**Quicksort Algorithm Implementation, Analysis, and Randomization**

**Pranoj Thapa**

**University of Cumberlands**

**MSCS-532-B01**

**Dr. Vanessa Cooper**

**November 17th 2024**

**Performance Analysis**

Time Complexity:

Best Case:

At each step, the array is split into two nearly equal sections by the pivot.   
The array of size n is divided in O(n) time at each recursive level.   
Due to the array size halving at each level, there are O(logn) levels of recursion.

Average Case:

The array is typically divided into two subarrays of approximately equal size by the pivot.

It follows the same logic as the best case: O(n) work per level O(logn) levels.

Worst Case:

Occurs when the array's pivot is either its largest or smallest element.

There are 0 elements in one subarray and n -1 elements in the other.

Recursion occurs n times, while partitioning takes O(n).

Space Complexity:

In the best and average cases, Quicksort can be executed in-place, requiring O(logn) space to store the recursive stack.  
The depth of the recursive stack causes the space complexity to be O(n) in the worst scenario.

Reason for average case and worst case:

The best-case behavior recurs for the average-case time complexity of Quicksort since, on average, the pivot creates two nearly balanced partitions, each of size n/2. This splitting is almost half and half to ensure that at each step of the recursion, the size of the problem reduces drastically. Given that the partitioning of the array takes O(n) and the recursion depth is O(logn) the two combine to give an algorithm whose time complexity is O(nlogn). Such balance is typical of randomized or varying input data sets to minimize the impact of undesirable input orders.

The use of the Quicksort time complexity matrix shows that the worst-case time complexity of O(n^2) is reached if the element pivot separates the array into two sub-arrays of high imbalance such as choosing the smallest or the largest element as the pivot in a sorted array. In this case, it means that each step of recursion deals with a smaller subarray and it takes n, n-1, n-2, … 1 comparison in all. Such linear depth of recursion with full traversal at each level is a quadratic- run time. What is left is using random pivot selection or other methods, such as the median-of-three algorithm, to avoid this worst-case behaviour.

**Randomization and Its Impact on Quicksort Performance**

Randomization in Quicksort improves the efficiency of the algorithm because it practically eliminates the chance of the worst-case scenario. In the deterministic case such as always choosing the left most or right most element as a pivot, the partitions obtained can be worst for some data sets, for example, sorted or reverse-sorted arrays. Randomized Quicksort on the other hand chooses a pivot arbitrarily so as to reduce the probability of picking only the smallest or the largest element at every level of recursion exponentially. Thus, the algorithm continues to achieve an average case of a time complexity of O(n log n) across the spectrum of ‘small’ input orders, including adversarial orders.

The improvement comes from the fact that random pivot selection means that the subarrays are divided much more evenly on the average. In addition to this, the opportunity is evenly distributed in each partitioning step hence minimizing the effects of skewed input data. This last aspect of Randomized Quicksort speaks loudly in actuality where the exact nature of the input order is unknown or can be anything but random.

**Empirical Analysis**

The performance comparison on small input sizes shows that the deterministic Quicksort performs as fast as both versions of randomized Quicksort and takes less than one millisecond for all tested distributions. Randomized Quicksort scales nearly O(nlogn) as input size increases proving a significant performance gain and a consistent situation on sorted, reverse-sorted, and random input arrays. However, the deterministic version appears to be slightly more variable especially when dealing with sorted input. The variation in time taken is probably a result of possibly inferior partitioning. This observation conforms with theory as confirmed by the fact that randomness in the selection of the pivot in the randomized version dramatically reduces the real possibility of worst-case O(n^2) performances given that partitioning is more evenly spread in terms of the input distribution. Finally, the randomized version is safer and offers more similar performance in all kinds of situations.