Aryan Garg
19124018
IT(G-1)

DSA Lab Assignment
(Hashing & Sorting)

Q1.)

→ Insertion Sort

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

void Insertion(int A[], int n) // function for insertionSort
{
    int i, j, x;
    for(i=1 ; i<n ; i++)
    {
        j=i-1;
        x=A[i];
        while(j>-1 && A[j]>x)
        {
            A[j+1]=A[j];
            j--;
        }
        A[j+1]=x;
    }
}

int main(){
    int A[] = {11,34,12,8,4,23}; // taking the array and then the size of array =n
    int n=6;
    int i;      // index of array to be traversed
    printf("Initial Array:");
    for(i=0 ; i<n ; i++)
    {
        printf("%d", A[i]);
    }
    printf("\n")
    Insertion(Arr); // for call
    printf("Array After Sorting:");
    for(i=0 ; i<n ; i++)
        printf("%d", A[i]); // printing array's elements
    return 0;
}
```

Q2:)

→ Bubble_Sort

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

void swap (int *x, int *y) // fxn for swapping
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
void Bubble (int A[], int n) // fxn for Bubble sort
{
    int i, j, flag = 0;
    for (i=0; i<n-1; i++)
    {
        flag = 0;
        for (j=0; j<n-i-1; j++)
        {
            if (A[j] > A[j+1])
            {
                swap (&A[j], &A[j+1]);
                flag = 1;
            }
        }
        if (flag == 0)
        break;
    }
}
int main()
{
    int A[] = {23, 12, 67, 23, 3, 98}; //array of elements of size 6
    int n = 6; // size of array
    int i; //index of array to be traversed
    printf ("Initial Array: ");
    for (i=0; i<n; i++)
    {
        printf ("%d ", A[i]); // initial array before sorting
    }
```

```
printf("\n");
Bubble(A,n);  //call of fun.
printf("Array After sorting:");
for(i=0;i<n;i++)
    printf("%d", A[i]); // printing array's sorted elements.
printf("\n");
return 0;
}
```

Q3.) →

(4)

Selection Sort →

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

void swap(int *x, int *y)//fxn for swapping
{
    int temp=*x;
    *x=*y;
    *y=temp;
}
void selectionSort(int A[], int n) //fxn of selection sort
{
    int i,j,k;
    for(i=0;i<n-1;i++)
    {
        for(j=k=i;j<n;j++)
        {
            if(A[j]<A[k])
                k=j;
        }
        swap(&A[i],&A[k]);
    }
}
int main()
{
    int A[]={78,1,13,24,90,2};  //array of the elements
    int n=6; //size of Array
    int i; //index of array for traversing the array
    printf("Initial Array:");
    for(i=0;i<n;i++)
    {
        printf("%d ",A[i]); //printing initial array's elements
    }
    printf("\n");
    selectionSort(A,n); //call of the fxn
    printf("Array After sorting:"); //printing the elements after sorting of the array
    for(i=0;i<n;i++)
```

```
        printf("%d ", A[i]);
    printf("\n");
    return 0;
}
```

Q4.)→

Linear Probing →

```
#include<stdio.h>
#include<stdlib.h>
#define Size 10  //taking the size of the array to 10.

int hash(int Key)
{
  return Key % Size;  //return the place that value should be the key is placed
}                                                    logically
int probe(int H[], int Key)  //probing the element in the hash table
{
    int index = hash(Key);
    int i = 0;
    while(H[(index+i) % Size] != 0)
      i++;
    return (index+i) % Size;  //if it can't be placed at that place..
}
void Insert(int H[], int Key)  //insert in the hash table
{
    int index = hash(Key);
    if(H[index] != 0)
        index = probe(H, Key);
    H[index] = Key;
}
int Search(int H[], int Key)  //searching inside the hash table.
{
    int index = hash(Key);
    int i = 0;
    while(H[(index+i) % Size] != Key)
        i++
    return (index+i) % Size;
}
int main()
```

```
int HT[10] = {0};  // HashTable is initialized to 0
int i;  // to see the index i for traversing through the hash Table.
// insert into the hashTable
Insert(HT, 12);
Insert(HT, 25);
Insert(HT, 35);
Insert(HT, 26);
printf("Keys in the Hash Table Are:\n");
for(i=0; i<10; i++){
    printf("\n");
    printf("%d", HT[i]); // displaying all the elements of hashTable.
}
printf("\n key %d found at index %d\n", 12, search(HT, 12));
// searching in the HashTable for a key.
return 0;
}
```

Q5.) Double Hashing →

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10 // table size has been taken to 10
int h[TABLE_SIZE] = {NULL}; //Initially the entire array elements and values is defined to NULL
void insert() //function for inserting values to hash Table.
{
  int key, index, i, flag=0, hkey, hash2;
  printf("\nEnter a value to insert into hash table:\n");
  scanf("%d", &key); //scanning the input value
  hkey = key % TABLE_SIZE; //taking index where element will be found in table
  hash2 = 7 - (key % 7);
  for(i=0; i<TABLE_SIZE; i++)
  {
    index = (hkey + i * hash2) % TABLE_SIZE; //hash fxn.
    if (h[index] == NULL)
    {
      h[index] = key;
      break;
    }
  }
  if (i == TABLE_SIZE) // Element cant be inserted in table.
    printf("\nElement cannot be inserted in the hash Table:\n");
}
void search() { // fxn for searching of the Element in hash Table.
  int key, index, i, flag=0, hash2, hkey;
  printf("\nEnter the Element to be searched:\n");
  scanf("%d", &key); //taking in the element which needs to be found
  hkey = key % table_size;
  hash2 = 7 - (key % 7);
  for(i=0; i<TABLE_SIZE; i++)
  {
    index = (hkey + i * hash2) % table_size;
    if (h[index] == key) //value of element is found.
    {
      printf("Value of Element is found at index %d", index);
      break;
    }
```

```
}
if (i==table_size) //value of element is not found
    printf("\n Value is not found inside the hash table.\n");
}
void display() //fxn to display all the elements
{
    int i;
    printf("\n Elements in the hash table are:\n");
    for(i=0; i<TABLE_SIZE; i++)
        printf("\n Element at index %d is value = %d", i, A[i]);
}
int main() {
    int opt, i; //finding the option for what to do in the table.
    while(1)
    {
        printf("\n Enter Your Choice :");
        printf("\n Press \n1. Insert \n2-Display \n3-Search \n4-Exit\n");
        scanf("%d", &opt);
        switch(opt)
        {
            case 1:
                Insert(); //Calling Insert fxn.
                break;
            case 2:
                display(); //Calling Display fxn.
                break;
            case 3:
                search(); //Calling Search fxn.
                break;
            case 4:
                exit(0); //Exit from Table.
        }
    }
    return 0;
}
```