Understanding Algorithm Efficiency and Scalability

Randomized quicksort:

- The pivot element is selected from the random array, which needs to be partitioned.
- Average Time Complexity 'O (n log n),' the rando pivot element tries to create equal and balanced partitions on average.
- Worst-case time complexity O(n^2). Possibility of partitioning to unbalanced partitions.

Deterministic Quicksort:

- Average-case time Complexity is the same as randomized complexity O (n log n).
- Worst-case will also be O(n^2) because picking the first element could create unequal partitions.

Empirical Observations:

- **Randomized Quicksort:** It performs mainly consistently, although input sizes differ. Execution time grows logarithmically on increasing input sizes that justify the definition O(nlogn). Performance is more consistent than deterministic quicksort because the random selection of pivot elements partitions the array in balance on average.
- **Deterministic Quicksort:** It is like randomized performance when the input size is small; however, a larger input size takes more execution time due to its pivot element selection. Since it chooses the first element as a pivot, it creates unbalanced subarrays. That concludes that randomized quicksort has better consistency performance than deterministic quicksort.

Hash Table with Chaining:

A hash function is a series and distribution of keys and value pairs. Each key will store value and be known as slots for such a pair. It has the advantage of storing data in the same slots, creating chains while preventing data loss. Time complexity is constant as it is key-driven. That means the load factor will be low on any size.

Mathematically load factor (α) = number of elements(n)/ number of slots(m)

- **Insert:** Each key is equally hashed in all slots. The key is appended to the chain at the index. Because of that, in any size or input, the time to hashed will be the same as O(1).
- **Search:** Since the key is hashed with O(1), the expected time complexity will be $O(1 + \alpha)$, where α is a load factor.
- **Delete:** It will be the same as the search because it will be key-driven. That means it should take equal time to delete as it takes to search. Therefore expected complexity of $O(1 + \alpha)$. Mostly constant with a low load factor.

References:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3rd ed.). MIT Press.

Knuth, D. E. (1998). The art of computer programming, volume 3: Sorting and searching (2nd ed.). Addison-Wesley.

Weiss, M. A. (2012). Data structures and algorithm analysis in C++ (4th ed.). Pearson