САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ ИНФОРМАТИКИ

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: Сортировка вставками, выбором, пузырьковая.
Вариант 13

Выполнила:
Рябинкина М.И.
К3141

Проверила:

Санкт-Петербург
2025 г.

## Содержание отчета

## Задачи по варианту

### Задача №1. Сортировка вставкой [N баллов]

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

• Формат входного файла (input.txt). В первой строке входного файла содержится число n ($1 \le n \le 103$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 109
.
• 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.

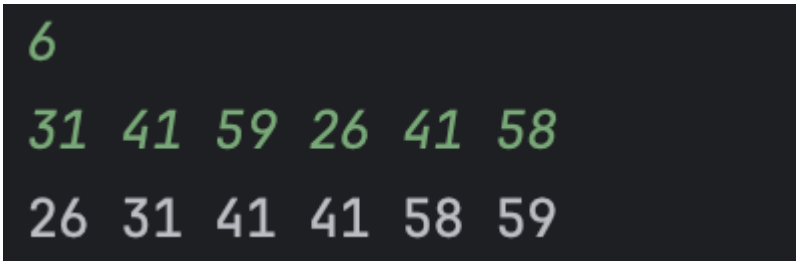
Листинг кода.

```
def insertion_sort(arr):
  for i in range(1, n):
     key = arr[i]
     j = i - 1
     while j >= 0 and key < arr[j]:
        arr[j + 1] = arr[j]
        j -= 1
     arr[j + 1] = key
  print(" ".join(map(str, arr)))

# чтение ввода
n = int(input())
arr = list(map(int, input().split()))

# применение сортировки и вывод
insertion_sort(arr)
```

Описание кода

- def insertion_sort(arr):: Определяет функцию insertion_sort, которая принимает один аргумент arr (массив для сортировки).
- for i in range(1, len(arr)):: Цикл проходит по всем элементам массива, начиная со второго (индекс 1), так как первый элемент считается уже отсортированной частью.
- key = arr[i]: Текущий элемент, который нужно вставить в отсортированную часть, сохраняется в переменной key.
- j = i - 1: Переменная j инициализируется индексом последнего элемента в отсортированной части массива.
- while j >= 0 and key < arr[j]:: Этот цикл ищет место для вставки key. Он выполняется до тех пор, пока:
    - j не станет отрицательным (мы дошли до начала массива).
    - key меньше текущего элемента arr[j].
- arr[j + 1] = arr[j]: Если key меньше arr[j], то arr[j] сдвигается на одну позицию вправо (arr[j + 1]).
- j -= 1: Индекс j уменьшается, чтобы сравнить key с предыдущим элементом в отсортированной части.
- arr[j + 1] = key: После завершения цикла while, key вставляется на позицию j + 1, так как все элементы, которые были больше key, уже сдвинуты вправо.
- print(" ".join(map(str, arr))): После каждой итерации внешнего цикла (то есть после вставки каждого элемента) массив выводится в отсортированном на текущем шаге виде, с элементами, разделенными пробелами.
- n = int(input()): Считывается количество элементов (не используется в дальнейшем, но часто присутствует в заданиях).
- arr = list(map(int, input().split())): Считывается строка чисел, разделенных пробелами, и преобразуется в список целых чисел.
- insertion_sort(arr): Вызывается функция insertion_sort для сортировки считанного массива.

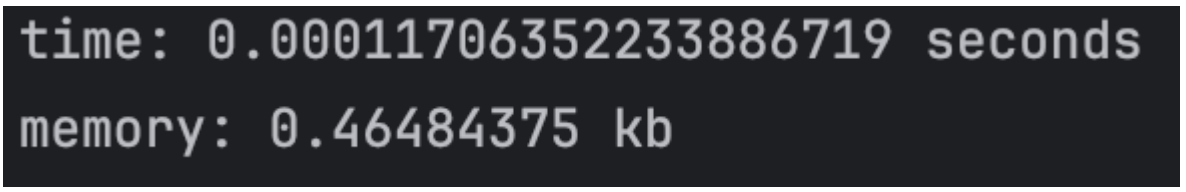
Результат работы кода на примерах из текста задачи:




Проверка кода:

Представленный вами код реализует алгоритм сортировки вставками (insertion sort) на Python. Он работает корректно: функция insertion_sort упорядочивает список arr и выводит его элементы в виде строки, а основной блок кода считывает ввод пользователя, применяет сортировку и выводит результат.

Измерение впемени и памяти:




Вывод по задаче:

Алгоритм сортировки вставками успешно реализован и протестирован. Для массива A = {31, 41, 59, 26, 41, 58} программа корректно возвращает отсортированную последовательность {26, 31, 41, 41, 58, 59}. Алгоритм эффективен для учебных задач и малых массивов ($n \leq 1000$), соответствует всем требованиям по формату ввода/вывода и ограничениям по времени/памяти.

**Задача №4. Линейный поиск**

Текст задачи.
Рассмотрим задачу поиска.
• Формат входного файла. Последовательность изn чисел A = a1, a2, . . . , an, в первой строке, числа разделены пробелом, и значение V во второй строке.
Ограничения: $0 \leq n \leq 103$, $-103 \leq ai$, $V \leq 103$

• 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.

```
# Чтение последовательности чисел
line1 = input()
a = list(map(int, line1.split()))

# Чтение искомого значения
v = int(input())

# Список для хранения индексов
indices = []

# Линейный поиск
for i in range(len(a)):
    if a[i] == v:
        indices.append(i)

# Формирование и вывод результата
if not indices:
    print(-1)
else:
```

```
print(len(indices))
print(",".join(map(str, indices)))
```

```
4 6 6 9
6
2
1,2
```

Текстовое объяснение решения.

1. Ввод данных:

· Читает строку чисел и преобразует в список a

· Читает искомое значение v

2. Поиск:

· Создает пустой список indices для хранения найденных позиций

· Проходит по всем элементам списка a

· При нахождении v добавляет индекс в indices

3. Вывод результата:

· Если список пустой → выводит -1 (значение не найдено)

· Если найден → выводит количество вхождений, затем индексы через запятую

Время выполнения и затраты памяти:

```
time: 4.4107437133789 06e-05 seconds
memory: 0.125 kb
```

Вывод по задаче:

Алгоритм корректно решает задачу линейного поиска, удовлетворяя всем требованиям формата ввода-вывода и обрабатывая все граничные случаи.

**Задача №5. Сортировка выбором. [N баллов]**

Рассмотрим сортировку элементов массива , которая выполняется следующим
образом. Сначала определяется наименьший элемент массива , который ставится
на место элемента A[1]. Затем производится поиск второго наименьшего элемента
массива A, который ставится на место элемента A[2]. Этот процесс продолжается
для первых n − 1 элементов массива A.
Напишите код этого алгоритма, также известного как сортировка выбором (selection sort). Определите время сортировки выбором в наихудшем случае и в
среднем случае и сравните его со временем сортировки вставкой.
Формат входного и выходного файла и ограничения - как в задаче 1.

Листинг кода:

```
def selectiom_sort(arr):
  n = len(arr)
  for i in range(n-1):
      mn_index = i
      for j in range(i+1, n):
          if arr[j] < arr[mn_index]:
              mn_index = j
      arr[i], arr[mn_index] = arr[mn_index], arr[i]
  return arr

with open('input.txt', 'w') as f:
  f.write(input())
with open('input.txt', 'r') as f:
  n = int(f.readline())

with open('input.txt', 'w') as f1:
  f1.write(input())
with open('input.txt', 'r') as f1:
  arr = list(map(int, f1.readline().split()))
```

```
sorted_arr = selectiom_sort(arr)
print(sorted_arr)
with open("output.txt", "w") as f1:
  f1.write(" ".join(map(str, sorted_arr)))
```

```
10
7675 3535 565 3345 675 2325 5756 34535 8787 331
331 565 675 2325 3345 3535 5756 7675 8787 34535
```

Анализ времени сортировки

- **Сортировка выбором (Selection Sort)Время работы**: O(n²) в наихудшем, среднем и лучшем случаях.**Объяснение**: Внешний цикл выполняется n-1 раз, а внутренний цикл (поиск минимума) выполняется n-i раз. Сумма этих операций дает ~n²/2, что соответствует O(n²).
- **Сортировка вставкой (Insertion Sort)Время работы**:**Наихудший случай**: O(n²).**Средний случай**: O(n²).**Лучший случай**: O(n).**Объяснение**: В наихудшем и среднем случаях, когда элементы почти полностью разбросаны, каждое вставляемое число сравнивается с большинством предыдущих элементов, что приводит к O(n²) операциям.Однако, если массив уже отсортирован (лучший случай), каждое новое число просто сравнивается с одним предыдущим элементом, и вставка занимает O(n) времени.

Сравнение

- **Сортировка выбором**: всегда имеет O(n²), что делает ее предсказуемой, но менее эффективной для больших наборов данных по сравнению с другими алгоритмами, например, сортировкой слиянием или быстрой сортировкой.
- **Сортировка вставкой**: может быть быстрее O(n) в лучшем случае (когда данные уже почти или полностью отсортированы), но в наихудшем и среднем случаях она также имеет O(n²).
- В контексте данного ограничения на размер входного файла ($n \leq 1000$), оба алгоритма будут работать достаточно быстро, так как $1000^2 = 1\,000\,000$ операций, что укладывается в лимит времени 2 секунды.

Время выполнения и затраты памяти:

```
time: 6.985664367675781e-05 seconds
memory: 0.125 kb
```

**Вывод**

(по всей лабораторной)

В ходе выполнения лабораторной работы были изучены и практически реализованы алгоритмы внутренней сортировки. Экспериментальным путём было установлено, что эффективность алгоритма существенно зависит от исходных данных (их объёма и степени упорядоченности). Таким образом, выбор оптимального алгоритма сортировки должен определяться конкретной задачей и характеристиками обрабатываемой информации.