

**Addis Ababa University**

**Addis Ababa Institute of Technology**

**Faculty of Electrical And Computer Engineering**

**Algorithm Analysis And Design course**

**Assignment One**

**Daniel Fekadu**

**ATR/5951/05**

**Date Dec-28 2016**

**Introduction**

In this assignment we are going to be testing the performance of different sorting algorithms. To that effect I have written a c++ implementation of these algorithms. This algorithms were tested by changing different parameters discussed in the following sections.

**Experiment Setup**

In this experiment we are going to measure the running time of each algorithm by giving it different sized arrays with different levels of presortedness. And the running time of the algorithms is measured By the number of system clock tics between the start and termination of the algorithm.

This experiment was done on a machine with the following spec

- cori3 1.8 Ghz processors

- 3.8 Gb ram

Due to the limitations of my machine I was able to run the experiment with a maximum array size of 100000. But this was enough to confirm that our theoretical assumptions are correct.

Our experiment can be expressed by the following psudo code.

for every presortedness parameter:

do for every repetition :

do for every size(n):

do generate\_random\_array()

do test\_all\_the\_algorithms\_on\_the\_same\_array()

report result

finally my code generates a CSV file for every presortedness value{0,0.25,0.5,0.75,1} containing the average running time of each algorithm.

**Results**

Start = 0 ,Stop=100000,Step = 20 Presortedness = 0 repetition = 2

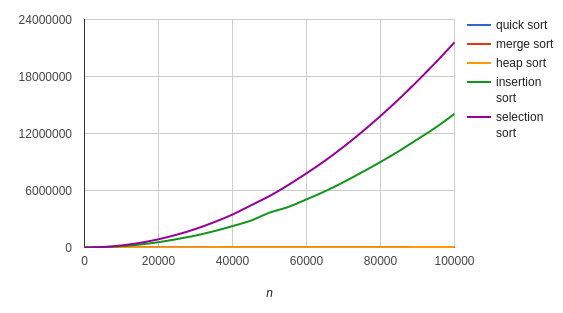


fig .1(Y axis = number of clock tics)

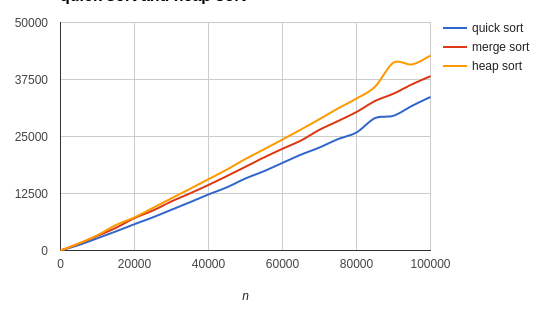


fig .2(Y axis = number of clock tics)

As you can see in fig 1 the lines of merge ,heap and quick sort overlap with one another. This happens because heap ,quick and merge sort are a lot faster than insertion and selection sort. So when we try to plot it on a graph scaled to accommodate insertion and selection sort the difference among heap quick and merge sort will be insignificant,hence not visible. So I have plotted them separately(fig 2).

Start = 0 ,Stop=100000,Step = 20 Presortedness = 0.25 repetition = 2

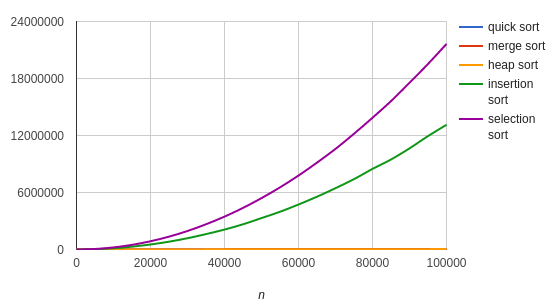


fig 3(Y axis = number of clock tics)

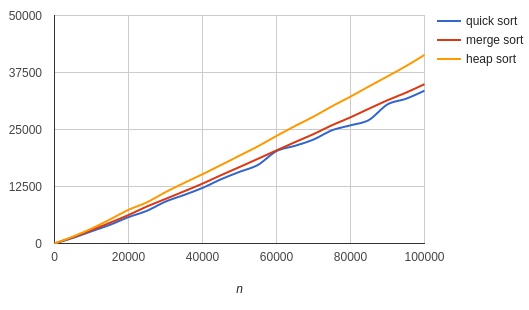


fig 4(Y axis = number of clock tics)

As you can see in fig 3 the lines of merge ,heap and quick sort overlap with one another. This happens because heap ,quick and merge sort are a lot faster than insertion and selection sort. So when we try to plot it on a graph scaled to accommodate insertion and selection sort the difference among heap quick and merge sort will be insignificant,hence not visible. So I have plotted them separately(fig 4).

Start = 0 ,Stop=100000,Step = 20 Presortedness = 0.5 repetition = 2

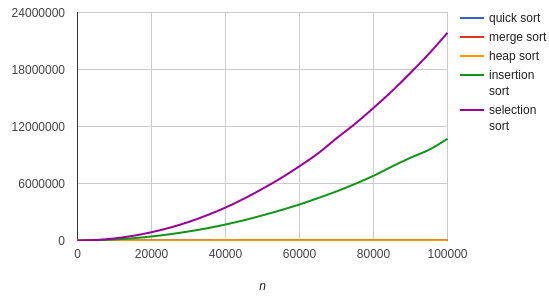


fig 5(Y axis = number of clock tics)

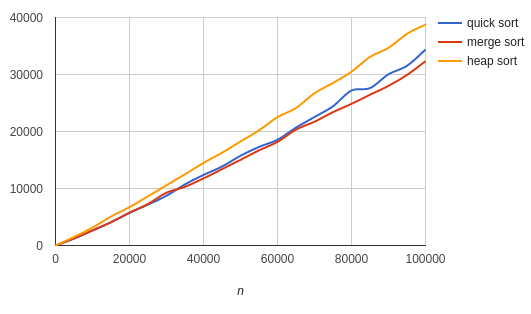
****

fig 6(Y axis = number of clock tics)

As you can see in fig 5 the lines of merge ,heap and quick sort overlap with one another. This happens because heap ,quick and merge sort are a lot faster than insertion and selection sort. So when we try to plot it on a graph scaled to accommodate insertion and selection sort the difference among heap quick and merge sort will be insignificant,hence not visible. So I have plotted them separately(fig 6).

Start = 0 ,Stop=100000,Step = 20 Presortedness = 0.75 repetition = 2

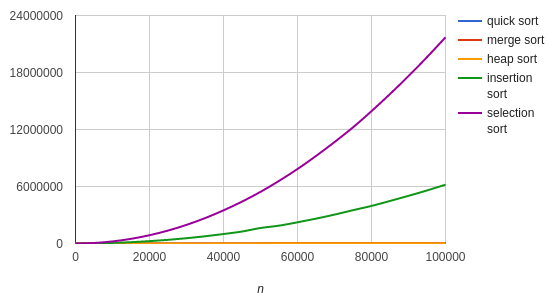


fig 7(Y axis = number of clock tics)

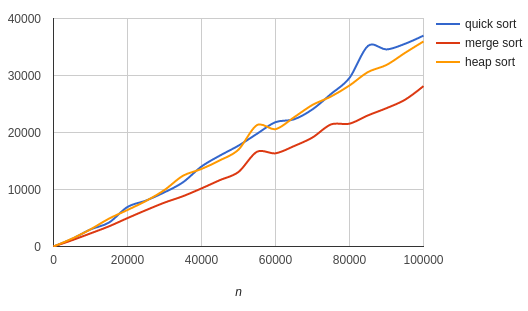


fig 8(Y axis = number of clock tics)

As you can see in fig 7 the lines of merge ,heap and quick sort overlap with one another. This happens because heap ,quick and merge sort are a lot faster than insertion and selection sort. So when we try to plot it on a graph scaled to accommodate insertion and selection sort the difference among heap quick and merge sort will be insignificant,hence not visible. So I have plotted them separately(fig 8).

Start = 0 ,Stop=100000,Step = 20 Presortedness = 1 repetition = 2

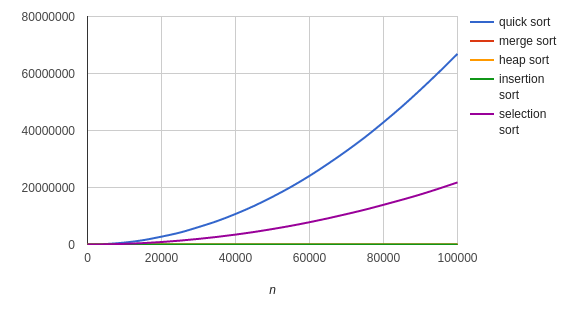


fig 9(Y axis = number of clock tics)

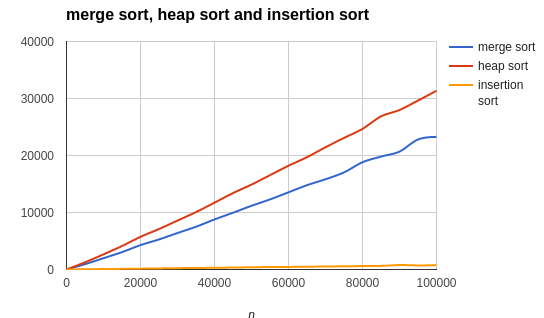


fig 10(Y axis = number of clock tics)

As you can see in fig 9 the lines of merge ,heap and insertion sort overlap with one another. This happens because heap ,quick and merge sort are a lot faster than insertion and selection sort. So when we try to plot it on a graph scaled to accommodate quick and selection sort the difference among heap quick and merge sort will be insignificant,hence not visible. So I have plotted them separately(fig 10).

**Conclusion**

All in all the running time all the algorithms we tested were the same as our theoretical predictions.

We saw how the different algorithms behave under different scenarios. To mention some

1.Insertion sort has the best running time when the input array is sorted.

2.quick sort has the best running time except when the input array is sorted(or mostly sorted) as expected from our theoretical analysis.

3.selection sort has the worst running time under almost every circumstance except for when the input array is sorted in which case its running time will be better than quick sort.

4.the performance of merge sort does not vary with the level of presortedness of the input array .the same for heap sort.

By running the algorithms in a more sophisticated machine , by increasing the number of repetitions and by increasing the array sizes we can improve the result of this experiment.

The complete code for this assignment including instructions on how to run the code can be found on this link.

The CSV files containing the results can be found in main\_directory/report/log/ .