Osama Tanveer
21801147
Section 3

Part 1: Describe briefly your design of the HashTable class and how you implement the collision resolution strategies (e.g., how to decide when to stop probing).

The original table is built using a dynamically allocated array. To keep track of the positions allocated, another dynamically allocated array is used. This array is named tableStatus. This is an int array whose individual indices correspond to the original table's positions. A **-1** at an index indicates an **unused** position in the table at the exact index, a **0** indicates a **removed** integer, and a **1** indicates a position currently **in use**. Moreover, the number of items in the table is stored in an int variable named itemsInTable. This variable is used when inserting - to check if the table is full or not, when removing - to check if there is any item present in table or not, and when analyzing to create an array of size itemsInTable to copy the items in table into this newly formed array. Apart from these, no extra data member is required.

Linear probing is implemented using the function provided i.e. the next position is checked until an empty location is found. When the end of the hashtable is reached, the loop wraps around the table and starts checking at position 0. Once the original location (calculated by the hash function) is reached, linear probing is stopped because all possible positions have been checked. The item is inserted if an empty location is found on this way, else it is not inserted. The loop terminates when the originally mapped location is reached after wrapping around.

Quadratic probing is also implemented using the provided function i.e. the next position is calculated using $hash(key) + i^2$ where i increases by 1 from 0. If an empty location is encountered during this calculation, the item is inserted. However, one problem arises that due to this a sequence forms for possible positions and this sequence oscillates once from left to right and then from right to left. For example, let's say we have a table of size 11. Continuous insertion of 2 would result in quadratic probing resulting in positions 2, 3, 6, 0, 7, 5 being calculated by the collision resolution function. On the next insertion, the function calculates 5 and then goes on to calculate 7, 0, 6, 3, 2. These two sequences are continually calculated. To prevent this infinite loop, the loop implemented terminates on the first occurence of the value calculated by the original hash function, in this case at 5. This prevents an infinite loop. Any further elements which result in mapping to the original location will not be inserted into the table. Therefore, in case of quadratic probing, even though the table might be partially empty with space to put some number, these numbers are not put because their position can not be determined due to their originally calculated hash value.

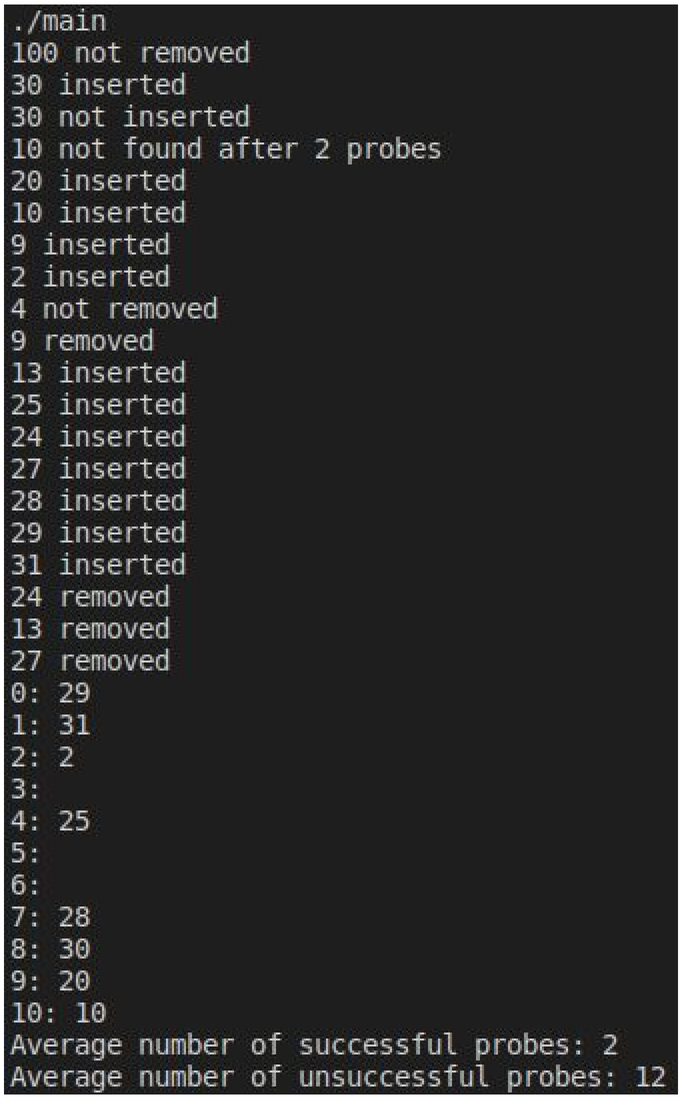
For Double Hashing, similarly, the formula provided is used. However, in this case, a terminating counter is kept, in case infinite loop happens. The terminating counter is set to twice the size of the table to ensure that enough iterations are allowed for the function to map to the required index. In case, the terminating counter is exceeded, the item to be inserted is not inserted. If, however, an empty space is found, the item is inserted. The terminating counter is set the same across the insert, search, and remove function to ensure that each follows a consistent methodology.

Part 2: Testing with input file

Table Size = 11
Contents of test file:

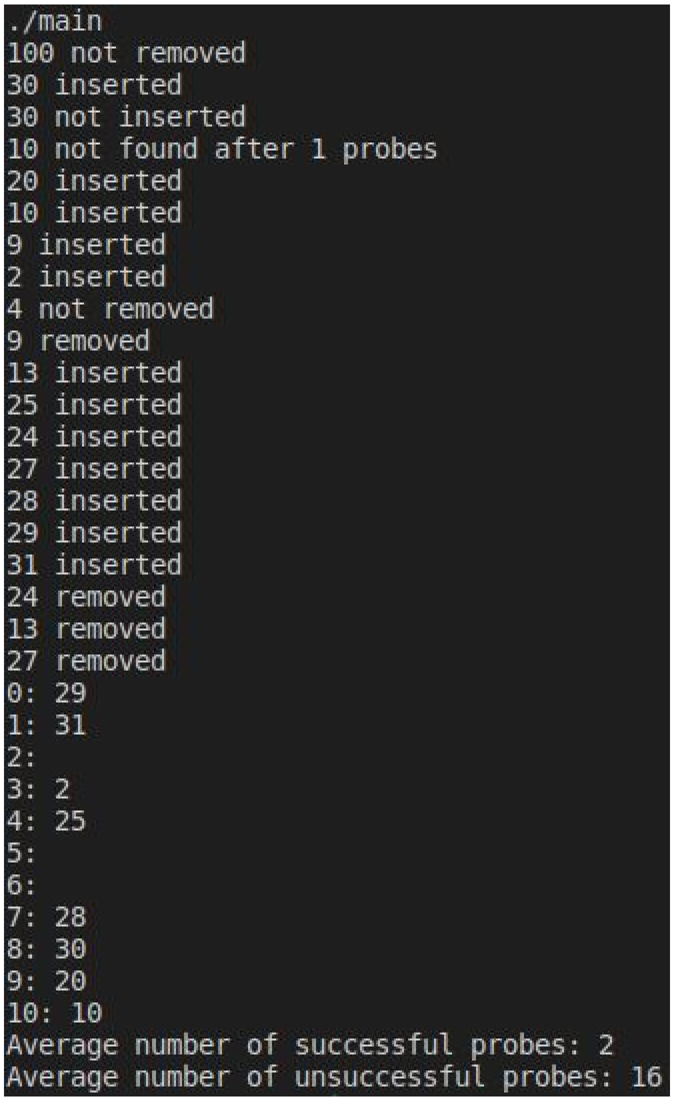
```
R 100
I 30
I 30
S 10
I 20
I 10
I 9
I 2
R 4
R 9
I 13
I 25
I 24
I 27
I 28
I 29
I 31
R 24
R 13
R 27
```

Using Linear Probing



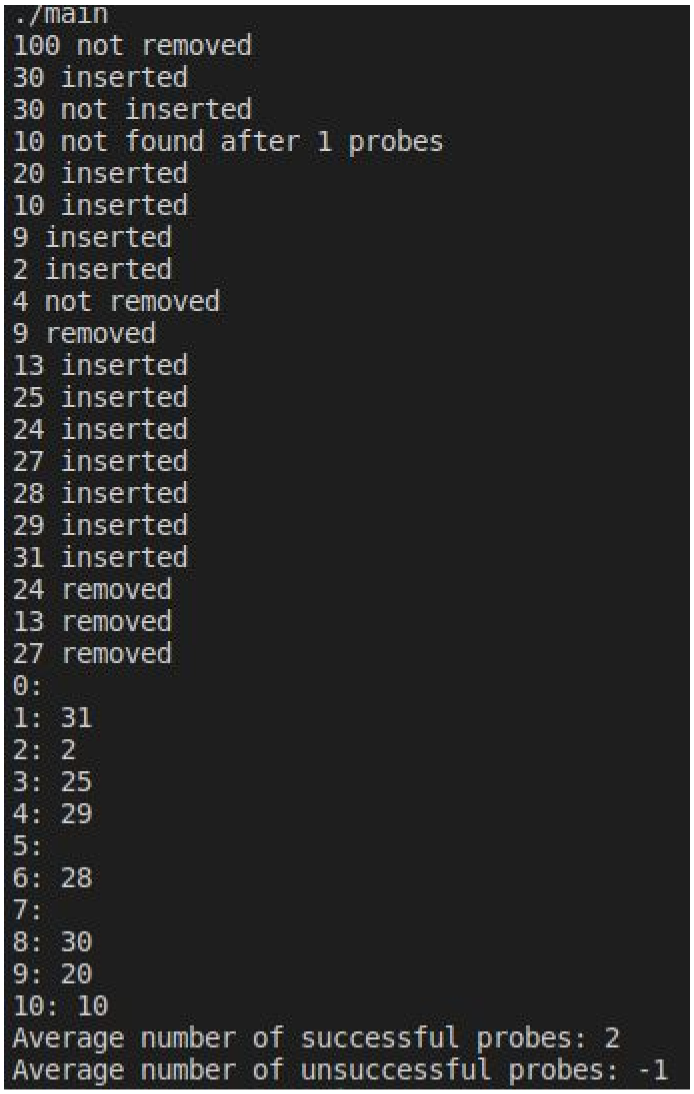
```
./main
100 not removed
30 inserted
30 not inserted
10 not found after 2 probes
20 inserted
10 inserted
9 inserted
2 inserted
4 not removed
9 removed
13 inserted
25 inserted
24 inserted
27 inserted
28 inserted
29 inserted
31 inserted
24 removed
13 removed
27 removed
0: 29
1: 31
2: 2
3:
4: 25
5:
6:
7: 28
8: 30
9: 20
10: 10
Average number of successful probes: 2
Average number of unsuccessful probes: 12
```

Using Quadratic Probing



```
./main
100 not removed
30 inserted
30 not inserted
10 not found after 1 probes
20 inserted
10 inserted
9 inserted
2 inserted
4 not removed
9 removed
13 inserted
25 inserted
24 inserted
27 inserted
28 inserted
29 inserted
31 inserted
24 removed
13 removed
27 removed
0: 29
1: 31
2:
3: 2
4: 25
5:
6:
7: 28
8: 30
9: 20
10: 10
Average number of successful probes: 2
Average number of unsuccessful probes: 16
```

Using Double Hashing



```
./main
100 not removed
30 inserted
30 not inserted
10 not found after 1 probes
20 inserted
10 inserted
9 inserted
2 inserted
4 not removed
9 removed
13 inserted
25 inserted
24 inserted
27 inserted
28 inserted
29 inserted
31 inserted
24 removed
13 removed
27 removed
0:
1: 31
2: 2
3: 25
4: 29
5:
6: 28
7:
8: 30
9: 20
10: 10
Average number of successful probes: 2
Average number of unsuccessful probes: -1
```

Part 3: Comparison with theoretical values

$\alpha = \frac{8}{11} = 0.72$

Linear Probing

Theoretical Values:

$$Successful\ Search = 1 + \frac{\alpha}{2} = 1 + \frac{0.72}{2} = 1.36$$
$$Unsuccessful\ Search = \alpha = 0.72$$

Empirical Values:

$$Successful\ Search = 2$$
$$Unsuccessful\ Search = 12$$

For successful search, it can be seen that the values are approximately equal. The difference may have been due to a small table being overly full and requiring more comparisons to reach the final required item.

For an unsuccessful search, the difference in values is caused by one major reason. When an item is removed from a position, the position status is set to 0, which indicates that the item has been removed. Since, to find the number of unsuccessful probes, we start from the position and end at an empty location. This empty location is indicated by a status of -1. If a status of 0 is found, it is still counted. Therefore, the count increases a lot.

Quadratic Probing

Theoretical Values:

$$Successful\ Search = \frac{-ln(1-\alpha)}{\alpha} = \frac{-ln(1-0.72)}{0.72} = 1.768$$
$$Unsuccessful\ Search = \frac{1}{1-\alpha} = 3.571$$

Empirical Values:

$$Successful\ Search = 2$$
$$Unsuccessful\ Search = 16$$

For successful search, the values are very close. Since the theoretical values are an ideal situation, the empirical value cannot equate it. The loop termination of quadratic probing depends on the check of repeating values, which may have resulted in a slight overcount.

For an unsuccessful search, the difference in values is caused due to the same reason as in linear probing. The removed item’s status is set to 0 instead of -1, therefore the iterations continue.

Double Hashing

<u>Theoretical Values:</u>

$$Successful\ Search = \frac{-ln(1-\alpha)}{\alpha} = \frac{-ln(1-0.72)}{0.72} = 1.768$$
$$Unsuccessful\ Search = \frac{1}{1-\alpha} = 3.571$$

<u>Empirical Values:</u>

$$Successful\ Search\ =\ 2$$
$$Unsuccessful\ Search\ =\ -1$$

For successful search, the values are very close. The empirical values do not represent an ideal situation and may have slightly increased due to more comparisons as a result of greater loop iterations, which may not exist in the ideal situation.