29/2/24

Hacker rank :

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
  int id;
  int depth;
  struct node *left, *right;
};

void inorder(struct node* tree)
{
   if(tree = Null)
        return;
   inorder(tree->left);
   printf("%d", tree->id);
   inorder((tree->right));
 }

int main(void)
{
    int no_of_nodes, i=0;
    int l,r, max_depth, k;
    struct node* temp = NULL;
    scanf("%d", &no_of_nodes);
    struct node* tree = (struct node*) calloc (no_of_nodes,
                                    sizeof(struct node));

    tree[0].depth =1;
    while(i< no_of_nodes)
    {
       tree[i].id = i+1;
       scanf("%d %d", &l, &r);
       if(l == -1)
            tree [i].left = NULL;
       else
            {  tree[i]. left = &tree[l-1];
               tree[i]. left -> depth = tree[i].depth + 1;
               max_depth = tree [i]. left ->depth;
            }
if(r == -1)
    tree[i].right = NULL;
else
    {
        tree[i].right = &tree[r-1];
        tree[i].right->depth = tree[i].depth + 1;
        max_depth = tree[i].right->depth+2;
    }
i++;
}
scanf("%d", &i);
while (i--)
{
    scanf("%d", &k);
    r = l;
    while(l <= max_depth)
    {
        for(K = =0; K < no_of_nodes; ++K)
        {
            if (tree[K].depth == l)
            {
                temp = tree[K].left;
                tree[K].left = tree[K].right;
                tree[K].right = temp;
            }
        }
        l = l +r;
    }
    inorder(tree);
    printf("\n");
}
return 0;
}
```

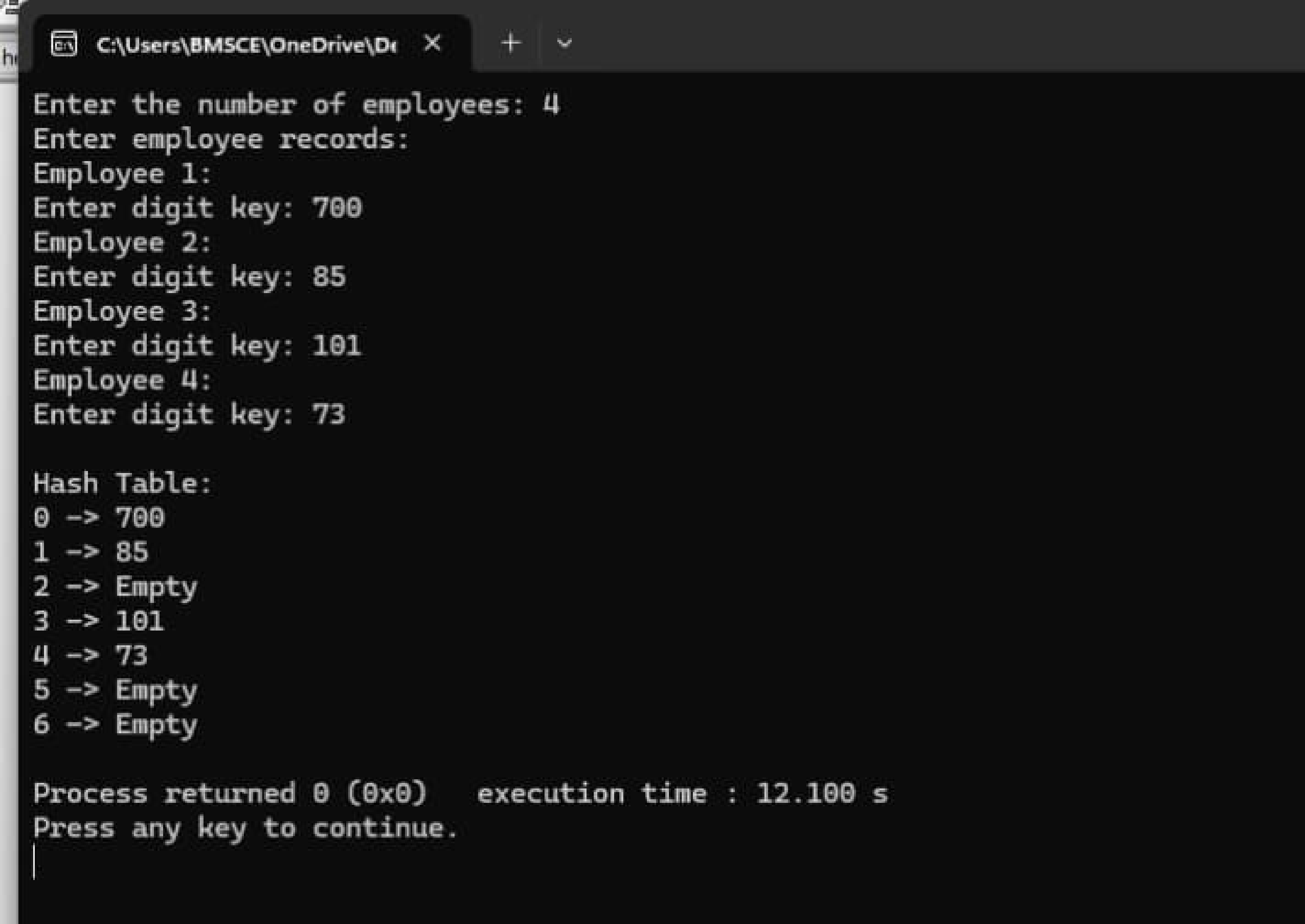
```
o/p:  input:-
      3                     3 1 2
      2   3                 2 1 3
      -1  -1
      -1  -1
      2
      1
      1
```

Week 10 29/2/24

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_EMPLOYEES 100
#define HASH_TABLE_SIZE 7

struct Employee
 {
   int Key;
 };
int hashFunction(int Key);
void insertEmployee(struct Employee employees[], int
       hashTable[], struct Employee emp);
void displayHashTable(int hashTable[]);

int main()
 {
   struct Employee employees[MAX_EMPLOYEES];
   int hashTable[HASH_TABLE_SIZE] = {0};
   int n, m, i;
   printf("Enter the number of employee:");
   scanf("%d", &n);
   printf("Enter the number of employees:");
   scanf("%d", &n);
   printf("Enter employee records:\n");
   for (i=0; i<n; ++i) {
           printf("Employee %d:\n", i);
           printf("~~Employee~~ Enter 4-digit Key");
           scanf("%d", &employees[i].Key);
           insertEmployee(employees, hashTable,
                                  employees[i]);
      }

  printf("\nHash Table:\n");
  displayHashTable(hashTable);
  return 0;
}
```

```
int  hashFunction(int key)
   {
     return key % HASH_TABLE, SIZE ;
   }
void insertEmployee(struct Employee employees[], int
          hashTable[], struct Employee emp)
   {
     int  index = hashFunction(emp.key);

  while (hashTable [index] ! = 0)
     {
       index = (index + 1) % HASH_TABLE_SIZE;
     }
   hashTable[index] = emp.key;
 }
void displayHashTable (int  hashTable[])
   {
     int i;
     for(i = 0;  i < HASH_TABLE_SIZE ; ++i)
       {
         printf("%d ->" , i);
         if( hashTable[i] = = 0)
           {
             printf("Empty\n");
           } else
             {
               printf(" %d \n", hashTable [i]);
             }
       }
   }
```

[illegible]



```
Enter the number of employees: 4
Enter employee records:
Employee 1:
Enter digit key: 700
Employee 2:
Enter digit key: 85
Employee 3:
Enter digit key: 101
Employee 4:
Enter digit key: 73

Hash Table:
0 -> 700
1 -> 85
2 -> Empty
3 -> 101
4 -> 73
5 -> Empty
6 -> Empty

Process returned 0 (0x0)   execution time : 12.100 s
Press any key to continue.
```