COMP 301 Analysis of Algorithms Lab 2

Q1.

a) Insertion sort is tested for an array of size 10.

Original array:

[24, 24, 10, 16, 31, 83, 50, 95, 63, 44]

After insertion sort:

[10, 16, 24, 24, 31, 44, 50, 63, 83, 95]

b) Reverse insertion is tested for array of size 10.

Original array:

[24, 24, 10, 16, 31, 83, 50, 95, 63, 44]

After reverse insertion sort:

[95, 83, 63, 50, 44, 31, 24, 24, 16, 10]

c) Time for best-case of insertion sort: 57690 nanoseconds

d) Time for worst-case of insertion sort: 55868712 nanoseconds

e) Time for best-case of insertion sort: 204920 nanoseconds

Time for worst-case of insertion sort 5224466697 nanoseconds

The values for both best-case and worst-case increased. However, sorting an already sorted array is still faster than sorting a reversely sorted array.

Q2.

b) Merge sort is tested for array of size 16.

Original array:

[57, 91, 29, 16, 87, 78, 24, 46, 37, 9, 16, 84, 4, 94, 61, 26]

Array after merge sort:

[4, 9, 16, 16, 24, 26, 29, 37, 46, 57, 61, 78, 84, 87, 91, 94]

Q3.

1. Running time for merge sort and insertion sort is calculated as given below:

Time for merge sort: 301

Time for insertion sort: 129313691

For this input size, merge sort works faster than insertion sort.

b) **for**(**int** i = 1; i <= 100; i++){

**if**(8\*Math.*pow*(i, 2)< 64\*i\*Math.*log*(i)/Math.*log*(2))

System.***out***.println(i);

}

Insertion is faster than merge sort for n<43 in (0,100)