Perfect Hashing
Lab4

Name	ID
Habiba Osama Zaky	19015575
Mai Ahmed Hussien	19016736

1-Problem Statement:

Constructing Universal hash Family by picking h matrix (bxU) randomly and multiplying by the value of x in binary and then get the index of the value. Define h(x)=hx.Where we do addition mod 2.with b bits ->M=2^b.

$$\overset{h}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}} \overset{x}{\begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}} = \overset{h(x)}{\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}}$$

O(N^2)-Space Solution

Say we are willing to have a table whose size is quadratic in the size N of our dictionary S. Then, here is an easy method. Let H be universal and M = N^2. Pick a random h from H and try it out, hashing everything in S. So, we just try it, and if we got any collisions, we just try a new h. On average, we will only need to do this twice.

O(N)-Space Solution

The main idea for this method is to use universal hash functions in a 2-level scheme. The method is as follows. We will first hash into a table of size N using universal hashing. This will produce some collisions. However, we will then rehash each bin using Method 1, squaring the size of the bin to get zero collisions. So, the way to think of this scheme is that we have a first-level hash function h and first-level table A, and then N second-level hash functions h1, ..., hN and N second-level tables A1, ..., AN . To look up an element x, we first compute i = h(x) and then find the element in Ai[hi(x)].

2- Implementation explanation

The O(n^2) method:

1) generate random H whose number of rows =log(n^2)/log(2) & number of columns = U=32bits.

2) get the value of an element in binary and multiply it by random H.

3) get the index of an element from the multiplication if in that index there exists another element so it is collision then repeat the process again from the beginning until find a hash function that is suitable for all elements.

The O(n) method:

First, we generate a hash function,which is multiplied by every entered element in binary rep. And change the result into decimal again to get an index. Then we put the elements which got the same index in an arraylist in an array which is mapped to by that index, so all the elements of the same indes are in the same arraylist.

Secondly, we take this array of arraylists and send each non-empty arraylist to the first method (N^2). We store each result from the first method in a 2d array named “finalHashTable” (each resulting second hash table in its index location of the first hash table).Also, we save the generated hash functions of each array.

For the look up, we use the already generated hash function of the second method and get the element’s ,to be searched for, binary rep. And multiply them and change the result into decimal to get the index. Check if that index exists, if exists, we get the hash function used for the elements who had this index before and multiply it with the binary rep of that element to get another index and then if the element was found in that index in the second hash table then we print it and it’s place, else it will print “Not Found!!”.

Else if doesn't exist, we print “Not Found!!”.

The Classes in our code :

- In the Hashing class:

We construct the universal hash functions by generating a random H matrix and multiplying it by the binary representation of the element and getting the index of the element.

- Hashing class as it contains the methods
  -method to generate random matrix H(bxU) (`randomH`)
  -method to convert the element to binary representation(`convertToBinary`)
  -method to convert to decimal representation(`convertToDecimal)`
  - method multiply the random h with x (`multiply`)

- In the Nsquare class:

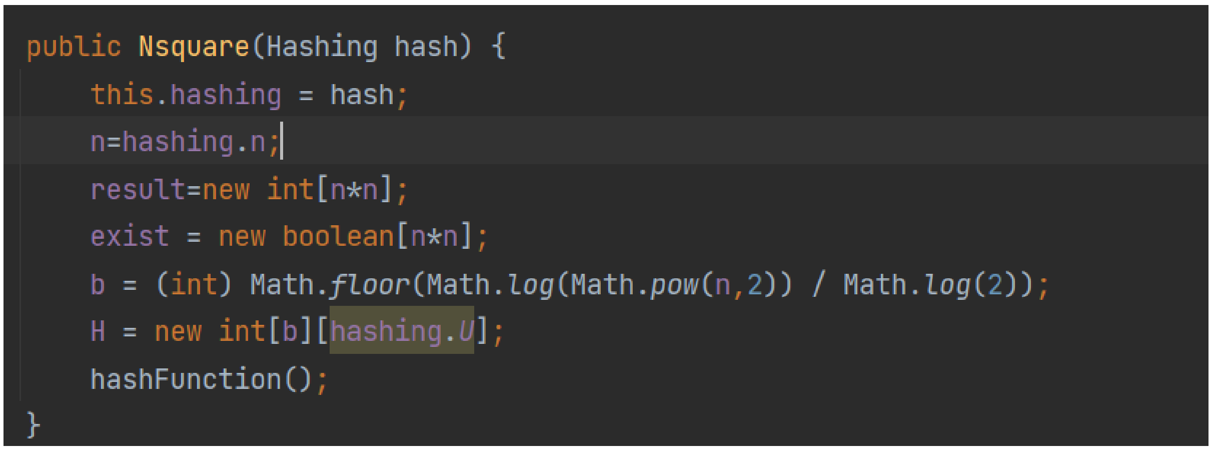
Contains method (hash function()) which loops through the array until it find suitable hash function for all values

-In the “O n” Class:

Contains method (hash function()) which loops through the array, puting the elements which got the same index in an arraylist in an array which is mapped to by that index, so all the elements of the same indes are in the same arraylist. Taking this array of arraylists and sending each non-empty arraylist to the first method (N^2).

3-Verifying that the hash table constructed consume O(n^2)-space in the quadratic space method and O(N)-space in the linear space method:

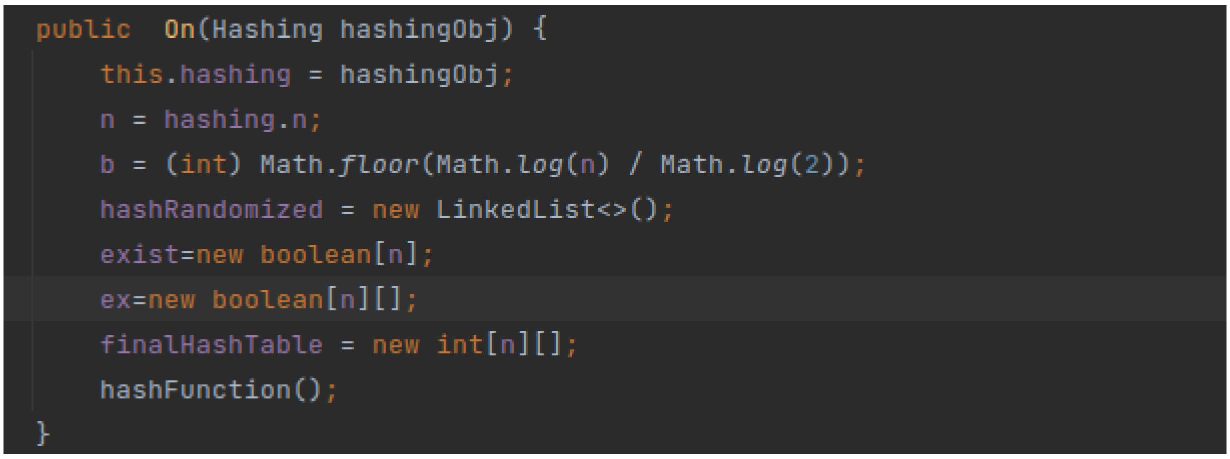
- O(n^2)-space in the quadratic space:



```java
public Nsquare(Hashing hash) {
    this.hashing = hash;
    n=hashing.n;
    result=new int[n*n];
    exist = new boolean[n*n];
    b = (int) Math.floor(Math.log(Math.pow(n,2)) / Math.log(2));
    H = new int[b][hashing.U];
    hashFunction();
}
```

Since The total used space = n^2(for result array) + n^2(for exist array) + 32*2*log(n)(for random matrix H) = O(n^2)

- O(N)-space in the linear space method:



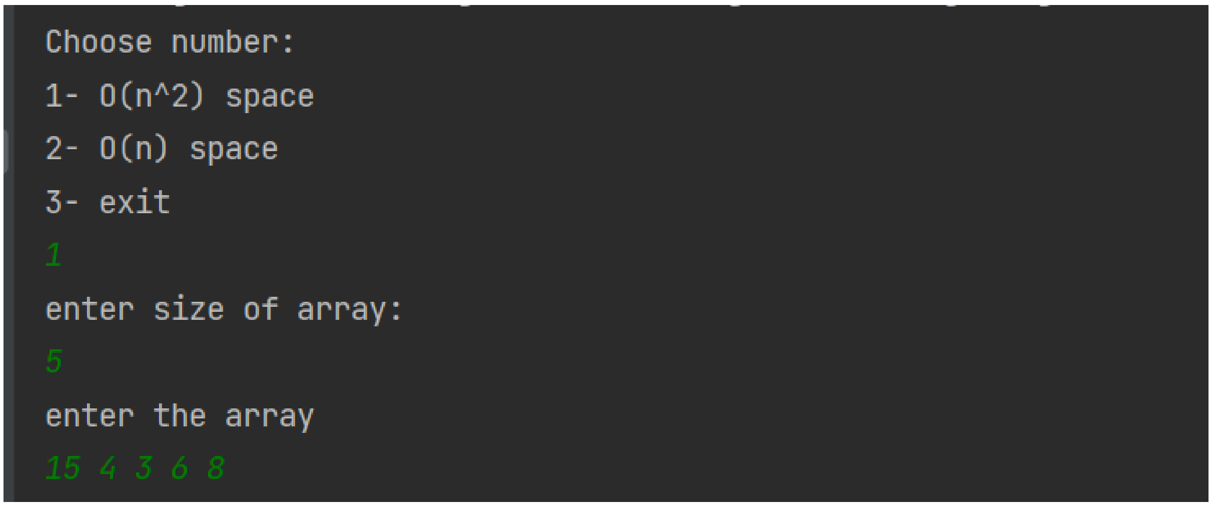
```java
public  On(Hashing hashingObj) {
    this.hashing = hashingObj;
    n = hashing.n;
    b = (int) Math.floor(Math.log(n) / Math.log(2));
    hashRandomized = new LinkedList<>();
    exist=new boolean[n];
    ex=new boolean[n][];
    finalHashTable = new int[n][];
    hashFunction();
}
```

Since The total used space = n(for final hash table array) * m(which is smaller than n)+ n(for ex table array) * m(smaller than n)+n(for exist array)+ 32*2*log(n)(for random matrix H) = O(n).

—> For level-1 hash table T,choose m=n,and let ni be random variable for number of keys that hash to slot I in T. By using ni 2 slots for the level-2 hash table Si .the expected total storage required for two-level scheme is therefore
E[ΣΘ(ni^2)]=Θ(n)

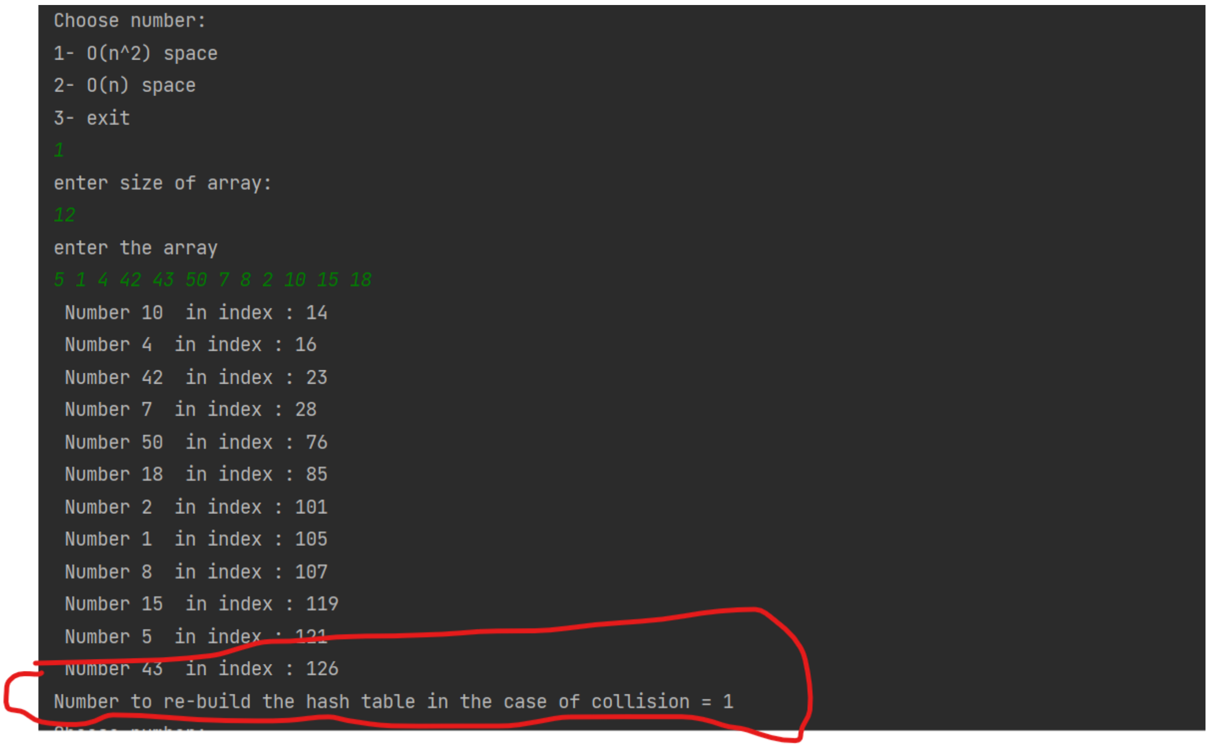
User Guide:

-The user choose one of the methods
Number 1 for O(n^2) and number 2 for O(n)
-Then user enter the size of array
-Then the elements of the array
Number 3 for exit



```
Choose number:
1- O(n^2) space
2- O(n) space
3- exit
1
enter size of array:
5
enter the array
15 4 3 6 8
```

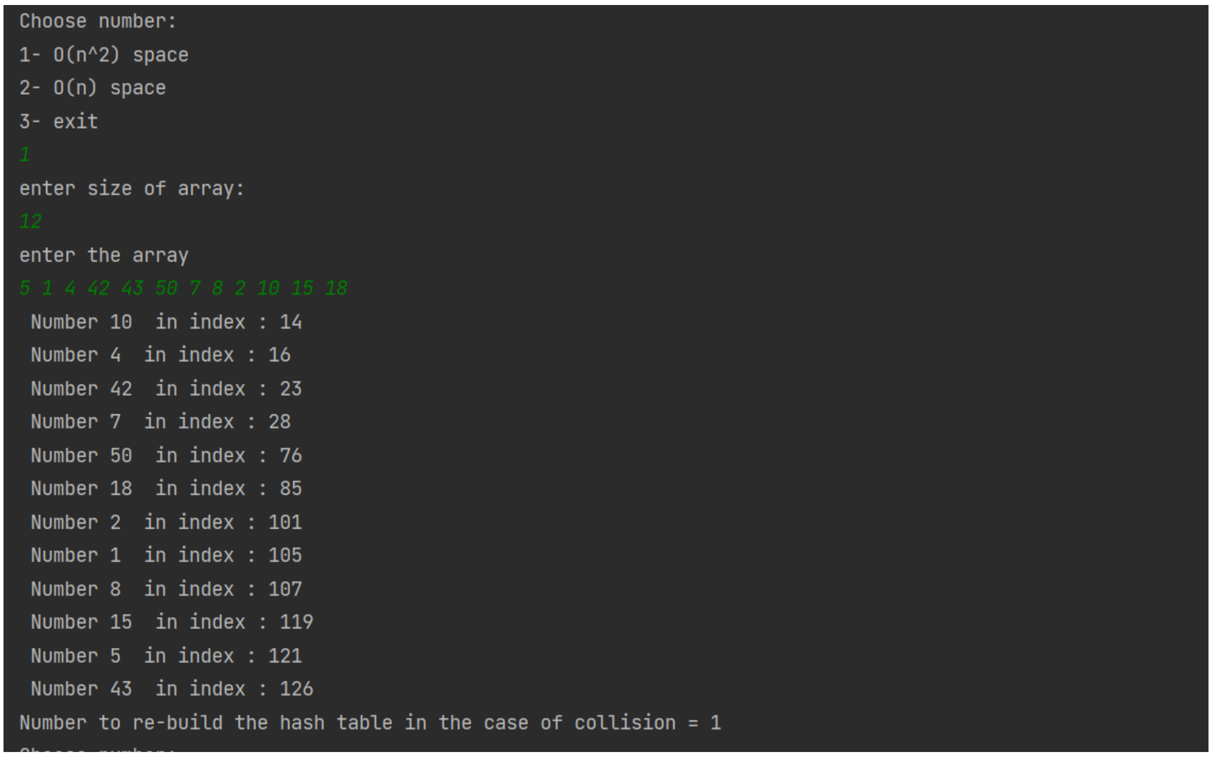
In the O(n^2) we print each element and its location and also number of rebuilds of the hashtable when collision occurs.



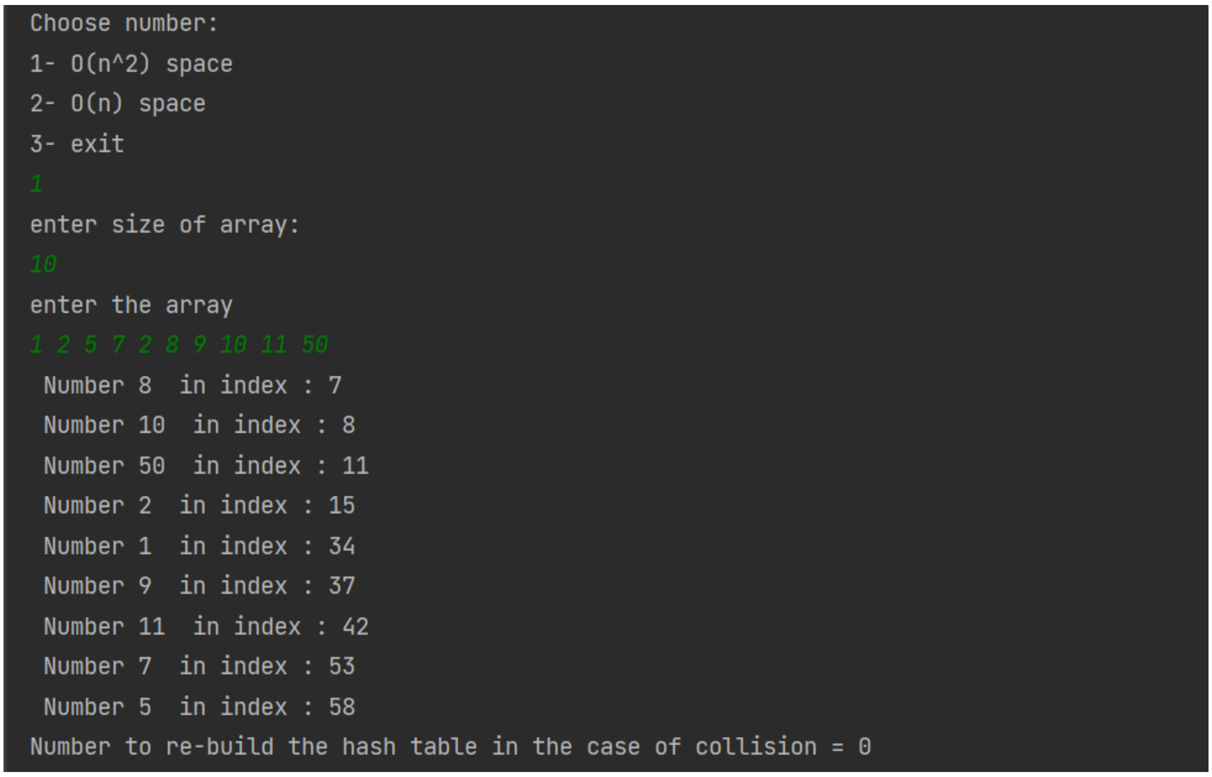
```
Choose number:
1- O(n^2) space
2- O(n) space
3- exit
1
enter size of array:
12
enter the array
5 1 4 42 43 50 7 8 2 10 15 18
 Number 10  in index : 14
 Number 4  in index : 16
 Number 42  in index : 23
 Number 7  in index : 28
 Number 50  in index : 76
 Number 18  in index : 85
 Number 2  in index : 101
 Number 1  in index : 105
 Number 8  in index : 107
 Number 15  in index : 119
 Number 5  in index : 121
 Number 43  in index : 126
Number to re-build the hash table in the case of collision = 1
```

Test Cases:

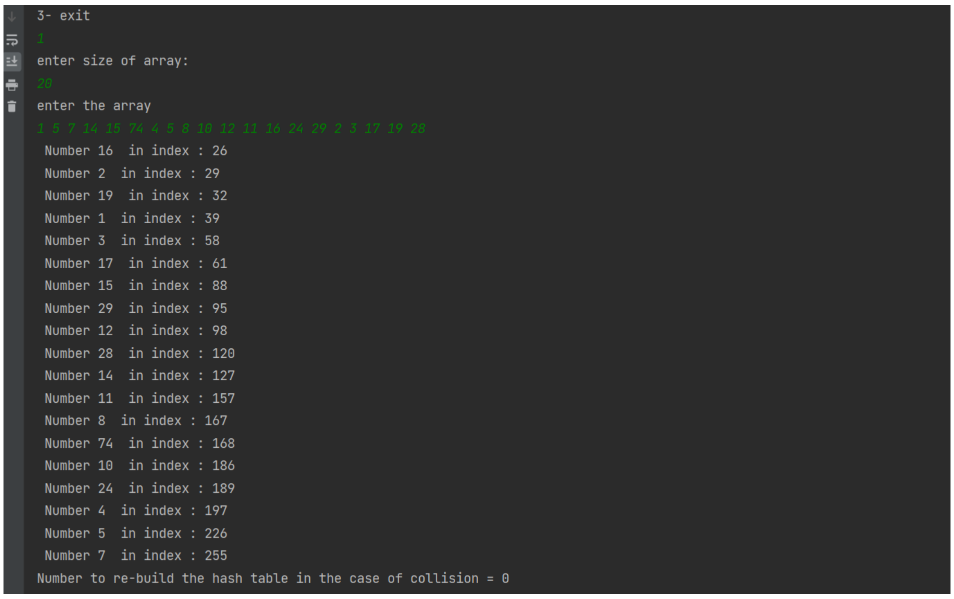
-O(n^2) tests:



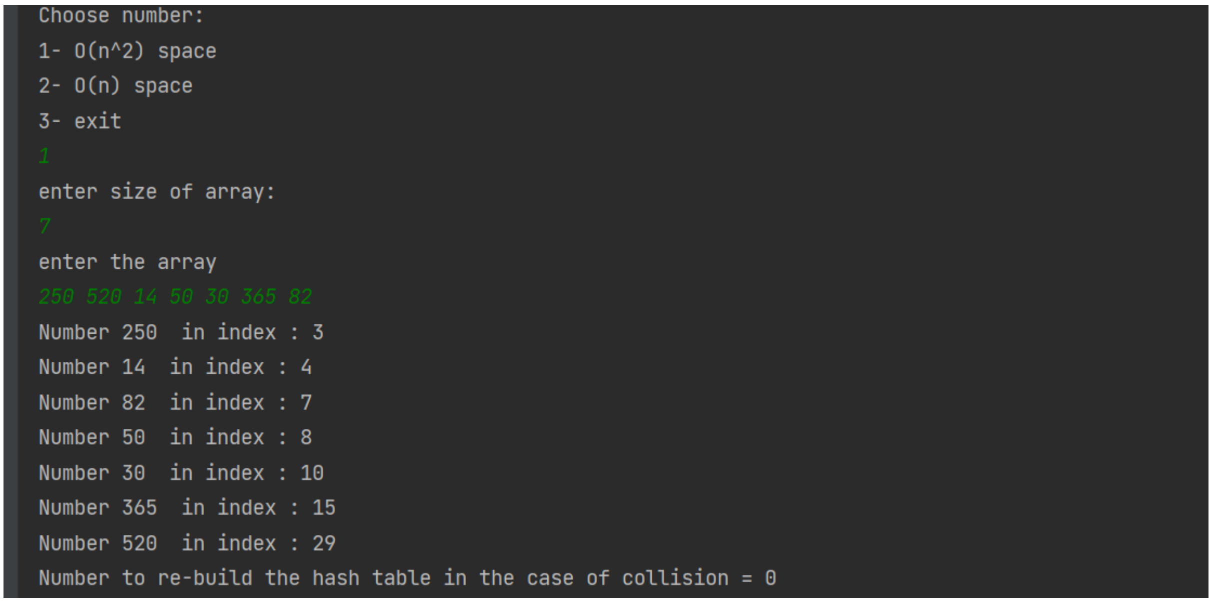
```
Choose number:
1- O(n^2) space
2- O(n) space
3- exit
1
enter size of array:
12
enter the array
5 1 4 42 43 50 7 8 2 10 15 18
 Number 10  in index : 14
 Number 4  in index : 16
 Number 42  in index : 23
 Number 7  in index : 28
 Number 50  in index : 76
 Number 18  in index : 85
 Number 2  in index : 101
 Number 1  in index : 105
 Number 8  in index : 107
 Number 15  in index : 119
 Number 5  in index : 121
 Number 43  in index : 126
Number to re-build the hash table in the case of collision = 1
```



```
Choose number:
1- O(n^2) space
2- O(n) space
3- exit
1
enter size of array:
10
enter the array
1 2 5 7 2 8 9 10 11 50
 Number 8  in index : 7
 Number 10  in index : 8
 Number 50  in index : 11
 Number 2  in index : 15
 Number 1  in index : 34
 Number 9  in index : 37
 Number 11  in index : 42
 Number 7  in index : 53
 Number 5  in index : 58
Number to re-build the hash table in the case of collision = 0
```

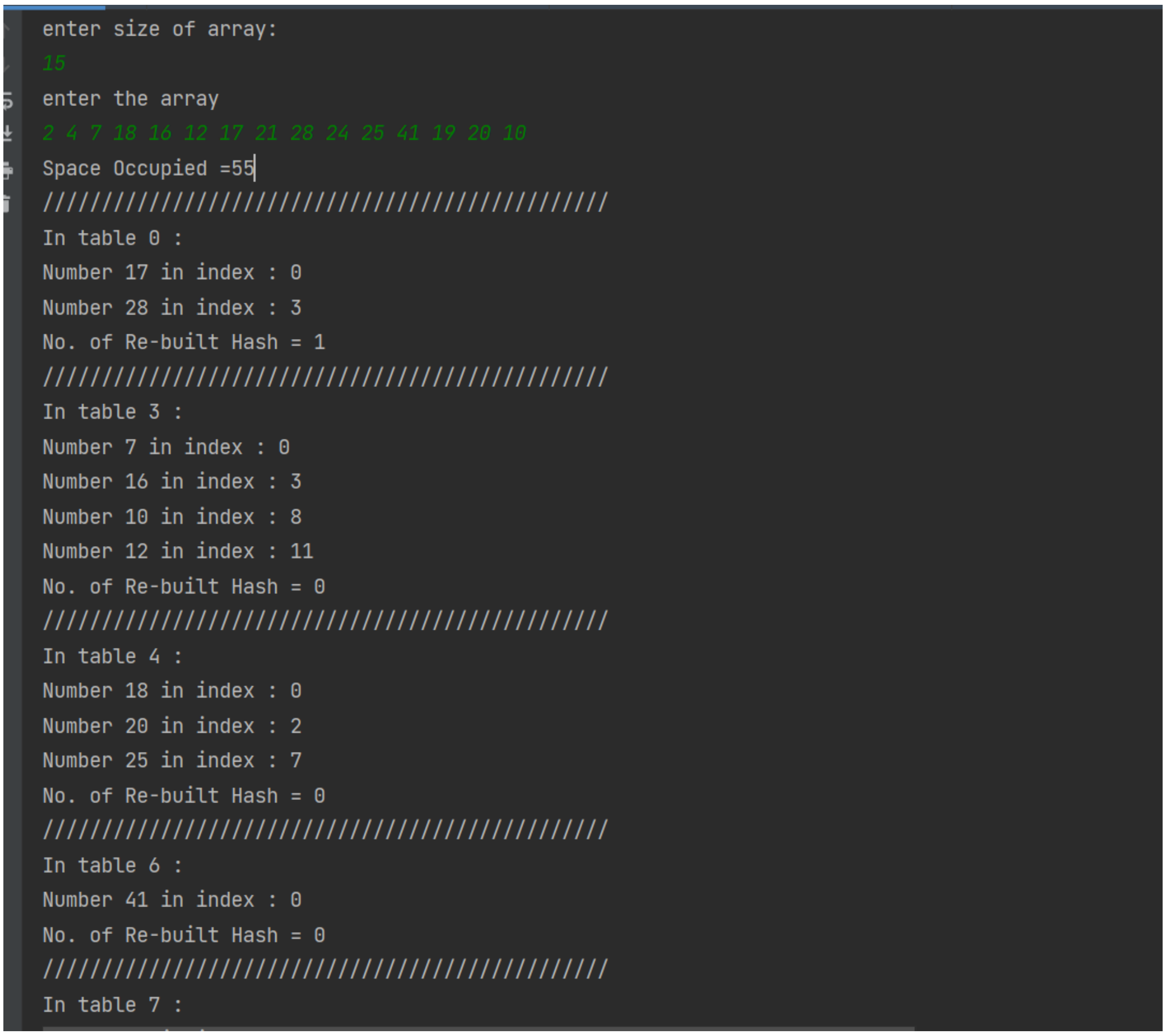


```
3- exit
1
enter size of array:
20
enter the array
1 5 7 14 15 74 4 5 8 10 12 11 16 24 29 2 3 17 19 28
 Number 16  in index : 26
 Number 2  in index : 29
 Number 19  in index : 32
 Number 1  in index : 39
 Number 3  in index : 58
 Number 17  in index : 61
 Number 15  in index : 88
 Number 29  in index : 95
 Number 12  in index : 98
 Number 28  in index : 120
 Number 14  in index : 127
 Number 11  in index : 157
 Number 8  in index : 167
 Number 74  in index : 168
 Number 10  in index : 186
 Number 24  in index : 189
 Number 4  in index : 197
 Number 5  in index : 226
 Number 7  in index : 255
Number to re-build the hash table in the case of collision = 0
```

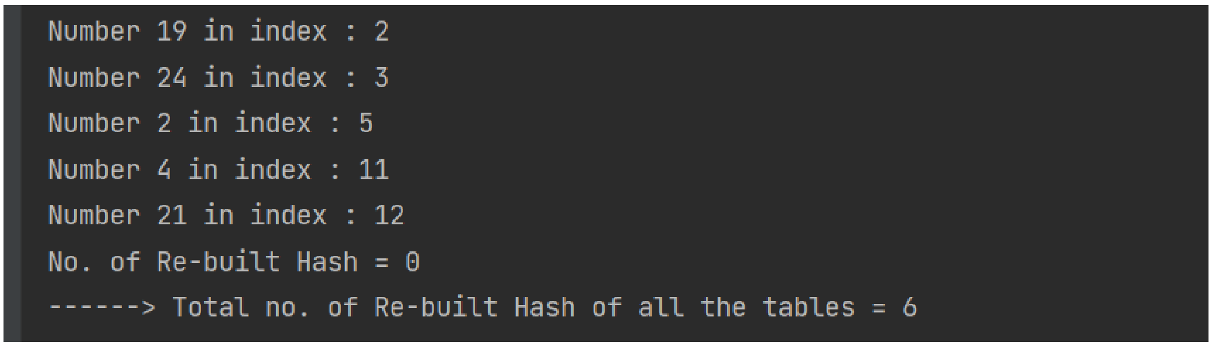


```
Choose number:
1- O(n^2) space
2- O(n) space
3- exit
1
enter size of array:
7
enter the array
250 520 14 50 30 365 82
Number 250  in index : 3
Number 14  in index : 4
Number 82  in index : 7
Number 50  in index : 8
Number 30  in index : 10
Number 365  in index : 15
Number 520  in index : 29
Number to re-build the hash table in the case of collision = 0
```

-O(n) tests:



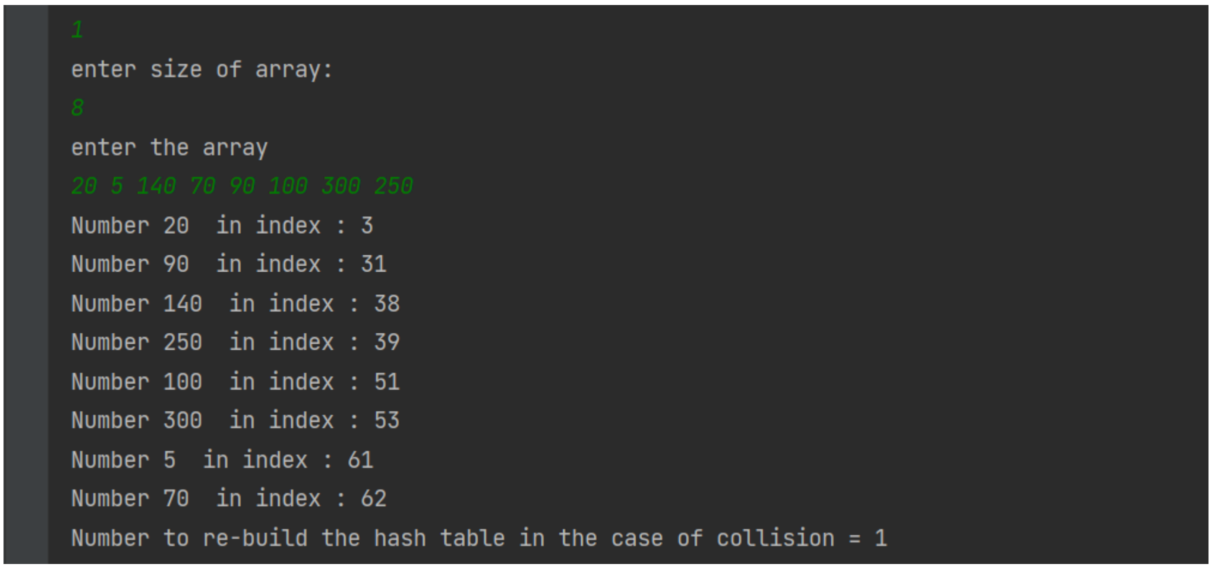
```
enter size of array:
15
enter the array
2 4 7 18 16 12 17 21 28 24 25 41 19 20 10
Space Occupied =55
////////////////////////////////////////////////
In table 0 :
Number 17 in index : 0
Number 28 in index : 3
No. of Re-built Hash = 1
////////////////////////////////////////////////
In table 3 :
Number 7 in index : 0
Number 16 in index : 3
Number 10 in index : 8
Number 12 in index : 11
No. of Re-built Hash = 0
////////////////////////////////////////////////
In table 4 :
Number 18 in index : 0
Number 20 in index : 2
Number 25 in index : 7
No. of Re-built Hash = 0
////////////////////////////////////////////////
In table 6 :
Number 41 in index : 0
No. of Re-built Hash = 0
////////////////////////////////////////////////
In table 7 :
```



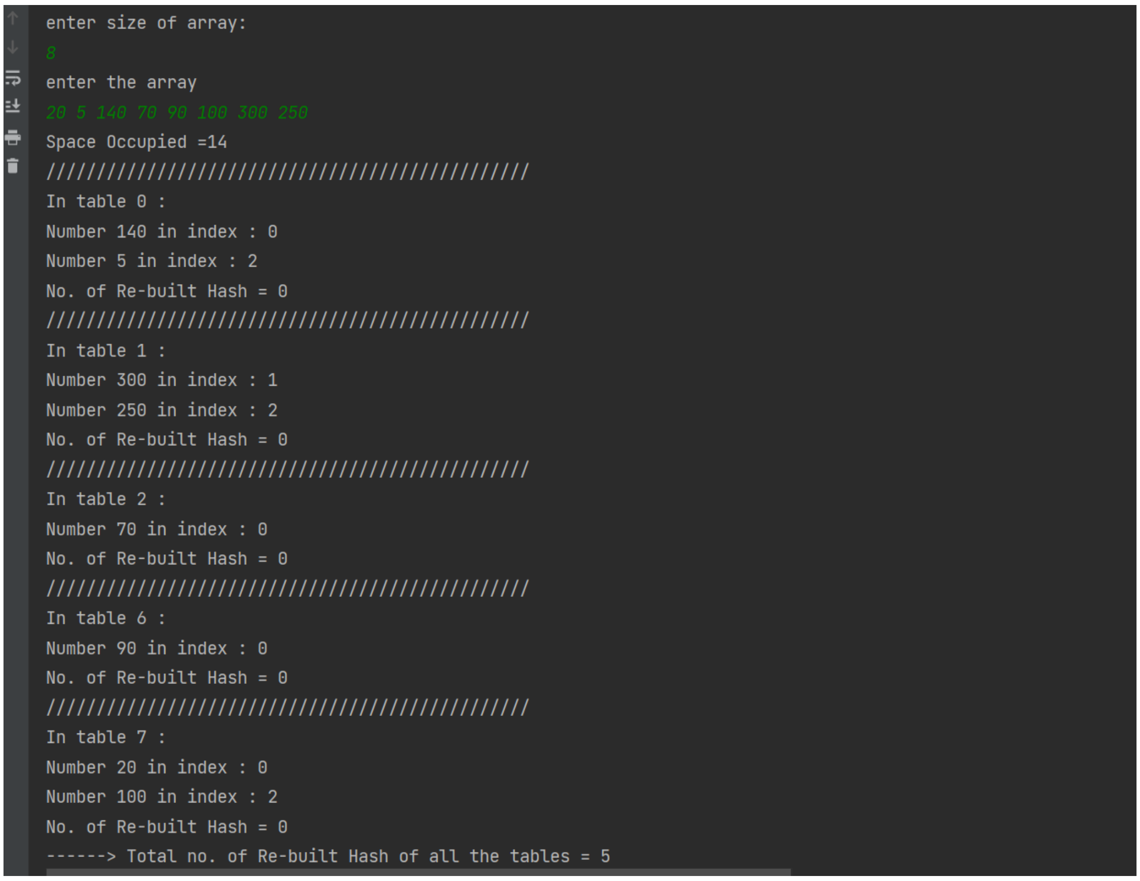
```
Number 19 in index : 2
Number 24 in index : 3
Number 2 in index : 5
Number 4 in index : 11
Number 21 in index : 12
No. of Re-built Hash = 0
------> Total no. of Re-built Hash of all the tables = 6
```

To report no. of collision in both methods and space occupied:

Input array =[20, 5, 140, 70 ,90, 100, 300 ,250]
Total space in first method is n^2 = 64
The number of rebuild hash functions in first method = 1
Total space in second method = 14 <64 n^2 The number of rebuild hash functions in second method = 0,0,0,0,0 so the total is 5

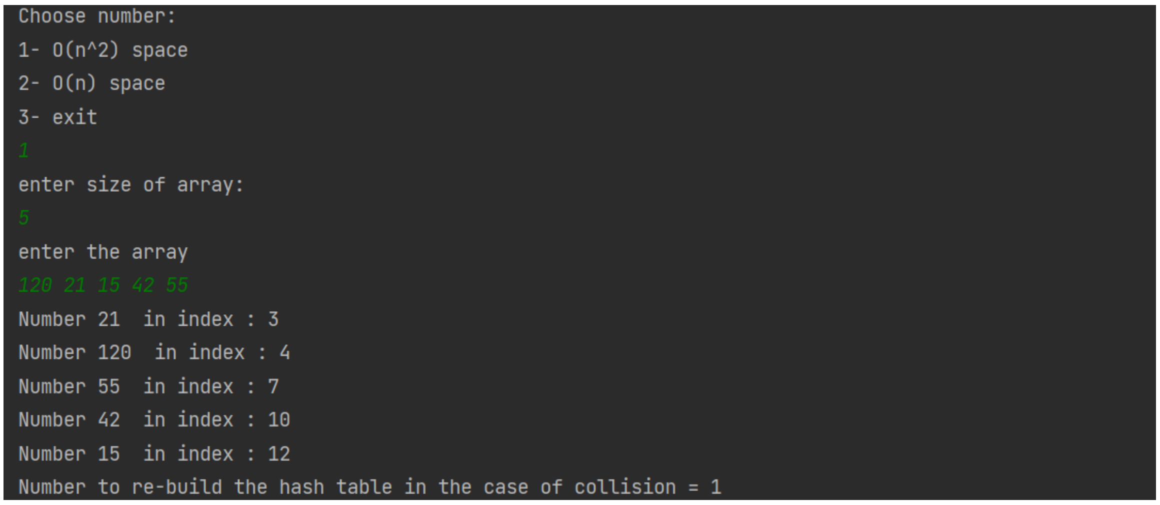


```
1
enter size of array:
8
enter the array
20 5 140 70 90 100 300 250
Number 20  in index : 3
Number 90  in index : 31
Number 140  in index : 38
Number 250  in index : 39
Number 100  in index : 51
Number 300  in index : 53
Number 5  in index : 61
Number 70  in index : 62
Number to re-build the hash table in the case of collision = 1
```

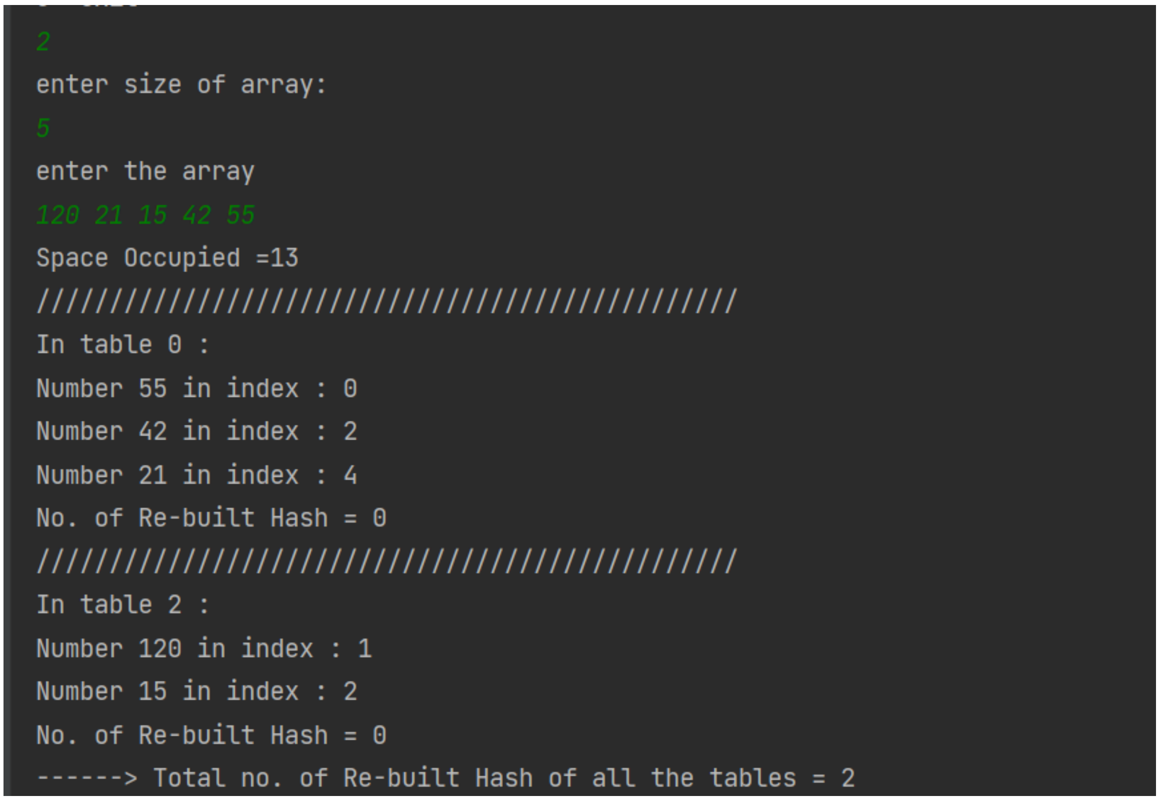


```
enter size of array:
8
enter the array
20 5 140 70 90 100 300 250
Space Occupied =14
/////////////////////////////////////////////////
In table 0 :
Number 140 in index : 0
Number 5 in index : 2
No. of Re-built Hash = 0
/////////////////////////////////////////////////
In table 1 :
Number 300 in index : 1
Number 250 in index : 2
No. of Re-built Hash = 0
/////////////////////////////////////////////////
In table 2 :
Number 70 in index : 0
No. of Re-built Hash = 0
/////////////////////////////////////////////////
In table 6 :
Number 90 in index : 0
No. of Re-built Hash = 0
/////////////////////////////////////////////////
In table 7 :
Number 20 in index : 0
Number 100 in index : 2
No. of Re-built Hash = 0
------> Total no. of Re-built Hash of all the tables = 5
```

Input array =[120, 21, 15, 42, 55]
Total space in first method is n^2 = 25
The number of rebuild hash functions in first method = 1
Total space in second method = 13 <25(n^2)
The number of rebuild hash functions in second method = 0,0 so the total is 2



```
Choose number:
1- O(n^2) space
2- O(n) space
3- exit
1
enter size of array:
5
enter the array
120 21 15 42 55
Number 21  in index : 3
Number 120  in index : 4
Number 55  in index : 7
Number 42  in index : 10
Number 15  in index : 12
Number to re-build the hash table in the case of collision = 1
```



```
2
enter size of array:
5
enter the array
120 21 15 42 55
Space Occupied =13
//////////////////////////////////////////////////
In table 0 :
Number 55 in index : 0
Number 42 in index : 2
Number 21 in index : 4
No. of Re-built Hash = 0
//////////////////////////////////////////////////
In table 2 :
Number 120 in index : 1
Number 15 in index : 2
No. of Re-built Hash = 0
------> Total no. of Re-built Hash of all the tables = 2
```

Assumptions:

- In the O(n^2) number of collision is on average 2 collisions
- if there is duplication in the array we remove the duplicate values .

....