Инвариант цикла while внутри алгоритма шейкерной сортировки:

Индексы l-1 и r указывают на текущие левую и правую границы отсортированного массива в начале итерации каждого цикла **for** соответственно. Индексы right и left-1 указывают на текущие левую и правую границы отсортированного массива внутри цикла **while**. В начале каждой итерации цикла **while** с индексами l и r массив **a** состоит из трёх частей:

- Элементы a[0..left-1] соответствуют отсортированным элементам левой части массива;
- Элементы a[left..right-1] соответствуют неотсортированным элементам массива;
- Элементы a[right..N-1] соответствуют отсортированным элементам правой части массива.

Сформулирую инвариант цикла:

*“В начале каждой итерации обоих циклов **for** внутри цикла **while** подмассивы a[0..left-1] и a[right..N-1] состоят из элементов, которые изначально находились в a[0..left-1] и a[right..N-1] соответственно, но теперь расположены в отсортированном порядке.».*

Доказательство инварианта цикла while в алгоритме шейкерной сортировки:

Инициализация. Перед первыми итерациями циклов **for** *a[0..l-1] и a[r..N-1]* содержат по одному элементу, значит можно считать, что они отсортированы.

Сохранение. В теле первого цикла **for** происходит сдвиг элементов вправо до тех пор, пока позицию *a[right]* не займёт тот элемент, который должен там стоять. В теле второго цикла **for** происходит сдвиг элементов влево до тех пор, пока позицию *a[left-1]* не займёт тот элемент, который должен там стоять.

Завершение. Цикл **while** завершится в тот момент, когда right будет меньше left. Из-за этого условия подмассивы *a[0..left-1] и a[right..N-1]* будут иметь общие элементы, следовательно их можно рассматривать как один массив. Так как оба подмассива будут отсортированы, значит и весь массив будет отсортирован.

Сложность алгоритма шейкерной сортировки:

while (left <= right)		n
{		
for (int l = left;l <= right;l++)		$\sum_{l=} t_l$
{		
if (a[l-1] > a[l])		
swap(a[l-1], a[l])		
}		
right--;		n-1
for (int r = right;r >= left;r--)		
{		
if (a[r-1] > a[r])		
swap(a[r-1], a[r])		
}		
left++;		n-1
}		