**Analysis for Randomized Quick Sort Algorithm**

**By Prachi Gupta, G01025257, User: pgupta13**

Randomized Quicksort algorithm is basically used to get the average case performance by randomly picking the pivot. In the solution this is achieved in two ways,

1. Calling random function on the indices of the array to pick pivot randomly
2. Using median of three method and selecting the pivot as the middle value of three randomly chosen indices whose elements are compared

Answer 1. Running time statistics are given below:

For average case performance:

Counter values set in the code for operations in Insertion Sort with input as 100, 1000 and 10000 is given below:

|  |  |
| --- | --- |
| Value of n | Average Counter value for 30 runs |
| 100 | 3250 |
| 1000 | 44963 |
| 10000 | 589825 |

For worst case performance:

Counter values set in the code for operations in Insertion Sort with input as 100 and 1000 is given below:

|  |  |
| --- | --- |
| Value of n | Average Counter value for 30 runs |
| 100 | 6930 |
| 1000 | 519480 |

The running time of the Randomized Quicksort is fairly good(average case) when random input array is given. To get average case performance, we select randomly chosen elements as pivot for partition. The overall scenario is good when randomization is achieved. On the other hand, when array with same elements is passed to the algorithm, gives worst case performance. For example, for n=100 the counter notes a value of 6930 which is double the value when random elements array is passed. This performance degrades even more when run for n=1000(In the solution, a for loop assigns value 3 as element of array for all indices). The above table captures the results. For very large input n of same elements array, the algorithms throws ‘StackOverflow’ exception.

The asymptotic analysis of Insertion sort and Merge sort is given below. Note that they are theoretically proven formulas and have been seen in practical cases all practical cases.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Best Case | Average case | Worst Case |
| Quick Sort | Ω(n log(n)) | Θ(n log(n)) | O(n^2) |

In the table we can see time complexity analysis for the sort for various cases. The table figures out that the randomized quick sort has an average performance in all cases except for worst case scenario where all elements are same. The tables show that the sort is fairly good if our test input is random and have elements distributed all over the array range. The performance issues arises when the elements are similar and notes a drastically large values of counter in the solution program.

Answer 2. For the bonus question, I have implemented with the main problem’s solution and asked for user’s input on how should the user choose pivot, like the case asked in question one or in question two(median of three method).

|  |  |
| --- | --- |
| Value of n | Average Counter value for 30 runs |
| 100 | 4449 |
| 1000 | 58582 |
| 10000 | 724097 |

In the solution, the comparison for calculating the median of the three randomly chosen elements captures a few counter values, the reason why there is a difference between the 1 question’s method and median of three method.

The algorithms works fairly good but not as good as the first one(due to extra calculations for median values). The average case has a little dissimilarity with the question one’s implementation. The asymptotic analysis for this solution remains same as shown previously. The only change is how we choose our pivot to achieve more randomness in order to achieve average case performance every time. The algorithms runs fine for large inputs as well.