**Practical No.1**

**Title:**

Consider 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.

**Objective:**

* To make use of two collision handling techniques
* To compare them using number of comparisons required to find a set of telephone numbers.

**Source Code:**

def tele\_database():

phone\_data = []

n = int(input("Enter Number of Clients :- "))

print("Enter Phone Numbers :-\n")

for \_ in range(n):

x = int(input("--> "))

phone\_data.append(x)

return phone\_data

def hash\_function\_1(key\_ele, m\_size):

h1 = key\_ele % m\_size

return h1

def hash\_function\_2(key\_ele):

h2 = 1 + (key\_ele % 11)

return h2

def hashtable(ht):

print(f"\nHash Value \tKey")

for ele in range(len(ht)):

if ht[ele] != -1:

print(f"\n\t{ele} \t---> \t{ht[ele]}")

else:

print(f"\n\t{ele}")

phone\_database = tele\_database()

m = int(input("Enter Hash Table Size :- "))

hash\_table = [-1] \* m

opt = int(input("If collision occurs which collision resolution technique do you want to use?\n\t1. Linear "

"Probing\n\t2. Double Hashing :- "))

for k in phone\_database:

h\_1 = hash\_function\_1(k, m)

h\_2 = hash\_function\_2(k)

if hash\_table[h\_1] == -1:

hash\_table[h\_1] = k

else:

if opt == 1:

while hash\_table[h\_1] != -1:

h\_1 = (h\_1 + 1) % m

hash\_table[h\_1] = k

elif opt == 2:

i = 0

while hash\_table[h\_1] != -1:

i += 1

h\_1 = (h\_1 + i \* h\_2) % m

hash\_table[h\_1] = k

hashtable(hash\_table)

**Output:**

