Algorithm Design and Analysis

Assignment 1

by Dimo Hristov Dimchev

Tests

For the tests of the first tree files I have included the comparison counter and the sorted and unsorted array of integers. As for the remaining 3 I have included just the comparison counter, because the arrays become too big to be displayed here.

Tests for Insertion Sort:

Test1

```
Values in test1.txt
029 081 089 050 038 060 091 088 064 035 052 090 065 061 009
065 053 017 074 004 014 040 047 071 064 078 095 006 037 064
026 007 060 046 043 099 002 076 033 013 080 067 089 065 026
012 062 064 019 012
```

Insertion sort comparison counter: 732

```
Values in test1.txt
002 004 006 007 009 012 012 013 014 017 019 026 026 029 033
035 037 038 040 043 046 047 050 052 053 060 060 061 062 064
064 064 064 065 065 065 067 071 074 076 078 080 081 088 089
089 090 091 095 099
```

Test2

```
Values in test2.txt
007 007 007 007 007 007 014 014 014 014 014 014 014 026 029
029 029 029 029 029 035 035 035 037 040 040 040 050 050 050
053 060 065 065 065 065 065 065 065 065 065 065 074 074 088
088 088 091 091 091
```

Insertion sort comparison counter: 49

```
Values in test2.txt
007 007 007 007 007 007 014 014 014 014 014 014 014 026 029
```

```
029 029 029 029 029 035 035 035 037 040 040 040 050 050 050
053 060 065 065 065 065 065 065 065 065 065 065 074 074 088
088 088 091 091 091
```

Test3

```
Values in test3.txt
029 088 050 035 088 065 091 088 065 035 050 091 065 014 065
053 014 074 040 050 040 065 014 029 037 065 029 007 060 040
065 014 091 007 074 029 014 007 065 007 065 026 029 065 007
029 014 007 014 035
```

Insertion sort comparison counter: 799

```
Values in test1.txt
007 007 007 007 007 007 014 014 014 014 014 014 014 026 029
029 029 029 029 029 035 035 035 037 040 040 040 050 050 050
053 060 065 065 065 065 065 065 065 065 065 065 074 074 088
088 088 091 091 091
```

Test4

Insertion sort comparison counter: 244286

Test5

Insertion sort comparison counter: 3616

Test6

Insertion sort comparison counter: 250455

Tests for Quicksort:

Test1

```
Values in test1.txt
029 081 089 050 038 060 091 088 064 035 052 090 065 061 009
```

```
065 053 017 074 004 014 040 047 071 064 078 095 006 037 064
026 007 060 046 043 099 002 076 033 013 080 067 089 065 026
012 062 064 019 012
```

Quicksort comparison counter: 412

```
Values in test1.txt
002 004 006 007 009 012 012 013 014 017 019 026 026 029 033
035 037 038 040 043 046 047 050 052 053 060 060 061 062 064
064 064 064 065 065 065 067 071 074 076 078 080 081 088 089
089 090 091 095 099
```

Test2

```
Values in test2.txt
007 007 007 007 007 007 014 014 014 014 014 014 014 026 029
029 029 029 029 029 035 035 035 037 040 040 040 050 050 050
053 060 065 065 065 065 065 065 065 065 065 065 074 074 088
088 088 091 091 091
```

Quicksort comparison counter: 558

```
Values in test2.txt
007 007 007 007 007 007 014 014 014 014 014 014 014 026 029
029 029 029 029 029 035 035 035 037 040 040 040 050 050 050
053 060 065 065 065 065 065 065 065 065 065 065 074 074 088
088 088 091 091 091
```

Test3

```
Values in test3.txt
029 088 050 035 088 065 091 088 065 035 050 091 065 014 065
053 014 074 040 050 040 065 014 029 037 065 029 007 060 040
065 014 091 007 074 029 014 007 065 007 065 026 029 065 007
029 014 007 014 035
```

Quicksort comparison counter: 454

```
Values in test3.txt
```

```
007 007 007 007 007 007 014 014 014 014 014 014 014 026 029
029 029 029 029 029 035 035 035 037 040 040 040 050 050 050
053 060 065 065 065 065 065 065 065 065 065 065 074 074 088
088 088 091 091 091
```

Test4

Quicksort comparison counter: 15767

Test5

Quicksort comparison counter: 223565

Test6

Quicksort comparison counter: 17366

Tests for New Sort

Test1

```
Values in test1.txt
029 081 089 050 038 060 091 088 064 035 052 090 065 061 009
065 053 017 074 004 014 040 047 071 064 078 095 006 037 064
026 007 060 046 043 099 002 076 033 013 080 067 089 065 026
012 062 064 019 012
```

New Sort comparison counter: 2111

```
Values in test1.txt
002 004 006 007 009 012 012 013 014 017 019 026 026 029 033
035 037 038 040 043 046 047 050 052 053 060 060 061 062 064
064 064 064 065 065 065 067 071 074 076 078 080 081 088 089
089 090 091 095 099
```

Test2

```
Values in test2.txt
007 007 007 007 007 007 014 014 014 014 014 014 014 026 029
029 029 029 029 029 035 035 035 037 040 040 040 050 050 050
```

```
053 060 065 065 065 065 065 065 065 065 065 065 074 074 088
088 088 091 091 091
```

New Sort comparison counter: 680

```
Values in test2.txt
007 007 007 007 007 007 014 014 014 014 014 014 014 026 029
029 029 029 029 029 035 035 035 037 040 040 040 050 050 050
053 060 065 065 065 065 065 065 065 065 065 065 074 074 088
088 088 091 091 091
```

Test3

```
Values in test1.txt
029 088 050 035 088 065 091 088 065 035 050 091 065 014 065
053 014 074 040 050 040 065 014 029 037 065 029 007 060 040
065 014 091 007 074 029 014 007 065 007 065 026 029 065 007
029 014 007 014 035
```

New Sort comparison counter: 680

```
Values in test3.txt
007 007 007 007 007 007 014 014 014 014 014 014 014 026 029
029 029 029 029 029 035 035 035 037 040 040 040 050 050 050
053 060 065 065 065 065 065 065 065 065 065 065 074 074 088
088 088 091 091 091
```

Test4

New Sort comparison counter: 811677

Test5

New Sort comparison counter: 822334

Test6

New Sort comparison counter: 238691

Observations

If we consider the results from test 1, 2 and 5 for insertion sort we see that the best case is significantly faster than the average case. So, insertion sort would be better used for almost nearly sorted arrays (and sorted of course).

We can also see that Insertion sort does not perform well as Quicksort for example when it comes to larger arrays. So, the smaller the array the better Insertion sort would work.

New Sort is faster when there are duplicates in the array, because it swaps them and orders them when it finds their place. New Sort would perform better in array that has a lot of duplicates.

New Sort has the same performance for sorted and unsorted arrays if they have the same values (test2 and test3).

Quicksort performs the worst when the pivot is chosen to be the largest or the smallest number of the array. In tests 5 and 6 we can see the difference between taking the largest number that will be last in the sorted array (in test 5) and taking one of the smallest numbers that is not at the beginning of the sorted array. (test 6)

All in all, Insertion sort is best for sorted and small arrays, Quick sort is only quick when the pivot value is located near the middle of the sorted array and New Sort works best when there are duplicates in the array.