**Assignment 3: Understanding Algorithm Efficiency and Scalability**

Syed Noor UI Hassan

University of the Cumberlands

Algorithms and Data Structures (MSCS-532-B01)

November 10, 2024

## Randomized Quicksort

Analysis:-

It can be said that by choosing a pivot at random, the Randomised Quicksort algorithm helps to prevent worst-case situations (such as sorted or reverse-sorted arrays). In the average case, the recurrence relation for randomised quicksort is comparable to that of deterministic quicksort.

For array of n size, it can be said as:

$$T(n) = \frac{1}{n}\sum_{k=0}^{n-1}\big(T(k) + T(n-1-k)\big) + O(n)$$

Using the above we can say that the expected depth of the recursion os O(logn), while each partitioning step takes linear time, O(n). In this way, the average case time complexity is O(n log n).

Comparison

To support the empirical comparison of running time between Randomized Quicksort and Deterministic Quicksort, the two algorithms were applied to a randomly generated array of size 1000. In this experiment, Randomized Quicksort uniformly and randomly chose the pivot from the subarray always, whereas Deterministic Quicksort always chose the first element as the pivot. The outcome from the test conducted indicated that the time taken to complete the operation in Randomized Quicksort was close to 0.004 seconds while on the other hand, Deterministic Quicksort took negligible time. From these initial results, one can conclude that for this particular size of the input, both algorithms seem to be equally efficient with the Deterministic Quicksort being capable of providing results slightly faster.

As for the input distributions, it is assumed that already sorted, reverse sorted, or containing duplications elements will have different asymptotic performance as analyzed theoretically. Randomized quick sort which uses random selection of pivot is better off avoiding worst case time complexity of O(n^2) seen in Deterministic quick sort. For sorted and reverse-sorted arrays, Deterministic Quicksort continuously chooses poor pivots, which results to skewed partitions and a worst-case scenario. Randomized Quicksort, on the other hand, is not prone to such a problem because of random pivot selection, but its time complexity is also the best case of O(nlogn).

When the array contains a duplicate, Randomized Quicksort will prove more efficient as the random selection of pivot makes partitioning more balanced than Deterministic Quicksort that may select a pivot that generates poor partitioning. The minor time difference that was found in the observed results could have been caused by the overhead of the random number generator in Randomized Quicksort but this effect is not very significant when the input sizes increase.

The results obtained are consistent with theoretical predictions, whereby Randomized Quicksort does not get worst-case situations that Deterministic Quicksort could encounter attributable to pivot selection. However, actual differences can be observed only on small input sizes since both algorithms complete the process relatively fast on small arrays and, thus, differences can be rather attributed to a lack of theoretical benefits provided by randomized pivoting.

## Hash with Chaining

Analysis:-

Expected Time Complexity

The expected duration and complexity for these operations in the case of simple uniform hashing is O(1). Nevertheless, this is dependent on the load factor α, which is the table's ratio of elements (n) to slots (m). In the worst case it can be said that the operations degrade to O(n). This is because the entire list in the chain needs to be traversed.

Load factor

The load factor is nothing but $\alpha = \frac{n}{m}$

The hash table's "fullness" can be determined by the load factor. Fewer elements per chain result in a low load factor, which speeds up insert, delete, and search operations. As the $\alpha$ increases, the average chain length increases. This leads to slower operations.

Strategies to Maintain Low Load Factor

The strategies are mentioned below:-

Dynamic Resizing- When the load factor goes beyond a certain level the table can be reshaped to enhance the efficiency of the table. This involves creating a new table with larger number of slots and then redistributing all the existing elements.

Universal Hashing- Using a hash function selected from a universal hash family guarantee that the number of collisions is kept to a minimum since the keys are spread out evenly throughout the table.

Prime Number Table Size- Hash table size should be chosen as the prime number this is due to the fact that clustering of hash values will be minimized thereby reducing further collisions.

Such strategies assist in keeping the performance high because load factor is kept low and collisions are rare.