Quicksort Algorithm Implementation

*Firsly there was a problem with the pseudo code, in the last line before the return statement in the partition method, it should say to swap pi with end instead of i.

Quicksort implementation:

```
48    private static int[] quickSort(int arr[], int start, int end)
49    {
50        if (start < end)
51        {
52            int piv = partition(arr, start, end);
53
54            //divide into two, before and after pivot
55            quickSort(arr, start, piv - 1); //swap the first element with the
56            quickSort(arr, piv + 1, end); // swap the element after the pivot
57        }
58        return arr;
59 }
```

Partition implementation

```
17    private static int partition (int arr[], int start, int end)
18    {
19        int pivot = arr[end], pi = start;
20
21        for (int i = start; i <= end - 1; i++){
22
23            // If current element is smaller than the pivot
24            if (arr[i] <= pivot)
25            {
26                swap(arr,pi,i);// swap pi and i
27                pi++; // increment index of smaller element
28            }
29        }
30        swap(arr,pi, end);//swap pi and end
31        return pi;
32    }
```

Shuffle implementation to prevent against worst case scenario;

```
    private static void shuffle(int[] arr)
    {
        for (int i=1; i < arr.length; i++)
        swap(arr, i, (int)(Math.random() * i));
    }
```

I copy pasted my quicksort from last week into this week. This is how I implemented shuffle and insertion sort into the quicksort.

```
47
48      private static int[] quickSort(int arr[], int start, int end)
49      {
50          shuffle(arr);
51
52          if(arr.length <= 10) {
53              insertionSort(arr);
54          }
55          else {
56          if (start < end)
57          {
58              int piv = partition(arr, start, end);
59
60              //divide into two, before and after pivot
61              quickSort(arr, start, piv - 1); //swap the first element with th
62              quickSort(arr, piv + 1, end); // swap the element after the pivo
63          }
64      }
65          return arr;
66  }
```

This probably isn't a great implementation of the median of 3 but it works.

```
6
7       private static int partition (int arr[], int start, int end)
8       {
9           int pivot = arr[end], middle = arr[((end - start) / 2) + start]; // pivot
0
1           if(((arr[end] >= arr[start]) && (arr[end] <= arr[middle]))
2                   || ((arr[end] <= arr[start]) && (arr[end] >= arr[middle]))){
3               pivot = arr[end]; //if end is the median, make it the pivot
4           }
5           else if(((arr[middle] >= arr[start]) && (arr[middle] <= arr[end]))
6                   || ((arr[middle] <= arr[start]) && (arr[middle] >= arr[end]))) {
7               pivot = arr[middle]; //if middle is the median, make it the pivot
8           }
9           else {
0               pivot = arr[start]; //then start must be the median, so make it the pivot
1           }
2
```

The way I did it was take one of them and see if it is in the middle. If its both smaller or equal than one, and greater or equal than the other OR the median but the other two are swapped around than it is chosen as the pivot, it tests this with the next one as well and if not it chooses the last one.

Time complexity Analysis:

	10	100	1000	10000
Quicksort	0.0	0.0	1.0	2.0
Enhanced Quicksort	0.0	0.0	0.0	1.0