CS 201 EXPERIMENTAL REPORT FOR HW-5

By: Esrah Zahid (S020289)

Absract

This experiment explores an experimental based selection of the best sorting algorithm to find the kth largest number in a set of N numbers using one of the 4 selection algorithm types.
The 4 algorithm types were implemented in our previous homework assignments.
We will be taking a look at how effiecient each algorithm is and comparing it with our theoretical time complexity.

Name	Best	Average	worst
InsertionSort	N	N^2	N^2
SelectionSort	N^2	N^2	N^2
HeapSort	NLogN	NlogN	NLogN
QuickSort	NLogN	N	N^2

I. Experimental Setup

>>The data size (N) in my experiment will be the following:
10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000.

>>The contents of the text file will remain the same for each of the algorithms.

>>All input will be generated using 'Test Input Generator-v5'.

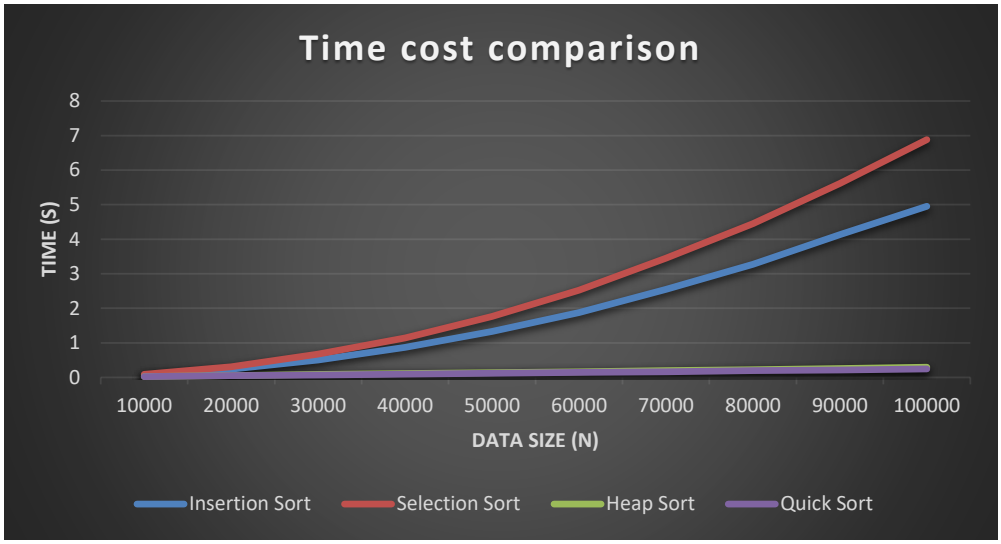
>>The algorithms used will be:
1) Insertion Sort
3)Selection Sort
2)Quick Sort (hybrid)
3)Heap Sort

>>I will be running each algorithms only once and taking a note of each algorithm's running time in secc
I will use this time to fill in the table shown below.

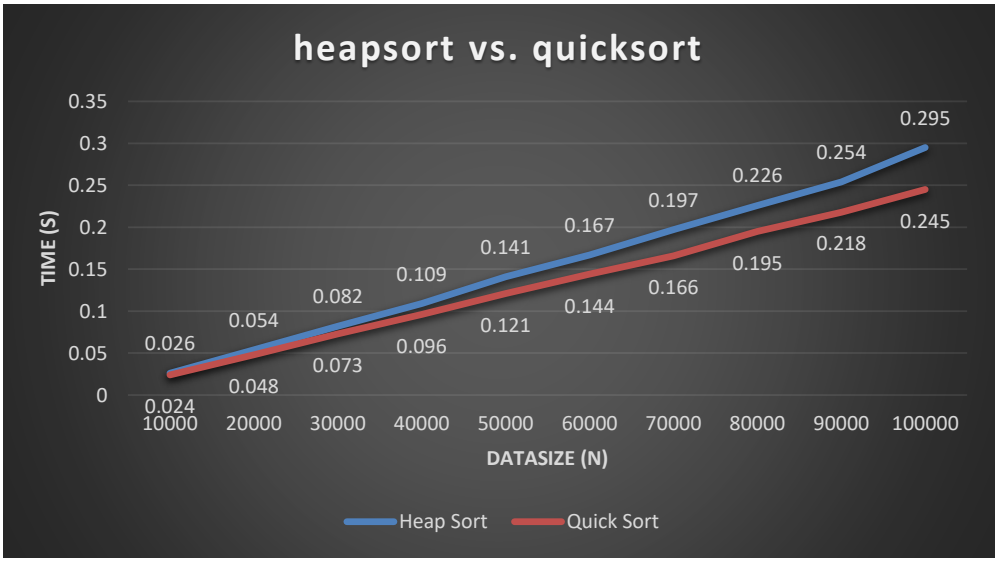
II. Result

My findings using the experimental setup are listed in the table below:

Data Size (N)	Insertion Sort	Selection Sort	Heap Sort	Quick Sort
10000	0.072	0.089	0.026	0.024
20000	0.243	0.306	0.054	0.048
30000	0.505	0.671	0.082	0.073
40000	0.871	1.146	0.109	0.096
50000	1.332	1.764	0.141	0.121
60000	1.877	2.521	0.167	0.144
70000	2.545	3.454	0.197	0.166
80000	3.277	4.448	0.226	0.195
90000	4.138	5.617	0.254	0.218
100000	4.953	6.882	0.295	0.245



Let us take a closer look at heapsort and quicksort:



iii. Discussion

From the table and the figure above, we can conclude that when the scale of input N was small, the difference of time cost between these algorithms was small. But with the scale of input N becoming larger and larger, the difference became larger and larger. Among these algorithms, the QuickSort and HeapSort were the fastest compared to the other 2 traditional sorts.

With the input scale increasing, the QuickSort has certainly on advantage over other algorithms with a running time of 0(N) it is one of the fastest running algorithms.

My results match the theoretical running time and with that we can conclude that our algorithm selection can be based on our input size because when the input size is small, the difference in time isnt very noticeble but when it comes to larger inputs, the reason quickselect is faster is because we do not need to sort (by doing partition on) all the elements, but only do operation on the partition we need. Therefore we can safely disgard the array partition we dont need which is essential for a quicker output with a running time of O(N)

Another point to note is reason QuickSort outperforms heapsort by a very small difference is the amount of swapping. Since heapsort requires a lot of swapping a lot more time is consumed. But in cases where quicksort is likely To be performing poorly mainly when data sets are sorted, heapsort is preferred.

ɔnds.