Practical no 1

''' ⭕ **PROBLEM STATEMET:-**

Consider a telephone book database of N clients. Make use of a hash table implementation to quickly look up client‘s telephone number. Make use of two collision handling techniques and compare them using number of comparisons required to find a set of telephone numbers (Python) '''

'''

 # QUICK REVISION NOTES:-

Linear probing: One searches sequentially inside the hash table.

val = (data+i) % size

ADVANTAGE:-

1. No extra space.

DISADVANTAGE:-

1. Searching time:- O(n)- worst case

O(1)- Avg,Best case

2. Deletion Difficulty.

3. Primary/Secondary Clustring

Quadratic probing: One searches quadratically inside the hash table.

val = (data+i*i) % size

ADVANTAGE:-

1. No extra space.

2. Primary Clusterd resolved.

DISADVANTAGE:-

1. Searching time:- O(n)- worst case

O(1)- Avg,Best case

2. No guarenty of finding an empty slot.

3. Secondary Clustring :- Two or more keys following the same probing the sequence.

```
'''

size = int(input("Enter size of Hash Table : "))
array1 =[]
array2 =[]

for i in range(size):
    array1.append(None)
    array2.append(None)



# Insertion Using Linera Probing

def insert_LineraProbing(data):
    i=0
    count=1
    value = (data+i)% size
    while (array1[value] != None):
        value = (data + i)% size
        i =i+1
        count = count +1
    array1[value] = data
    display_LinearProbing()
    print("Number of comparisons : ", count)

# Insertion Using Quadriatic probing

def insert_quadriaticProbing(data):
    i =0
    count=1
    value = (data + (i*i)) % size
```

```
    while (array2[value]!=None):
        if (count > 2*size):
            print("opps, Index Not Found....")
            break
        value = (data + (i*i)) % size
        i = i+1
        count = count +1
    array2[value]= data
    display_QuadriaticProbing()
    print("Number of Comparisons : ",count)

def display_LinearProbing():
    print("Linear Probing: ")
    print(array1)

def display_QuadriaticProbing():
    print("Quadriatic Probing : ")
    print(array2)



def search1(data):
    i =0
    count=1
    value = (data+(i))%size
    while (array1[value] !=None):
        if (array1[value]==data):
            print("Telephone Number Found ")
            print("Number of Comparisons in Linear Probing are : ",count)
            break
        elif(count>size*2):
            print("NO ELEMENT FOUND")
```

```
            break
        value = (data+(i))% size
        i = i+1
        count = count +1
    if(array1[value]==None):
        print("Opps, Element Not Present")
        print("Number of Comparisons : ",count)

def search2(data):
    i=0
    count=1
    value=(data+(i*i))%size
    while(array2[value]!=None):
        if (array2[value]==data):
            print("Telephone number found ")
            print("Number of comparisons in Quadriatic Probing are : ",count)
            break
        if(count > 2*size):
            print("Opps, Cannot Fetch Index")
            break
        value=(data+(i*i))%size
        i=i+1
        count=count+1
    if(array2[value]==None):
        print("No such Element Found")
        print("Number of comparisons : ", count)

while(True):
    print("1. Insert    2. Search")
    choice=int(input("Enter Choice : "))
    if(choice == 1):
```

```
        data1= int(input("Enter Telephone Number : "))
        print("\n")
        insert_LineraProbing(data1)
        insert_quadriaticProbing(data1)
        print("\n")

    if(choice==2):
        data1=int(input("Enter Telephone Number : "))
        print("\n")
        search1(data1)
        search2(data1)
        print("\n")
```

output

Enter size of Hash Table : 2

1. Insert 2. Search

Enter Choice : 1

Enter Telephone Number : 5555555552

Linear Probing:

[5555555552, None]

Number of comparisons : 1

Quadriatic Probing :

[5555555552, None]

Number of Comparisons : 1

1. Insert 2. Search

Enter Choice :