пробелом, и значение V во второй строке. Ограничения: $0 \leq n \leq 10^3$, $-10^3 \leq a_i, V \leq 10^3$
- **Формат выходного файла**. Одно число - индекс i, такой, что $V = A[i]$, или значение -1, если V в отсутствует.
- Напишите код линейного поиска, при работе которого выполняется сканирование последовательности в поисках значения V.
- Если число встречается несколько раз, то выведите, сколько раз встречается число и все индексы i через запятую.
- Дополнительно: попробуйте найти свинью, как в лекции. Используйте во входном файле последовательность слов из лекции, и найдите соответствующий индекс.

Решение:

```
def linear_search(A, V):
    result = []
    for i in range(len(A)):
        if A[i] == V:
            result.append(i+1)
    counter = len(result)
    if counter == 0:
        return -1
    elif counter == 1:
        return result
    else:
        return counter, result


with open('input.txt', 'r') as file:
    A = list(map(int, file.readline().split()))
    V = int(file.readline())
file.close()

if (0 <= len(A) <= 10**3) and (-10**3 <= V <= 10**3):
    with open('output.txt', 'w') as file:
        file.write(', '.join(map(str, linear_search(A, V))))
    file.close()
else:
    print('Введенные данные не соответствуют условию')
```

1. Опишем основную функцию - linear_searcht. На вход она принимает два параметра: A - массив элементов и V - искомый элемент. Далее производим поиск элемента V в массиве A: для этого сравниваем каждый элемент массива с V. Если они равны, то в список result добавляем (i+1) - индекс найденного элемента, начиная с единицы. После завершения цикла присваиваем переменной counter значение длины списка result - это и будет количество элементов, равных V. В конце прописываем условия, согласно которым функция будет возвращать данные. Если элементов, равных V, не нашлось, возвращаем (-1); если нашелся всего один элемент, возвращаем его индекс; если же таких элементов найдено несколько, возвращаем их количество и список их индексов.
2. Открываем файл input.txt, содержащий входные данные. Читаем первую строку и преобразуем ее в список целых чисел A, далее читаем вторую строку и приводим к целочисленному типу искомый элемент V. Закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, выполняем функцию linear_search(A, V), открываем файл output.txt и записываем в него результат работы программы. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Результат работы программы:

Входные данные:

```
1 2 2 3 4 15 7
2
```

Выходные данные:

```
2, [2, 3]
```

Тесты времени выполнения и затраченной памяти.

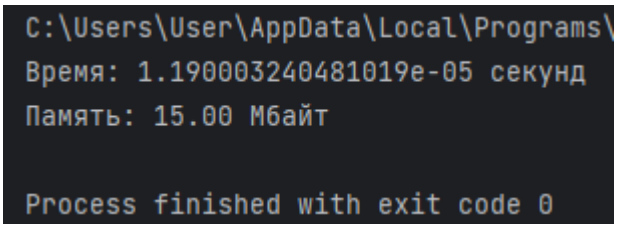
1. Пример

```
import time
import psutil
from lab1.task4.src.task4 import linear_search

test_nums1 = [1, 2, 2, 3, 4, 15, 7]
test_v = 2

start_time1 = time.perf_counter()
test1 = linear_search(test_nums1, test_v)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f} Мбайт')
```

Результат:



Вывод по задаче:

В ходе решения данной задачи мы научились осуществлять линейный поиск некоторого элемента среди массива элементов и выяснили, каково время его выполнения и затраты памяти.

Задание №5. Сортировка выбором

5 задача. Сортировка выбором.

Рассмотрим сортировку элементов массива , которая выполняется следующим образом. Сначала определяется наименьший элемент массива , который ставится на место элемента $A[1]$. Затем производится поиск второго наименьшего элемента массива A, который ставится на место элемента $A[2]$. Этот процесс продолжается для первых $n-1$ элементов массива A.

Напишите код этого алгоритма, также известного как сортировка выбором (selection sort). Определите время сортировки выбором в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой.

Формат входного и выходного файла и ограничения - как в задаче 1.

Решение:

```
def selection_sort(n, numbers):
    for i in range(n):
        minimum = numbers[i]
        index = i
        for j in range(i+1, n):
            if numbers[j] <= minimum:
                minimum = min(minimum, numbers[j])
                index = j
        numbers.pop(index)
        numbers.insert(i, minimum)

    return numbers


with open('input.txt', 'r') as file:
    n = int(file.readline())
    numbers = list(map(int, file.readline().split()))
file.close()

if 1 <= n <= 10**3:
    with open('output.txt', 'w') as file:
        file.write(' '.join(map(str, selection_sort(n, numbers))))
    file.close()
else:
    print('Введенные данные не соответствуют условию')
```

1. Сначала опишем основную функцию - selection_sort. На вход она принимает два параметра: n - число элементов массива и numbers - массив элементов. Далее производим сортировку выбором данного массива: попарно сравниваем элементы, сохраняя минимальный из них в переменную minimum и переменной index присваиваем значение индекса минимального элемента. Следующим шагом с помощью метода pop удаляем и возвращаем элемент с индексом

index и добавляем на место с индексом i элемент minimum, используя метод insert. После выполнения этого алгоритма функция возвращает отсортированный список.

2. Открываем файл input.txt, содержащий входные данные. Читаем первую строку и приводим к целочисленному виду полученное число (n). Далее читаем вторую строку и преобразуем ее в список целых чисел (numbers). Закрываем файл.
3. Проверяем полученные данные на соответствие условию. Если они подходят, выполняем функцию selection_sort(n, numbers), открываем файл output.txt, записываем в него результат работы программы - отсортированный массив numbers, после чего закрываем файл. В случае, когда введенные данные не соответствуют ограничению, выводим ошибку.

Результат работы программы на примере из текста задачи:
Входные данные:

```
6
31 41 59 26 41 58
```

Выходные данные:

```
26 31 41 41 58 59
```

Тесты времени выполнения и затраченной памяти.

1. Пример из текста задачи

```
import time
import psutil
from lab1.task5.src.task5 import selection_sort

test_nums1 = [31, 41, 59, 26, 41, 58]

start_time1 = time.perf_counter()
test1 = selection_sort(len(test_nums1), test_nums1)
print(f'Время: {time.perf_counter() - start_time1} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 1.0600022505968809e-05 секунд
Память: 15.01 Мбайт

Process finished with exit code 0
```

2. Минимальное значение

```
import time
import psutil
```

```
from lab1.task5.src.task5 import selection_sort

test_nums2 = [0]

start_time2 = time.perf_counter()
test2 = selection_sort(len(test_nums2), test_nums2)
print(f'Время: {time.perf_counter() - start_time2} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 3.8000289350748062e-06 секунд
Память: 14.74 Мбайт

Process finished with exit code 0
```

3. Максимальное значение

```
import time
import psutil
import random
from lab1.task5.src.task5 import selection_sort

test_nums3 = [random.randint(-10**9, 10**9+1) for _ in range(1,
10**3+1)]

start_time3 = time.perf_counter()
test2 = selection_sort(len(test_nums3), test_nums3)
print(f'Время: {time.perf_counter() - start_time3} секунд')
print(f'Память: {psutil.Process().memory_info().rss / 1024**2:.2f}
Мбайт')
```

Результат:

```
C:\Users\User\AppData\Local\Programs\
Время: 0.01377299998421222 секунд
Память: 15.05 Мбайт

Process finished with exit code 0
```

Сравнение сортировки выбором и сортировки вставкой		
	Сортировка выбором	Сортировка вставкой
Время выполнения в лучшем случае	3.89997148886323e-06 секунд	3.200024366378784e-06 секунд

Время выполнения в худшем случае	0.0193365000304766 секунд	0.022426300041843206 секунд

Вывод по задаче:

В ходе решения данной задачи мы научились осуществлять сортировку выбором массива целых чисел и выяснили, каково время ее выполнения и затраты памяти на предельных и средних значениях.

Вывод

В ходе выполнения лабораторной работы №1 мы изучили сортировку вставкой, выбором, пузырьковую, сортировку вставкой по убыванию, а также научились осуществлять линейный поиск. Помимо этого протестировали время работы данных алгоритмов и затрачиваемую на их выполнение память и выяснили, что по обоим параметрам они являютмся не самыми оптимальными.