Assignment # 3

Kristopher Beauchemin - 40109581

Concordia University

COMP 352 - Data Structures and Algorithms

November 13th, 2020

Question #1:

Part A:

```
int maxDepth(Node* node)
{
    if (node == NULL)
       return 0;
    else
    {

       int lDepth = maxDepth(node->left);
       int rDepth = maxDepth(node->right);

       if (lDepth > rDepth)
             return(lDepth + 1);
       else return(rDepth + 1);
    }
}
```

Time Complexity: O(n)

- Since only need to check all nodes only once

Space Complexity: O(n)

- Since only need to store all nodes only once in the system stack due to recursion

Part B:

```
int CountFullNodes(Node* node)
{
    static int count = 0;
    if (node == NULL)
       return 0;
    else
    {
       if (node->left && node->right)
       {
             count++;
       }
       CountFullNodes(node->left);
       CountFullNodes(node->right);
    }
    return count;
}
```

Note: static int is only initialized once, at the beginning of execution

Time Complexity: O(n)

- Since only need to check all nodes only once

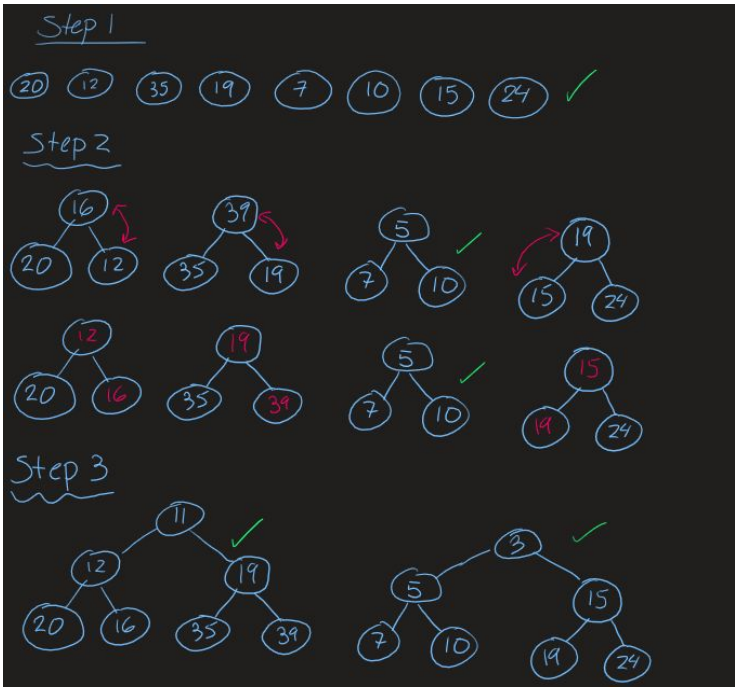
Space Complexity: O(n)

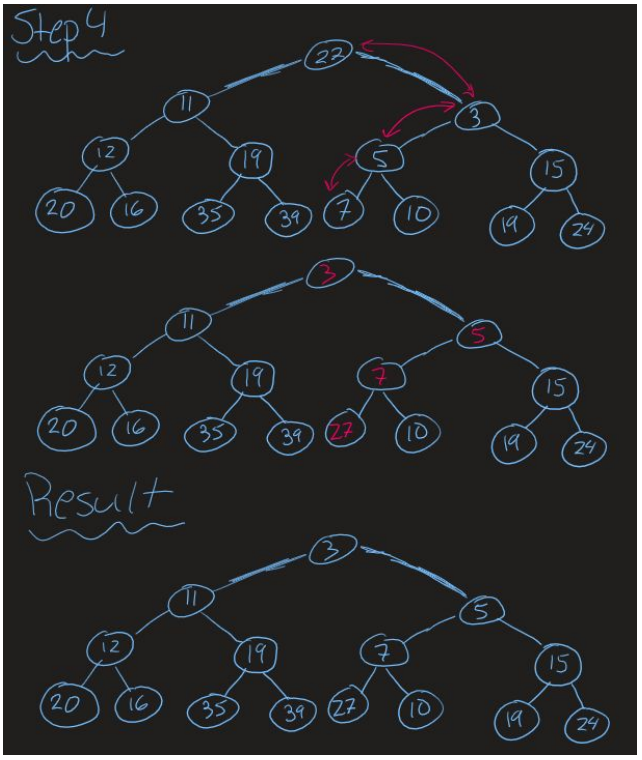
- Since only need to store all nodes only once in the system stack due to recursion

Question #2:

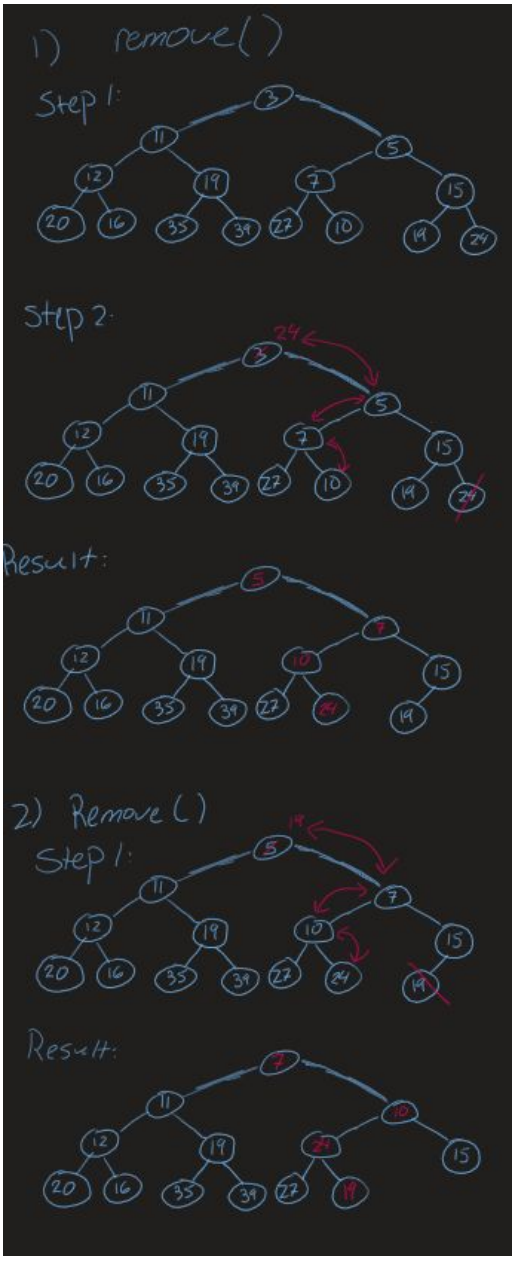
Part A:

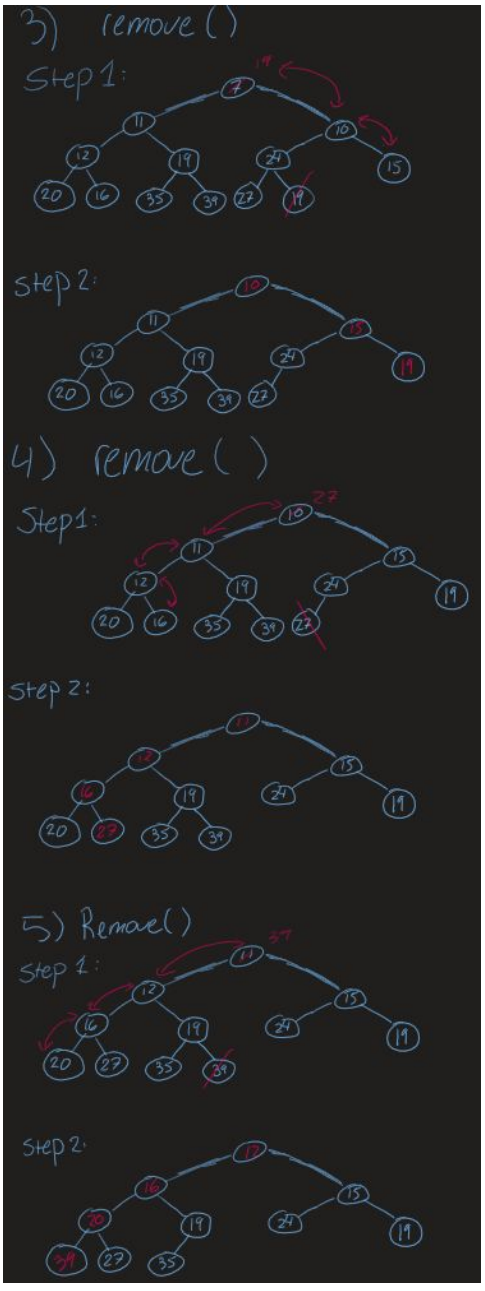
Part 1: Creating the Min Heap using Bottom Up Heap Algorithm:

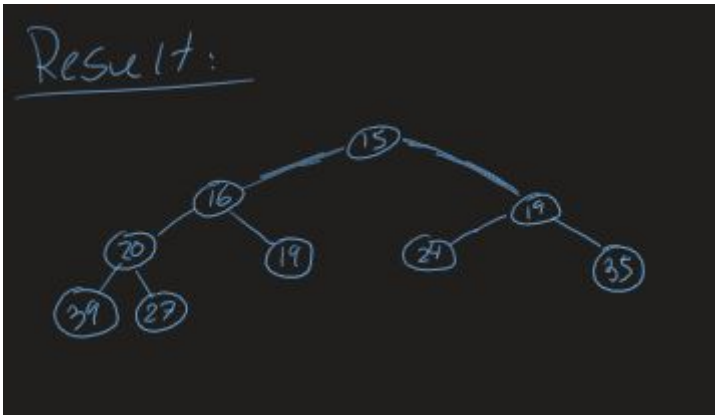




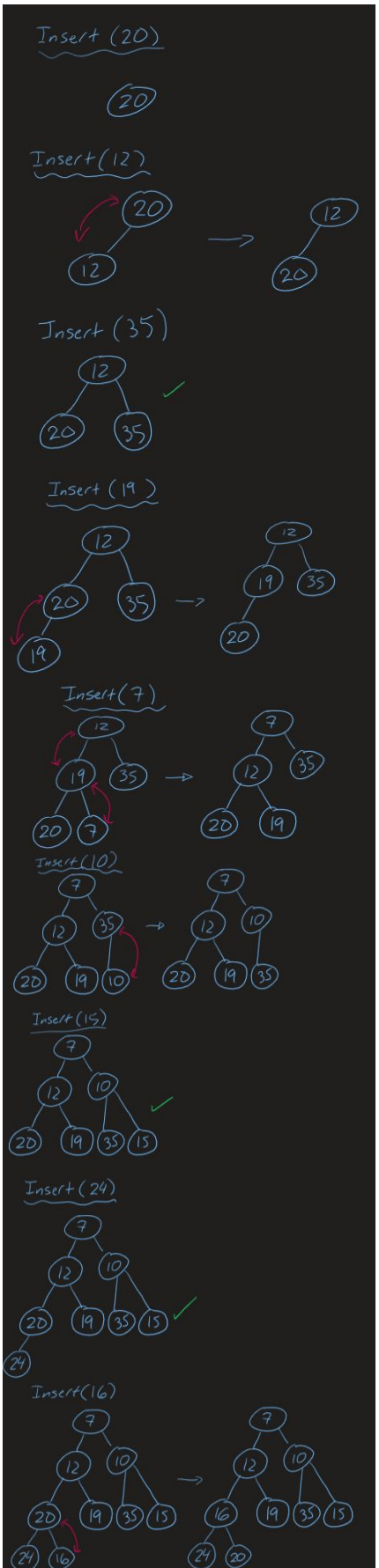
Part 2: Removing 6 elements:

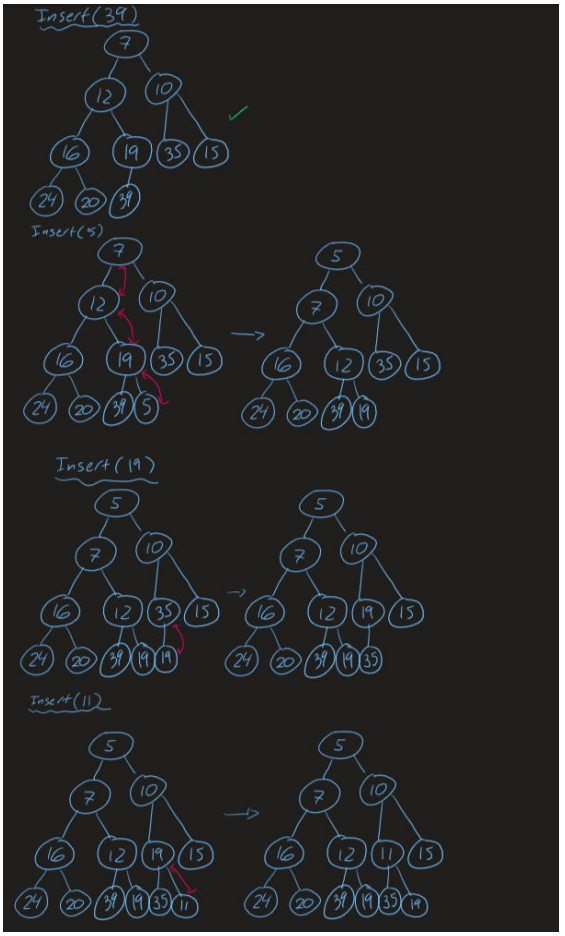


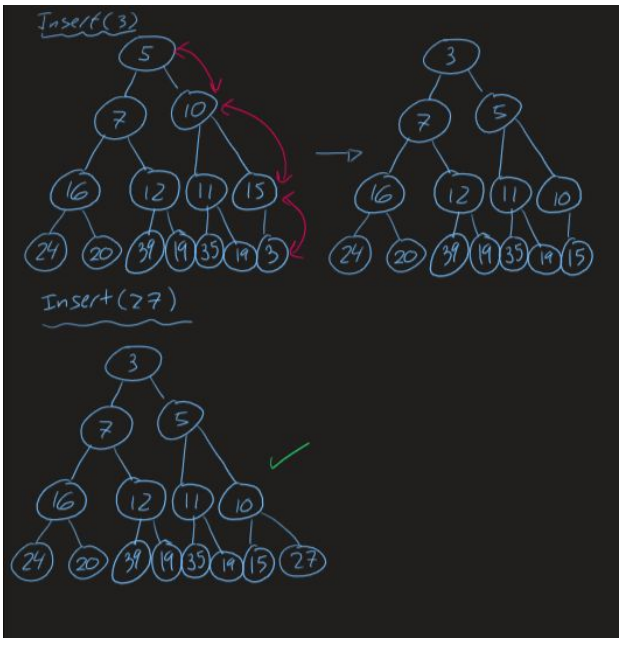




Part B:

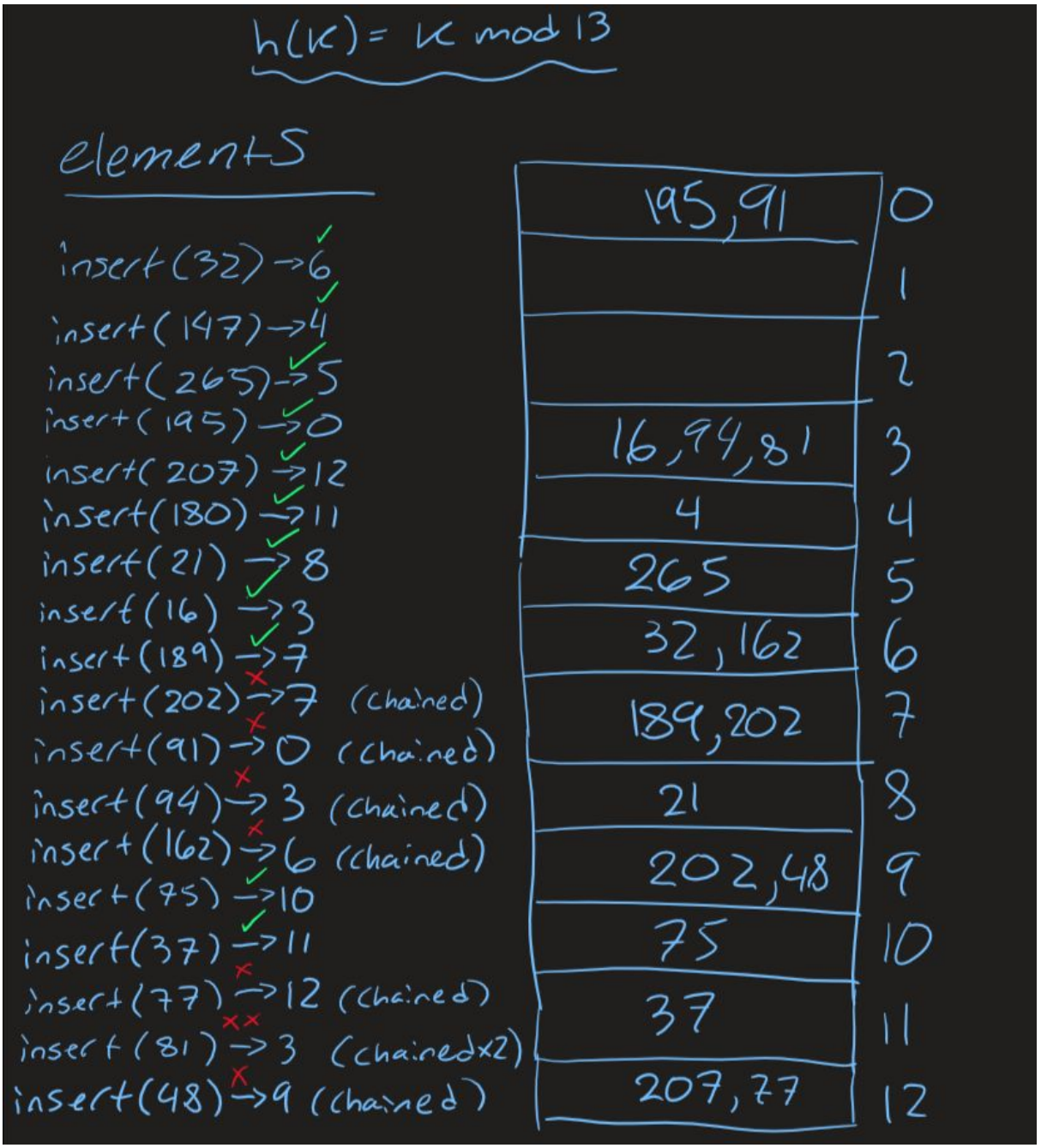

Insert (20)
Insert (12)
Insert (35)
Insert (19)
Insert (7)
Insert (10)
Insert (15)
Insert (24)
Insert (16)


Insert (39)
Insert (5)
Insert (19)
Insert (11)


Insert (3)
Insert (27)

Question #3:

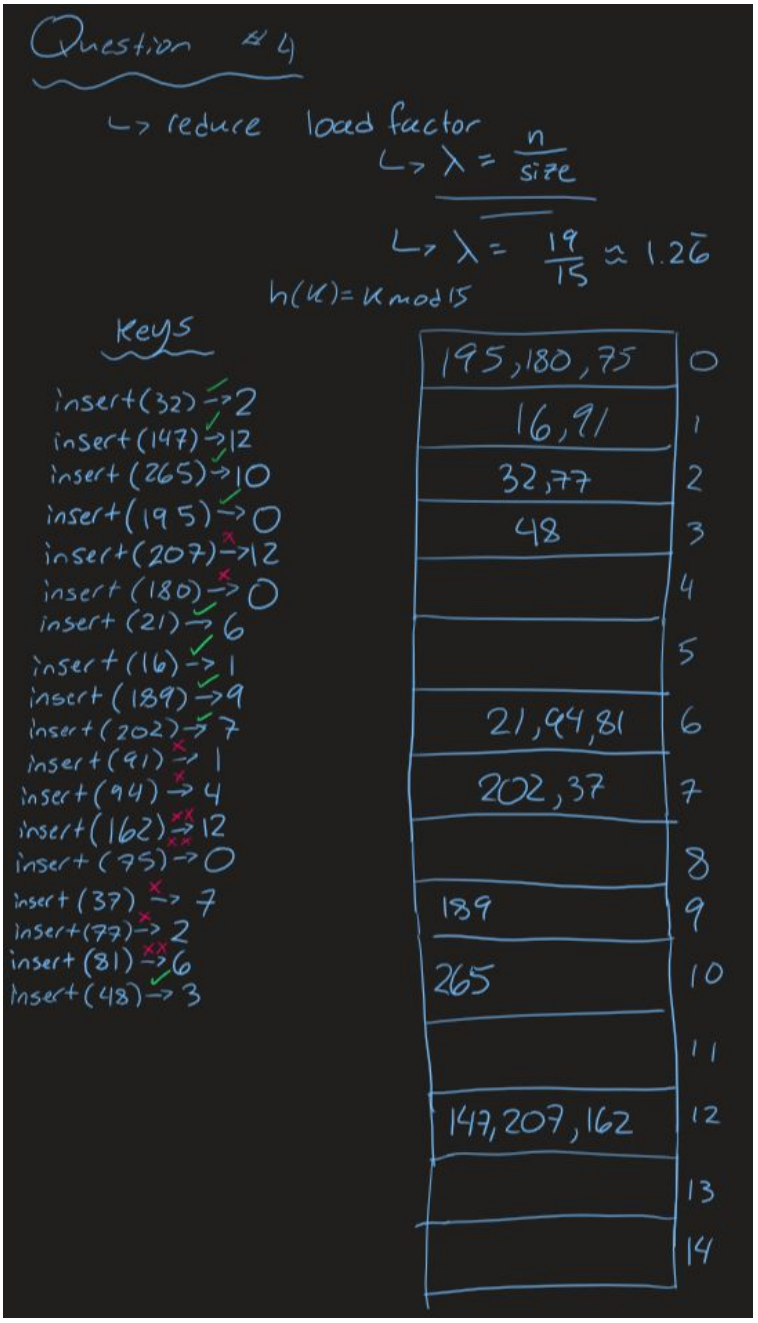
Part I:



Part II:

The above insertion caused exactly 8 collisions.

Question #4:



Total collisions: 12 (More than in Question 3)

Reducing the load factor is a good idea, but in this case, with the data set provided, the new hash function does not map the keys as well as it did in Question 3. In this case, we would want to keep the hash function the same and the length of the array as well. We would ideally want a different hash function in this case.

Question #5:

Part I:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
-	39	-	29	42	-	35	-	-	-	48	35	12	-	-	-	29	-	18

Part II:

Largest cluster is 3: Indices: 10, 11, 12

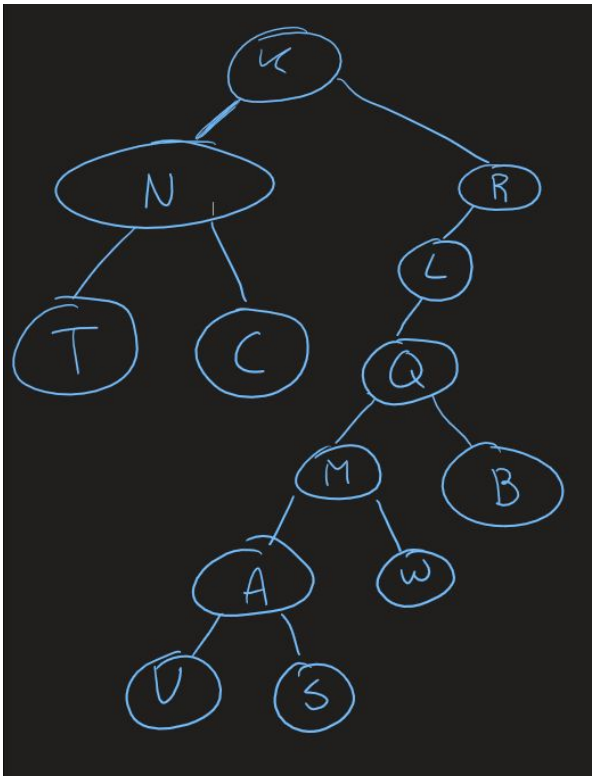
Part III:

Number of collisions: 10

Part IV:

Load Factor = 9 / 19 ~ 0.474

Question #6:



Programming Question:

Time Complexity:

ArrayHeap Implementation:

Big-O: $O(n)$ - Caused by Building the Heap
Big-Ω $\Omega(log_2(n))$ - Caused by Heapify Algorithm and insert for min heap

SortedList Implementation:

Big-O: $O(n)$ - Inserting an element in a sorted array to shift all elements.
Big-Ω $\Omega(n)$ - Caused by Insertion to array

Space Complexity:

ArrayHeap Implementation:

Big-O: $O(n)$ - Array being used as Heap is of size n.
Big-Ω $\Omega(n)$ - Array being used as Heap is of size n.

SortedList Implementation:

Big-O: $O(n)$ - Array being used for an ArrayList is of size n.
Big-Ω $\Omega(n)$ - Array being used for an ArrayList is of size n.

Difference Between Each Implementations:

Looking at the results between the ArrayHeap and the SortedList, the ArrayHeap finished 100k jobs in 7379ms and the SortedList finished in 50548ms. This indicates that the ArrayHeap is 585% faster than the SortedList implementation for 100k jobs!