Name: Anurag AGARWAL

Student ID: 40232644

Part 1

Q1.

Answer a:

Keys 19, 26, 13, 48 and 17 are inserted in hash table as:

For key 19, h(19) is 19%7 = 5. Therefore, 19 is placed at 5th index in the hash table.

For key 26, h(26) is 26%7 = 5. Therefore, 26 is placed at 5th index in the hash table after 19 as in a linkedlist chain.

For key 13, h(13) is 13%7 = 6. Therefore, 13 is placed at 6th index in the hash table.

For key 48, h(48) is 48%7 = 6. Therefore, 48 is placed at 6th index in the hash table after 13 as in a linkedlist chain.

For key 17, h(17) is 17%7 = 3. Therefore, 17 is placed at 3rd index in the hash table.

Output: (_,_,17,_,[19->26],[48->17],_)

Answer b:

Keys 19, 26, 13, 48 and 17 are inserted in hash table as:

For key 19, h(19) is 19%7 = 5. Therefore, 19 is placed at 5th index in the hash table.

For key 26, h(26) is 26%7 = 5. However, index 5 is already occupied with 19. Therefore, using linear probing, 26 will be placed at index 6.

For key 13, h(13) is 13%7 = 6. However, index 6 is already occupied with 26. Therefore, using linear probing, 13 will be placed at index 7.

For key 48, h(48) is 48%7 = 6. However, index 6 and 7 are already occupied with 26 and 13. Therefore, using linear probing, 48 will be placed at index 1.

For key 17, h(17) is 17%7 = 3. Therefore, 17 is placed at 3rd index in the hash table.

Output: (48,_,17,_,19,26,13)

Answer c:

Keys 19, 26, 13, 48 and 17 are inserted in hash table as:

For key 19, h(19) is 19%7 = 5. Therefore, 19 is placed at 5th index in the hash table.

For key 26, h(26) is 26%7 = 5. However, 5 is already occupied, hence we use our second function to calculate the index for 26 by using

the formula [h(k) + i*h'(k)]%7. So, [h(26) + 1*h'(26)]%7 = [5 + 1*(5-(26%5))]%7 = (9%7) = 2. Hence, 26 is placed at index 2.

For key 13, h(13) is 13%7 = 6. Therefore, 13 is placed at 6th index in the hash table.

For key 48, h(48) is 48%7 = 6. However, 6 is already occupied, hence we use our second function to calculate the index for 48 by using

the formula [h(k) + i*h'(k)]%7. So, [h(48) + 1*h'(48)]%7 = [6 + 1*(5-(48%5))]%7 = (8%7) = 1. Hence, 48 is placed at index 1.

For key 17, h(17) is 17%7 = 3. Therefore, 17 is placed at 3rd index in the hash table.

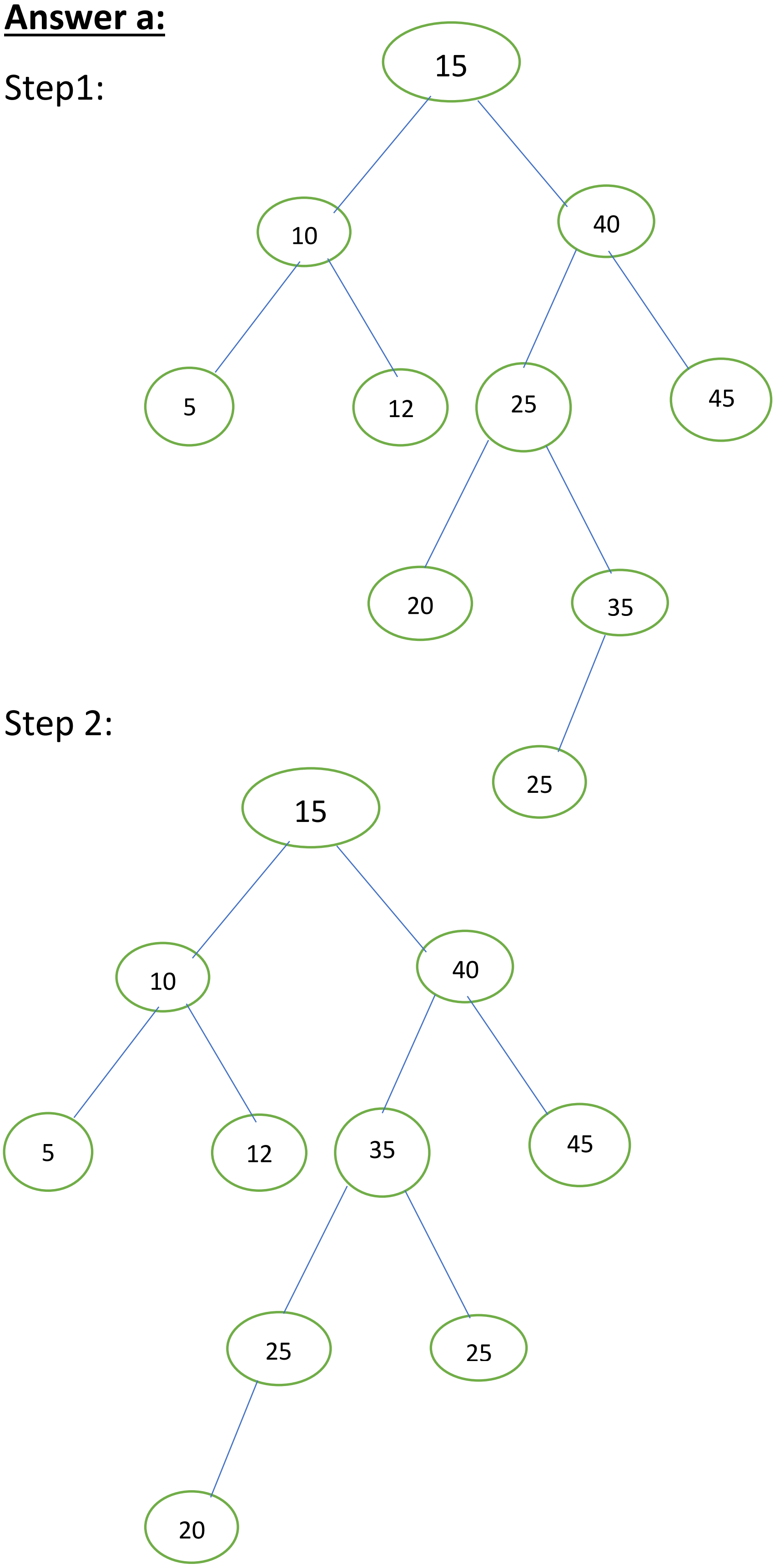
Output: (48,26,17,_,19,13,_)

Q2.

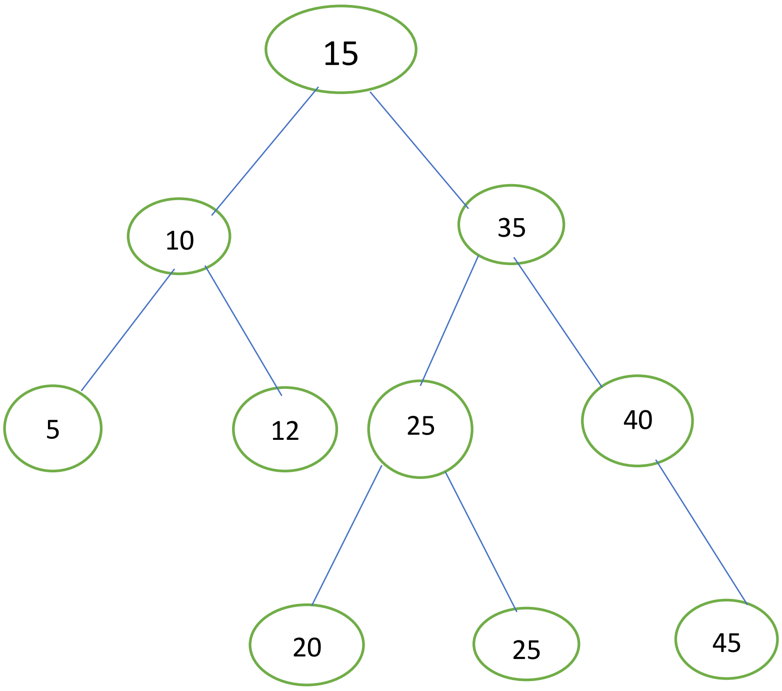
Answer a:

Step1:

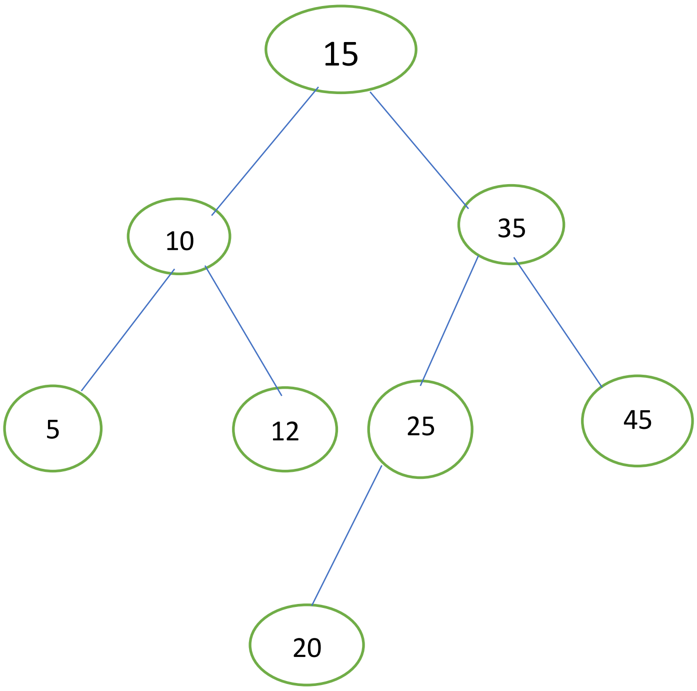
Step 2:



Step 3:



Answer b.



Q3.

Answer:

Resolving collision using a Pseudorandom Number Generator seeded with key.

```
RNG = new Pseudorandom Number Generator seeded with k.
nextNumber = next pseudorandom number generated by RNG
    index = nextNumber % M
    Loop infinitely:
        // check for duplicate insertions (not allowed)
        if arr[index] == k:
            return false
        // check if the slot of the array is empty (i.e., it is safe to insert)
        else if arr[index] == NULL:
            arr[index] = k
            return true
        // there is a collision, so re-calculate index
        else:
            nextNumber = next pseudorandom number generated by RNG
            index = nextNumber % M
        // all possible indexes are reached.
        if all M locations have been probed:
            throw an exception OR enlarge arr and rehash all existing elements.
```

Answer a.

No, the above approach of using Pseudorandom Number generated by the key will not be a better approach than Double Hashing.

Answer b.

No, as double hashing uses 2 functions to determine the index of the value which has lesser chances of repetition of indices, whereas in the above scenario moving in a circular loop is expected.