*Exercise 3: Following our discussion in class about Hashtables provide the answers to*

*the following questions: (you can either provide a word document or a hand-written*

*document with the answers)*

*1. Define the following terms in your own words:*

• **Hashtable** - Is a data structure where you store data in an array that’s organized by a hashfunction determining the index.

• **Hashfunction**- is a function that takes a key value and uses that to determine which index of the array the data should be stored in.

• **Collision** - is when the hashfunction stores data at an index that’s already storing data.

*2. Research and provide the answer to the following question (please cite your*

*resources)*

**• What is the difference between a Hashtable and a Hashmap?**

A HashMap allows the presence of a null key and null values, while a Hashtable doesn’t allow any null keys/indexes or values.

Resource: <https://www.geeksforgeeks.org/differences-between-hashmap-and-hashtable-in-java/>

**• What is the desired running time for search, insert and delete in a**

**Hashtable?**

The desired running time, the average running time for Search, insert and Delete in a Hashtable is O(1). Because normally linear probing or chaining isn’t used to the point that to search for a data point, you have to go through the entire length of the array to circle around to where linear probing left them, normally collision is rare and data index points are where the hash functions initially pointed them to so they can be found, inserted and deleted almost instantaneously.

Resource: <https://programming.guide/hash-tables-complexity.html>

*3. Provide two different ways to deal with collision and briefly explain both of them.*

**Chaining**: Have each index be a LinkedList, and when collision occurs, simply store the data as a subsequent link in the LinkedList.

**Open Addressing**: When collision occurs, and the index determined by the hashfunction is already full, then you can instead store the data at the next available index. One way is to simply go down the list of indexes until you find one that’s empty.

*4. Similar to the fruits examples discussed in class, provide your own example that*

*illustrates storing strings using a hashfunction into a hashtable with linear probing.*

*(Your example should include collision scenarios)*

Storing a string of color names in a hashtable.

The hashfunction being the number of characters in the name String,, the length of the string mod 8, in case the name is longer than 7 characters.

Create Hashtable with array indexes ranging from 0-7

Add “red”, 3 characters so it goes into index 3

Add “blue”, 4 characters so it goes into index 4

Add “White”, 5 characters so it goes into index 5

Add “Purple” 6 characters so it goes into index 6

Add “Black”, 5 characters, but index 5 is already filled so it uses Linear Probing to Find a different index.

The subsequent index 6 is full so it goes to the next index

The subsequent index 7 is empty so “Black” goes into index 7.