ЗАДАЧА №7. (АЛГОРИТМ СОРТИРОВКИ ВЫБОРОМ)

Алгоритм сортировки выбором (selection sort): 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 selection_sort(arr):
    """
    Сортировка выбором (selection sort).

    Алгоритм на каждом шаге находит минимальный элемент в неотсортированной части
    и меняет его с первым элементом этой части.
    """
    n = len(arr)

    for i in range(n):
        # Ищем индекс минимального элемента в неотсортированной части
        min_index = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j

        # Меняем местами текущий элемент с найденным минимальным
        arr[i], arr[min_index] = arr[min_index], arr[i]

    return arr
```

На рисунке 9 представлен результат работы программы:

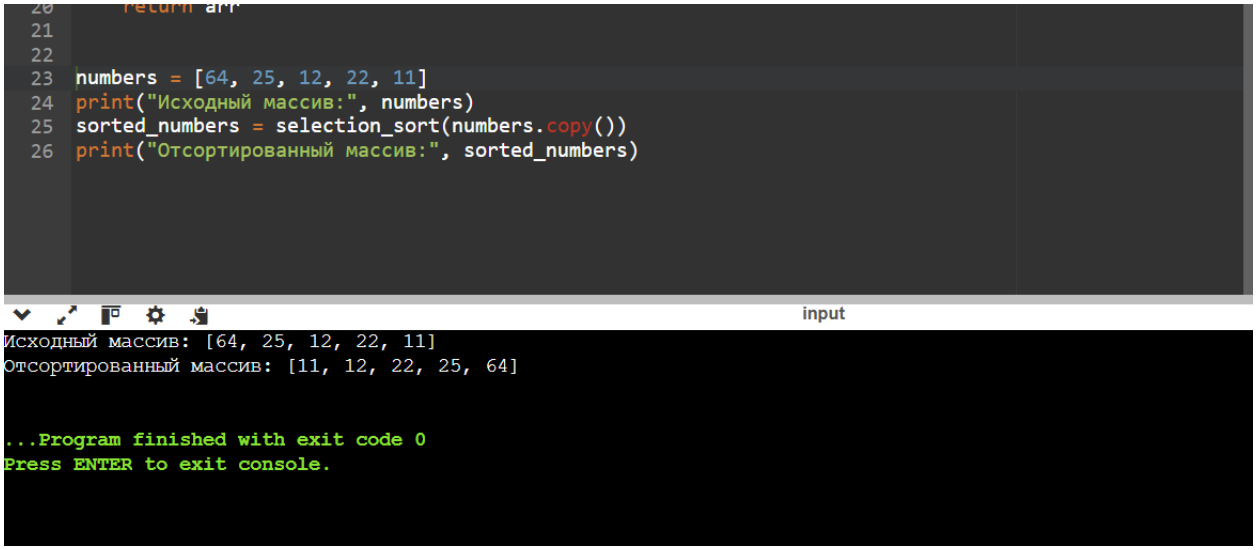

Рисунок 9 – Результат выполнения программы

Временная сложность

Лучший случай: $O(n^2)$ даже если массив уже отсортирован, алгоритм все равно выполняет все сравнения.

Средний случай: $O(n^2)$ всегда $n(n-1)/2$ сравнений.

Худший случай: $O(n^2)$ то же количество сравнений.

Пространственная сложность

Память: $O(1)$ сортировка выполняется на месте, дополнительные переменные не требуют дополнительной памяти.

Преимущества и недостатки

Преимущества:

Простота реализации легко понять и написать.

Минимальное количество обменов не более n-1 перестановок.

Работает на месте не требует дополнительной памяти $O(1)$.

Хорошо предсказуемо время выполнения всегда одинаково для данного размера массива.

Эффективен для маленьких массивов.

Недостатки:

Всегда выполняет $O(n^2)$ сравнений даже для уже отсортированного массива.

Неэффективен для больших массивов квадратичная сложность.

Нестабильный может изменить порядок равных элементов.

Нет оптимизации для лучшего случая в отличие от пузырьковой сортировки.