# Programming Assignment 3

### Priority Queues

By

Kishlay Kumar and K Pramod Krishna

E-A 008: Data Structures and Algorithms

Prof. Aakash Tyagi

TA : Raghav Awasty

# 

# **Introduction**

# 

We will implement three different versions of a hash table and study their performance. Namely:

(1) Chaining

(2) Linear Probing

(3) Double Hashing

# 

# **Theoretical Analysis**

**Hash tables:** They are a type of data structure in which the address or the index value of the data element is generated from a hash function. That makes accessing the data faster as the index value behaves as a key for the data value. In other words, Hash table stores key-value pairs but the key is generated through a hashing function.

**Chaining in Hash tables**

Chaining is a technique that avoids collisions in [hash tables](https://www.educative.io/edpresso/what-is-a-hash-table). A collision occurs when two keys are [hashed](https://www.educative.io/edpresso/what-is-hashing) to the same index in a hash table. Collisions are a problem because every slot in a hash table is supposed to store a single element.

**Insertion:**

Amortized time complexity = O(1)

Total time complexity for N elements = O(N)

**Removal:**

Amortized time complexity = O(N)

Total time complexity for N elements = O(N2 )

**Linear probing in Hash tables**

It is a scheme for resolving [collisions](https://en.wikipedia.org/wiki/Hash_collision) in [hash tables](https://en.wikipedia.org/wiki/Hash_table), [data structures](https://en.wikipedia.org/wiki/Data_structure) for maintaining a collection of key–value [pairs](https://en.wikipedia.org/wiki/Attribute%E2%80%93value_pair) and looking up the value associated with a given key. A hash table maintains one key-value pair in each cell. When the hash function creates a collision by mapping a new key to a hash table cell that is currently filled by another key, linear probing looks for the next available space in the table and inserts the new key there.

**Insertion**

Amortized time complexity = O(N)

Total time complexity for N elements = O(N2)

**Removal**

Amortized time complexity = O(N)

Total time complexity for N elements = O(N2)

**Hash Table - Double Hashing**

Open Addressed Hash tables use the collision-resolution method of double hashing. When a collision happens, the idea behind double hashing is to apply a second hash function to the key. The benefit of double hashing is that it produces a uniform distribution of records across a hash table, making it one of the greatest methods of probing. Clusters are not produced by this method. It is one of the most efficient ways to deal with collisions.

**Insertion**

Amortized time complexity = O(N)

Total time complexity for N elements = O(N2)

**Removal**

Amortized time complexity = O(N)

Total time complexity for N elements = O(N2)

**Experimental Setup**

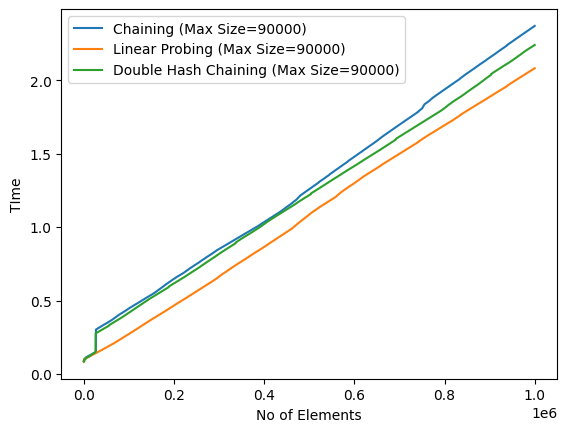
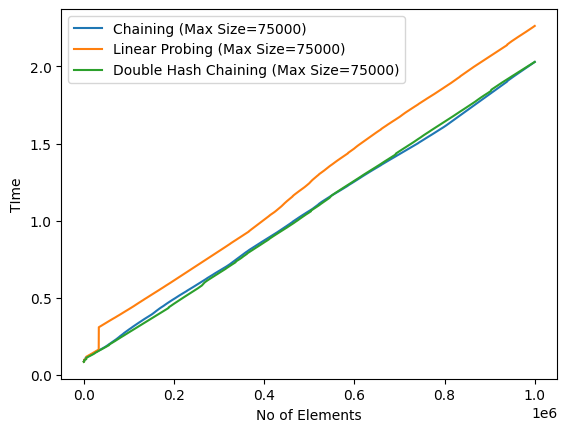
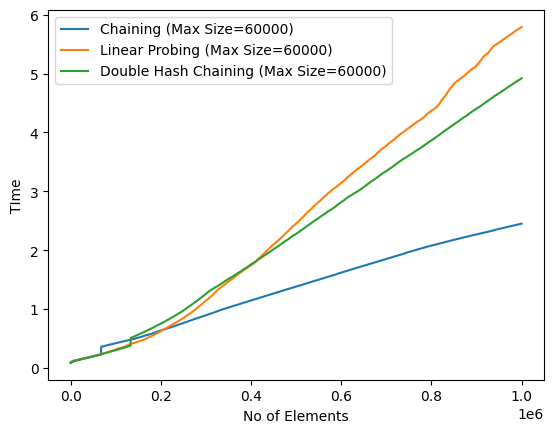
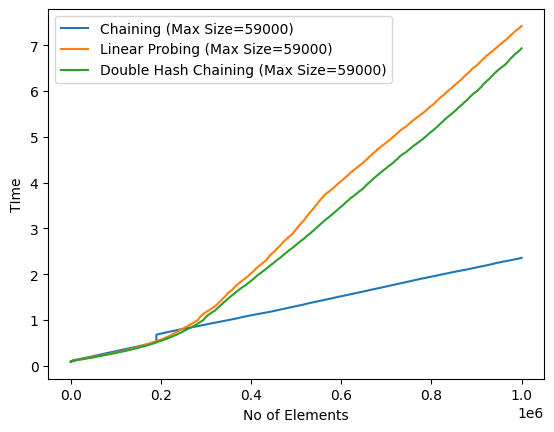
We used a machine running windows 11 OS with the following specs:



**Test Inputs & Process:**

We tested the insertion times for various size of hash tables like 59000, 60000, 75000 and 90000 using the provided dictionary.txt as the input. Recorded the time for every 10 insertion operations, put the data in an array, and then visualized it using matplotlib. Following are the results of the experiment.

**Experimental Results**



**Observations:**

* The chaining implementation consistently outperformed the other two performances, as predicted by the theoretical data.
* The chaining implementation performs at O(N), as predicted by the theoretical study. However, when the capacity size was initially smaller and comparable to the number of unique key values, linear probing and double hashing were initially performed at O(N2). Still, when the capacity expanded, they also performed at O(N).
* The hash table's starting size has an impact on performance as well. The graphs above show that as capacity rises, there is less of a performance gap between linear probing and double hashing, and after a capacity of 70000, linear probing begins to outperform double hashing. This is due to the fact that when capacity rises, the number of collisions begins to decline.
* Linear probing is preferable when the hash table's capacity is high and extra space cannot be used. If not, we can employ double hashing.
* Using the chaining implementation is always preferable if space is not an issue.
* Using the chaining approach is preferable when the hash table's capacity is quite low.

## 