Zhang Zhuo

Homework 9

1. Table:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10000 | 20000 | 30000 | 40000 | 50000 | 60000 | 70000 | 80000 | 90000 | 100000 |
| insertion | 67 | 110 | 154 | 156 | 395 | 578 | 836 | 1105 | 1378 | 1661 |
| merge | 7 | 2 | 5 | 5 | 6 | 7 | 8 | 9 | 7 | 9 |
| quick | 5 | 3 | 4 | 5 | 6 | 8 | 10 | 12 | 13 | 15 |
| heapsort | 9 | 3 | 4 | 5 | 7 | 7 | 10 | 9 | 12 | 13 |

1. Graph:
2. Conclusion:

Insertion sort has O(n^2) time complexity and it shows in the results. As the input increases in size the run time is increased significantly. In the graph it’s also the only one that takes way more time than the rest.

Merge sort has O(nlog(n)) time and the actual results show that the increase is significantly slower than insertion sort. In the actual results it actually performed better than O(n) but of course since the experiment is not very thorough it can not represent the average performance.

Quick sort also has O(nlog(n)) time and its actually performance is consistently marginally worse than merge sort similar to heap sort. Although only occasionally did I run into “stack overflow error” while running quick sort its recursion does consume a lot of memory space.

Heap sort is yet another O(nlog(n)) algorithm and all three have very similar performances.

So the actual performance of the four above algorithms do match their time complexities very well and algorithms with same time complexities do perform similarly. The only difference might be overhead from how they are implemented internally.

1. Source:

Insertion sort and merge sort source code are from the text book, and both source code of quick sort and heap sort are from geeksforgeeks website. Almost all algorithms are slightly modified to fit the need of this homework.