**INFS-519 homework-1 Analysis report Deepak Kanuri G01070295**

**Quick Sort**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N (size of file) | Run Time (T(n)) in nano-seconds | T(n)/N | T(n)/N\*\*2 | T(n)/n log(n) |
| 100 | 44476 | 444.76 | 4.4476 | 66.943050 |
| 1000-S | 493513 | 493.513 | 0.493513 | 44.82175871 |
| 1000 | 257448 | 257.448 | 0.257448 | 25.833190107 |
| 10,000 | 2911471 | 291.1471 | 0.02911471 | 21.9110025626 |
| 100,000 | 11276390 | 112.76390 | 0.0011276390 | 6.78906326561 |

**Insertion Sort**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N (size of file) | Run Time (T(n)) in  nano-seconds | # inversions | T(n)/N | T(n)/N\*\*2 | T(n)/n log(n) |
| 100 | 313043 | 2662 | 3130.43 | 31.3043 | 471.17666466 |
| 1000-S | 251033 | 621 | 251.033 | 0.251033 | 25.18946756 |
| 1000 | 7416382 | 249834 | 7416.382 | 7.416382 | 744.18448043 |
| 10,000 | 49463786 | 25046158 | 4946.3786 | 0.49463786 | 372.2520821 |
| 100,000 | 1201166898 | 2501541356 | 12011.66898 | 0.1201166898 | 723.174531 |

**Bubble Sort**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N (size of file) | Run Time (T(n)) in  nano-seconds | # inversions | T(n)/N | T(n)/N\*\*2 | T(n)/n log(n) |
| 100 | 378901 | 2637 | 3789.01 | 37.8901 | 570.3028319 |
| 1000-S | 7978320 | 568 | 7978.320 | 7.978320 | 800.57121166 |
| 1000 | 4442473 | 222531 | 4442.473 | 4.442473 | 445.77254264 |
| 10,000 | 214507576 | 22456612 | 21450.7576 | 2.14507576 | 1614.33036682 |
| 100,000 | 17259860691 | 2247504714 | 172598.60691 | 1.7259860691 | 10391.1103419 |

As we can observe from the tables quick sort algorithm is fastest in sorting any Input file. The algorithm ranked second is the insertion sort and finally comes the bubble sort.

The peculiar thing to be noticed is the duration difference between the 1000-integers file and 1000-almost-sorted-integers file. Here bubble sort and quick sort takes more time for the almost sorted file rather than the normal file. This indicates that the more sorted the file is the complex it is for these algorithms to sort them, as quick sort has to divide the file numerous times to sort, while bubble sort has to continuously parse through the file to find the oddly placed item in the array. Contrastingly insertion sort takes less time for almost sorted file. This is because insertion sort parses the array, finds the oddly placed item and then swaps it to its right place. Interestingly, if observe the inversions of insertion and bubble sorts for 1000 and 1000-almost sorted files, the inversions decreased for almost sorted file. Obviously this is because almost all the items were rightly placed, and just the ones oddly placed were to be swapped.

The unique thing I can proudly talk about my code is that the bubble sort takes less number of inversions than the insertion sort for any file. This is because I used an efficient bubble sort algorithm to sort and the output is still the same sorted order. My bubble sort algorithm reduces the size of the array to be sorted, for every outer loop iteration.

Now if we concentrate on the last 3 columns of all the tables, which are the T(n)/N, T(n)/N\*\*2 and T(n)/N log(N). For each of the 3 sorting algorithms none of the values converge for the column T(n)/N. So the none of the algorithms have a time complexity of O(N). The next column is T(n)/N\*\*2. Here the values for quick sort don’t converge, but for insertion and bubble sorts the values relatively do converge. There not much difference between the values while going down the column when compared to the other columns. So we can conclude that the insertion and bubble sorts have a time complexity of O(N). This is because the algorithms parse the array N times for every one of the N iterations.

Coming to quick sort, we can observe that the values relatively converge for the column T(n)/N log(N). This is because the quick sort uses “Divide and conquer” strategy. It divides the total array into two parts and selects a pivot, where values less than the pivot are stored in the left of the array and higher values on the right part of the pivot. Now these two parts are again fed to the quick sort algorithm and the process is done again for each part. This is done till all the values in each part are sorted. Thus for partitioning it takes log(n) time for each of the N items. Thus the time complexity of quick sort is O(N log N).

In general when we compare the three algorithms we can see that bubble sort and insertion sort is better to use if the items to be sorted is relatively less in number. We may increase the size of items to be sorted for insertion sort up to some extent, but after a certain stage bubble sort becomes troublesome and time consuming. This can be seen for the execution of bubble sort on 100,000 integers file. On the other hand insertion sort is proven to be useful if we have to sort an array which has very few items which not in-place.

Quick sort becomes more efficient if the size of the array to be sorted is large. Thus we can see a relation that the efficiency of the quick sort algorithm is directly proportional to the size of the size of array. Of-course the time increases with the size of array but its speed increases also when compared to other algorithms.

So finally to conclude: time complexity of Quick sort=O(N log N), time complexity of Insertion sort=O(N\*\*2) and the time complexity of bubble sort=O(N\*\*2).