CS2210 Assignment : 3

Q1

A hash table of size N = 7
Given function : h(K) = K % 7

Collisions being handled by seperate chaining :

Inserting 12 :
hash(12) = 12 % 7 = 5
⇒ 12 is inserted at position 5

Inserting 27 :
hash(27) = 27 % 7 = 6
⇒ 27 is inserted at position 6

Inserting 5 :
hash(5) = 5 % 7 = 5
⇒ 5 is inserted at position 5

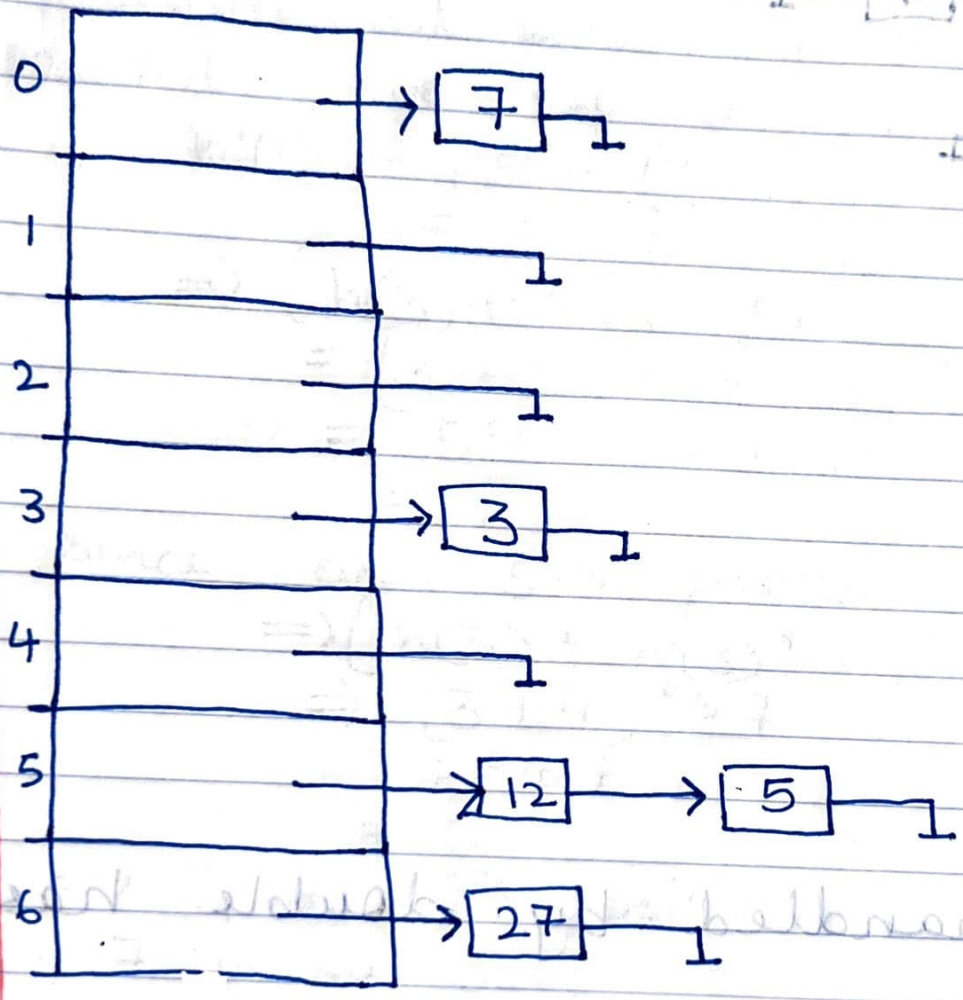
Inserting 3 :
hash(3) = 3 % 7 = 3
⇒ 3 is inserted at position 3

Inserting 7 :
hash(7) = 7 % 7 = 0
⇒ 7 is inserted at position 0

The hash Table after the insertion process is :

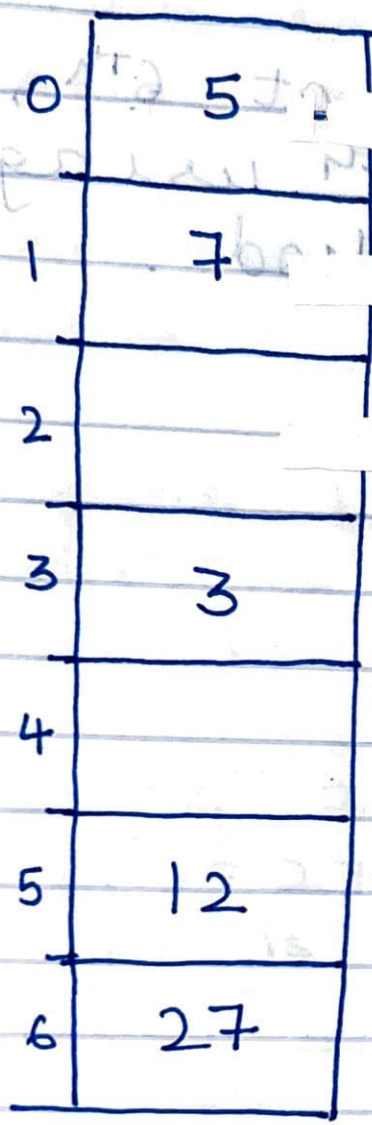
N = 7 ;

Insertions : 12 at 5th, 27 at 6th, 5 at 5th, 3 at 3rd, 7 at 0th using seperate collisions method.



Q2 Collisions handled by linear probing:
Insertion: 12 at 5th position; 27 at 6th position; 5 at 0 position {since there is already a value at 5th position, we go to 6th where again a value is present, then we shift to 'mod N' : 7%7=0; 3 at 3rd position, 7 at 1st position {since

we shifted 5 to the 0th position, we move up to the next position i.e., i | 3.



Q3 Collisions handled by double hashing:

Insertions:

- 12 at 5th position
- 27 at 6th position
- 5 supposed to be stored at 5th position but since the position is already occupied, we use secondary hash function:

$h'(5) \Rightarrow$ double hash value

$= 5 - (5 \% 5)$

$= 5 + 0 = 5$

$\rightarrow (h(5) + h'(5) * 1) \% 7$

$= (5+5) \% 7$

$= 10 \% 7$

$= 3$

$\Rightarrow$ 5 at 3rd position.

- 3 is supposed to be stored at 3rd position but since the position is already occupied, we use secondary hash function:

$h'(3) = 5 - (3 \% 5)$

$= 5 - 3 = 2$

$\Rightarrow (h(3) + h'(3) * 1) \% 7$

$= (3+2) \% 7$

$= 5 \% 7$

$= 5$

since even 5th position is occupied,

$\Rightarrow (h(3) + h'(3) * 2) \% 7$

$\Rightarrow (3+4) \% 7$

$= 7 \% 7$

$= 0$

$\Rightarrow$ 3 at 0th position.

- 7 is supposed to be stored at 0th position but since its already occupied, we use secondary hash function:

$h'(7) = 5 - (7 \% 5)$

$= 5 - 2$

$= 3$

~~since even 3rd position~~

$\Rightarrow (h(7) + h'(7) * 1) \% 7$

$= (0+3) \% 7$

$= 3 \% 7 = 3$

since even 3rd position is occupied,

$\Rightarrow (h(7) + h'(7) * 2) \% 7$

$\Rightarrow (0 + 6) \% 7$

$\Rightarrow 6$

since even 6th position is occupied,

$\Rightarrow (h(7) + h'(7) * 3) \% 7$

$\Rightarrow (0 + 9) \% 7$

$\Rightarrow 2$.

$\Rightarrow$ 7 is stored at the 2nd position.

0	3
1	
2	7
3	5
4	
5	12
6	27

Q4 $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

The time complexity of this algorithm

is given by :

$f(1) = c_0$ — *

$f(n) = f(n-1) + c_1 n + c_2, \quad \forall\ n > 0$

where c_1, c_2, c_0 are constants.

for the above recurrence equation,

let $f(n) = f(n-1) + c_1 n + c_2$ be (1)

let $f(n-1) = f(n-2) + c_1(n-1) + c_2$ be (2)

put $f(n-1)$ in $f(n)$ (i.e., in (1)).

$\Rightarrow f(n) = f(n-2) + c_1 n + c_1(n-1) + 2c_2$

$= f(n-2) + 2c_1 n - c_1 + 2c_2$ — (3)

we know $f(n-2) = f(n-3) + c_1(n-2) + c_2$

substituting $f(n-2)$ in (3)

$\Rightarrow f(n) = f(n-3) + 2c_1 n - c_1 + c_1(n-2) + 3c_2$

$f(n) = f(n-3) + 3c_1 n - 3c_1 + 3c_2$

let $3c_2 - 3c_1$ be 'kl' {constant}.

$f(n) = f(n-3) + 3c_1 n + kl.$ — (4)

Similarly, for any $n = n-k$ where $k > 0$

$f(n) = f(n-k-1) + (k+1)c_1 n + kl$

{where kl is constant}

Similarly for $n = n-k-1$

$f(n-k-1) = f(n-k) + c_1 nk + kl$ ... from eqn 4

Substituting $n-k=1$ as $k=n-1$ to find $f(1)$

$$f(n) = f(1) + (n-1)c_1 n + c$$

$$f(n) = c_0 + c_1 n^2 - c_1 n + c$$

{from eqn (*)}

let 'b' be a constant

$$b = c_0 + c$$

$$f(n) = c_1 n^2 - c_1 n + b$$

$\Rightarrow$ The time complexity of the algorithm is therefore n^2.

i.e., $T(n)$ of $f(n) = O(n^2)$.

Q5

Algorithm : totalLeaves (r, level)
In : root 'r' of a tree & 'level' an Integer value
Out : sum of distances from all leaves to the root of the tree

```
sum ⟶ level
if r.isLeaf() is true
    then return sum
else
{ level ⟶ level + 1
  for each child c of r do
  { sum → sum + totalLeaves (c, level) }
}
```

The time complexity of the above algorithm is ~~Ω(n)~~ O(n).
The worst case would be calculated by O(n). Time complexity of algorithm quantifies the amount of time taken by the algorithm to run as a function of the size of the input to the problem ~~T(n) is Θ(n)~~ It is O(n) because it has to traverse through all nodes in the for loop.

Q6 The given algorithm of the method algo which has 3 parameters : A, B, n where A and B are two arrays of size n. Inside the function, we declare two loop variables i and j and initialize them to 0. Using a while loop, iterate till j becomes equal to n. we achieve this by assigning A[j] to B[i] and since both loop variables are zero, the first element of both arrays becomes same.
Then, we check of the corresponding same elements of both arrays and if condition is true then i is ~~incremented by 1~~ assigned n-1. For first case i = 0, so here i becomes n-0, else i is incremented by 1 Then and j becomes i+1 which or n+1 since i is n. This is greater than n (n+1 > n). This breaks the while loop and exits.

Here time complexity is O(1).