ЗАДАЧА №6. (АЛГОРИТМ СОРТИРОВКИ ПУЗЫРЬКОМ)

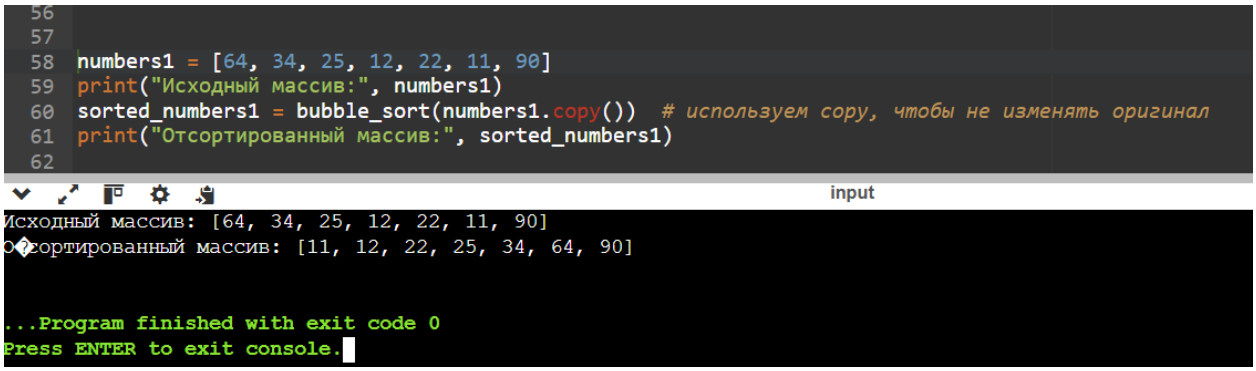
Алгоритм сортировки пузырьком (bubble sort) с оптимизацией (ранний выход, если массив уже отсортирован): a. напишите функцию с подробными комментариями; b. приведите пример вызова функции и результат; c. проанализируйте асимптотическую сложность по времени и памяти (лучший, средний, худший случаи).

Код программы:

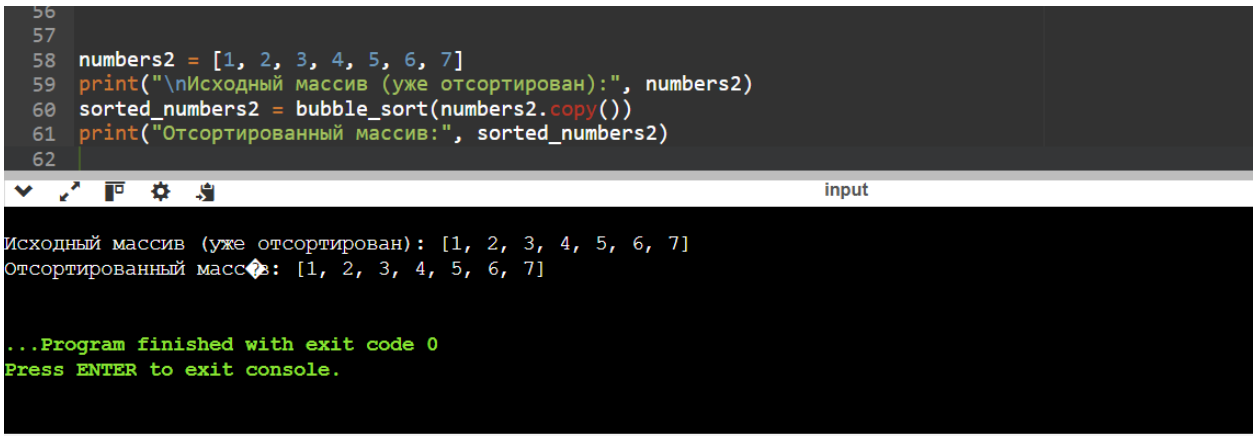
```
def bubble_sort(arr):
    """
    Сортировка пузырьком с оптимизацией (ранний выход).

    Алгоритм проходит по массиву, сравнивая соседние элементы
    и меняя их местами, если они стоят в неправильном порядке.
    Оптимизация: если за проход не было ни одной замены,
    массив уже отсортирован и можно завершить досрочно.
    """
    # Получаем длину массива
    n = len(arr)

    # Внешний цикл - количество проходов
    for i in range(n - 1):
        # Флаг, отслеживающий были ли замены в текущем проходе
        swapped = False

        # Внутренний цикл - сравнение соседних элементов
        # С каждым проходом самый большой элемент "всплывает" в конец,
        # поэтому можно уменьшать количество сравнений
        for j in range(n - 1 - i):
            # Сравниваем соседние элементы
            if arr[j] > arr[j + 1]:
                # Меняем элементы местами
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                # Отмечаем, что замена произошла
                swapped = True

        # Если замен не было, массив уже отсортирован
        if not swapped:
            break

    return arr
```

```
def bubble_sort_desc(arr):
    """
    Сортировка пузырьком по убыванию с оптимизацией.
    """
    n = len(arr)

    for i in range(n - 1):
        swapped = False

        for j in range(n - 1 - i):
            # Меняем знак сравнения для сортировки по убыванию
            if arr[j] < arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True

        if not swapped:
            break

    return arr
```

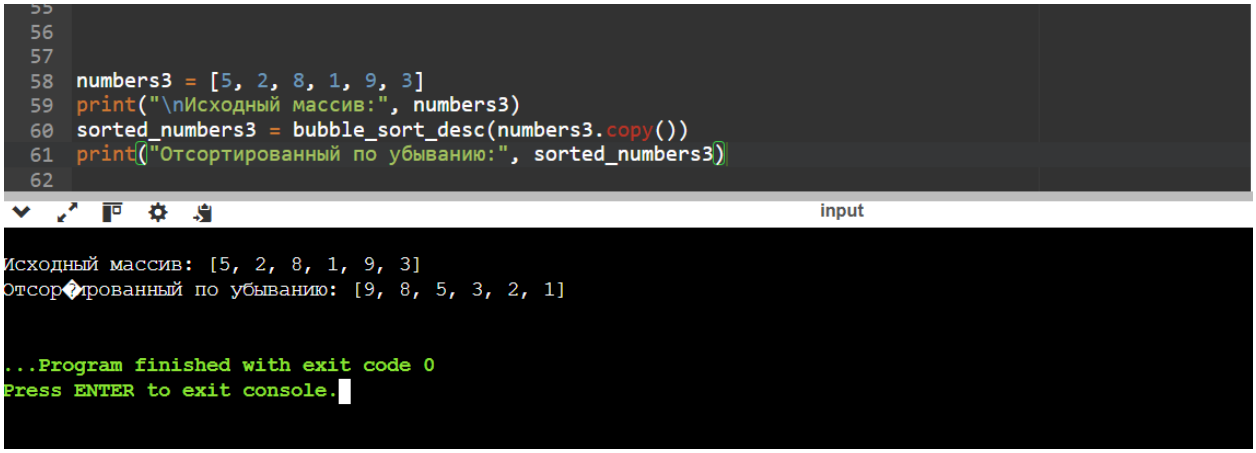
На рисунке 6 представлен результат работы программы с неотсортированным массивом:



На рисунке 7 представлен результат работы программы с отсортированным массивом по возрастанию:



На рисунке 8 представлен результат работы программы с отсортированным массивом по убыванию:



Временная сложность

Лучший случай: O(n) так как Массив уже отсортирован. Благодаря оптимизации (флаг swapped) алгоритм делает один проход (n-1 сравнений) и завершается.

Средний случай: $O(n^2)$ В среднем требуется около $n^2/2$ сравнений и $n^2/2$ обменов.

Худший случай: $O(n^2)$ Массив отсортирован в обратном порядке. Требуется n-1 проходов, каждый проход выполняет n-i-1 сравнений, всего $\sim n^2/2$ сравнений и обменов.

Пространственная сложность

Все случаи O(1) Алгоритм сортирует массив на месте (in-place), используя только фиксированное количество дополнительной памяти (переменные i, j, swapped, временная переменная для обмена).

Преимущества и недостатки

Преимущества:

Простота реализации и понимания.

Сортирует массив на месте (не требует дополнительной памяти).

Стабильный алгоритм (сохраняет порядок равных элементов).

С оптимизацией эффективен для почти отсортированных массивов.

Недостатки:

Низкая эффективность для больших массивов ($O(n^2)$).

Слишком медленный для практического использования с большими данными.