**Лабораторная работа**

**ИССЛЕДОВАНИЕ И ОЦЕНКА АЛГОРИТМОВ СОРТИРОВКИ**

**Цель работы**. Разработка программ, реализующих различные рекурсивные алгоритмы, и оценка их временной и пространственной сложности.

**Задание.** Разработка программ, реализующих различные алгоритмы сортировки, и оценка их временной и пространственной сложности.

**Пузырьковая сортировка:**

Этот метод - наиболее распространенный и простой. Он требует минимального объема памяти для данных, но затраты времени на его реализацию велики. При упорядочении выполняются следующие операции:

1) элементы массива сравниваются *попарно*: первое со вторым; второе с третьим; *i*-тое – с (*i*+1) - вым;

2) если они стоят неправильно (при *упорядочении* по возрастанию первый должен быть меньше второго или равен ему), то элементы меняются местами.

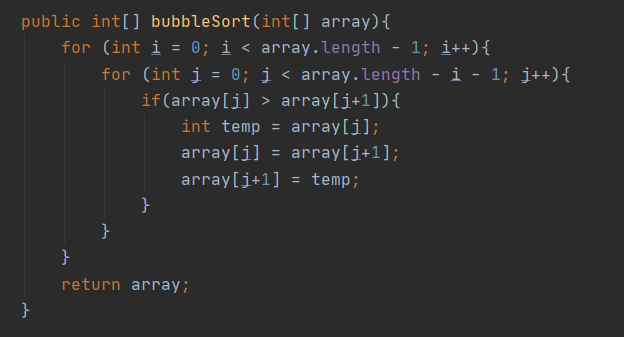
**Сортировка Шейкером:**

Если данные сортируются не в оперативной памяти, а на жестком диске, то количество перемещений элементов существенно влияет на время работы. Этот алгоритм уменьшает количество таких перемещений. За один проход из всех элементов выбираются минимальный и максимальный. Потом минимальный элемент помещается в начало массива, а максимальный - в конец. Таким образом, за каждый проход два элемента оказываются на своих местах, поэтому понадобится *n*/2 проходов, где *n* — количество элементов. 38

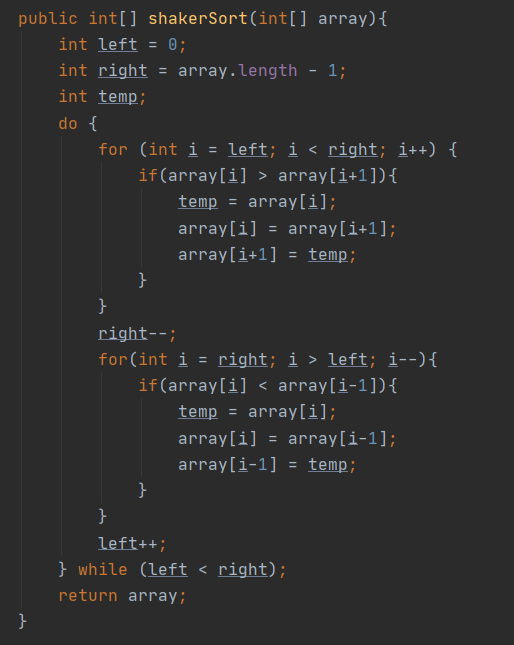
На каждом проходе массив просматривается слева направо до середины, а потом – наоборот (справа налево). В результате, как при сортировке «пузырьком», при просмотре слева направо максимальный элемент попадает на свое место, а при обратном – минимальный.

**Код программы.**

**Пузырьковая сортировка**



**Сортировка Шейкером**



**Формулы верхней оценки временной и емкостной сложности заданных алгоритмов.**

**Сортировка пузырьком**

Верхняя оценка сложности алгоритма: О(*n^2*)

Ёмкостная сложность алгоритма: О(1)

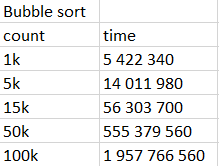
**Сортировка Шейкером**

Верхняя оценка сложности алгоритма: О(*n^2*)

Ёмкостная сложность алгоритма: О(1)

**Результаты экспериментальной оценки временной и емкостной сложности заданного алгоритма**.

**Bubble sort**



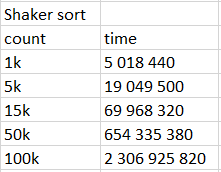
int i = 4 bytes

int j = 4 bytes

int temp = 4 bytes

4 + 4 + 4 = 12 => O(1)

**Shaker sort**



int left = 4 bytes

int right = 4 bytes

int temp = 4 bytes

2 \* int i = 8 bytes

4 + 4 + 4 + 8 = 20 => O(1)