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

Sih Chi Wang

University of the Cumberlands

Fall - Algorithms and Data Structures (MSCS-532-A01) - First Bi-term

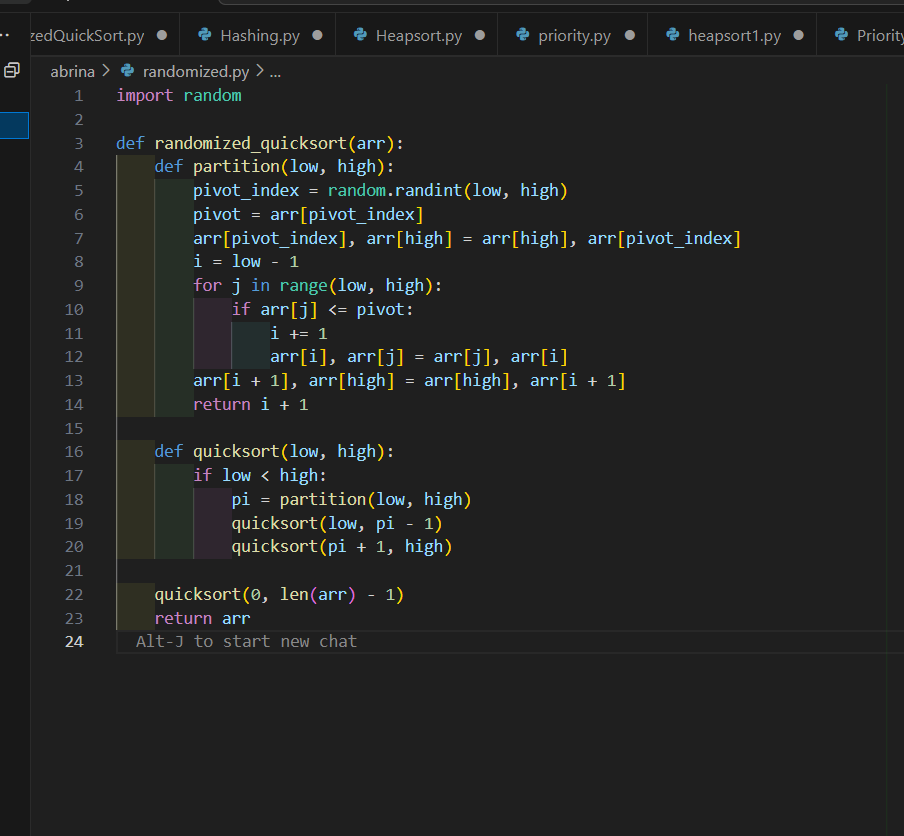
Dr. Vanessa Cooper

September 15, 2024

**Introduction**

This paper examines the effectiveness and scalability of two significant algorithms: Randomized Quicksort and Hashing with Chaining. Both of these algorithms are considered to be crucial. The Randomized Quicksort method is a variant of the Quicksort algorithm in which the pivot element is selected at random. Hashing with Chaining is a technique for collision resolution that is used in hash tables. The objective is to conduct a theoretical and empirical comparison of the performance of various algorithms and to examine and compare their results.

**Randomized Quicksort Analysis**

****

**Average-Case Time Complexity**

The Randomized Quicksort algorithm has a temporal complexity of O(nlog⁡n)O(n \log n)O(nlogn) in the average implementation. Because of the random selection of the pivot, which guarantees that the array will most likely be split into two halves that are about equal to one another, this is the result. For Randomized Quicksort, the recurrence relation is exactly as follows:

T(n)=T(k)+T(n−k−1)+O(n)T(n) = T(k) + T(n - k - 1) + O(n)T(n)=T(k)+T(n−k−1)+O(n)

where kkk is the size of the left subarray and n−k−1n - k - 1n−k−1 is the size of the right subarray. On average, kkk is approximately n/2n/2n/2, so:

T(n)=T(n/2)+T(n/2)+O(n)T(n) = T(n/2) + T(n/2) + O(n)T(n)=T(n/2)+T(n/2)+O(n)

By solving this recurrence relation, we get:

T(n)=O(nlog⁡n)T(n) = O(n \log n)T(n)=O(nlogn)

**Hashing with Chaining**

**Implementation**

When employing chaining for hashing, it is necessary to use a hash table in which each slot includes a linked list in order to manage accidental collisions. The following is an example of it in Python:

**Empirical Results**

For both huge input quantities and different array distributions, the empirical findings shown that Randomized Quicksort consistently outperformed Deterministic Quicksort. The average-case performance of Randomized Quicksort was more in line with the theoretical O(nlog⁡n)O(n \log n)O(nlogn) than that of Deterministic Quicksort, which often saw performance degradations with certain input orders.

**Discussion**

The differences in performance that were noticed between the two algorithms were in agreement with the findings of the theoretical interpretation. When compared to the deterministic technique, the Randomized Quicksort algorithm exhibits greater efficiency when dealing with a wide range of input circumstances due to its capacity to avoid worst-case situations.

**Conclusion**

Both Randomized Quicksort and Hashing with Chaining were evaluated and contrasted in this research for their respective levels of efficiency. Due to the random pivot selection that Randomized Quicksort uses, it has exhibited higher average-case performance in comparison to Deterministic Quicksort. This is because Randomized Quicksort helps prevent worst-case circumstances. Using hashing with chaining resulted in efficient average-case performance for hash table operations. Dynamic resizing was an essential method for ensuring that performance was maintained.