is easy since the element with highest priority will always be in the beginning of the list.

Hashing

Hashing is one of the searching techniques that ufes a constant time.
The time complexity of Hashing is Big O(1)
The pervious searching techniques is :-
Linear Search & Binary Search.
In these searching techniques, the search -ing depends upon the number of eleme -nts.
The searching techniaue which provides a constant time is Hashing.

In Hashing, the Hash Table & Hash Func -tion are used. Using the Hash Functi -on, we can calculate the address at which the value can be stored
The main idea behind the Hashing is to create the (Key / value) pairs.
If the Key is given then the algorith -m computes the Index at which the value would be Stored.
we can write it as:-

hash(Key) = index.

There are 3 ways of calculating the hash function:-

i) Division Method
ii) Folding Method
iii) Mid Square Method

* Division Method :- In division method, the hash function can be defined as :-

$h(K) = K \% n$;, where n is the size of hash table.

For example, if the key value is 6 & the size of the hash table is 10. If the hash function is applied to the key (6) then index would be :-

$h(6) = 6 \% 10 \Rightarrow 6$

The index is 6, at which the value is stored.

If the key is 74, then

$h(74) = 74 \% 10 \Rightarrow 4.$

The index is 4, at which 74 is stored.

* Mid Square Method = In Mid Square Method, the key is squared & some digits or bits from the middle of the square are taken as address. It is important that the same digits would be seleted from the squares of all the keys.
Suppose, our keys are 4 digits integers & the size of table is 1000. we will need 3 digit address. we can square the keys & take the 3rd, 4th & 5th digits from each squared number as the hash address for the corresponding keys.

For example:- Key = 1337
Square of Key = 1787569
Address is = 875.

Key = 1026
Square of Key = 1052676
Address is = 526.

* Folding Method = There are 2 folding methods are used:-
i) Fold Shift method.
ii) Fold Boundary method.

① Fold Shift : In this, the key value is divided into parts whose size matches the size of required address & added.

Key : 123456789

we divide the key into parts, i.e.,

123 | 456 | 789

Now, we add the parts individually,

```
  123
  456
+ 789
------
 1368
```

} this is our some, & now we neglect the left most carry part,

so, our index is 368.

② Fold Boundary : In this, the key value is divided in parts, whose size matches the size of required address & then the boundary keys are reversed & then added.

For example, Key : 123456789

we divide the key into parts, i.e.,

123 | 456 | 789

The boundary keys are reversed, i.e., 123 & 789 are reversed & then added :-

=> 321 + 456 + 987
=> 1764 → this is our some & we neglect the left most carry part.

so, in fold boundary method, our index address is 764.

—x——x——x——x——x——x——

when two different keys have the same address / index value, then the problem occurs between this is collision.

Let suppose, we use the Division Method for Hashing, and the keys are 26 & 76.

so,

h(26) = 26 % 10 => 6th index value
h(76) = 76 % 10 => 6th index value

Here, both the keys have the same index value. Therefore, two values are stored at the same index, i.e., 6 & this leads to the collision problem. To solve this, we have some collision techniques :-

i) Open Hashing
ii) closed Hashing.

* **Open Hashing** : In open hashing we have the chaining method to resolve the collision.

For example,

24, 19, 32, 44 } this is our Keys.

Hashing function is,

K mod 6.

① 24 mod 6 is 0th index

② 19 mod 6 is 1st index.

③ 32 mod 6 is 2nd index.

④ 44 mod 6 is 2nd index.

index	Keys
0	24
1	19
2	32
3	
4	
5	

Now, the collision occurs because 32, & 44 have the same index value. Here, what we do is, we link the 44 node address to 32 node's links part through linked list.

index	Keys
0	24
1	19
2	32 → [44]
3	
4	
5	

* Closed Hashing = In closed hashing, we have 3 techniques:-

i) Linear Probing
ii) Quadratic Probing
iii) Double Hashing Technique.

(1) Linear Probing = As the each cell in the hash table contains a key-value pair, so when the collision occurs by mapping a new key to the cell already occupied by another key, then linear probing technique search for the closest free locations & adds a new key to that empty/free cell.

(2) Quadratic Probing = It uses quadratic polynomial for searching until an empty slot is found.

$\{(h(k) + i^2) \% 10$

→ Quadratic Probing

$\{(h(k) + i) \% 10$

→ linear probing

* Infix to Postfix

K+L-M*N+(O^P)*W/U/V*T+Q

Input	Stack	Postfix
K		K
+	+	K
L	+	KL
-	-	KL+
M	-	KL+M
*	-*	KL+M
N	-*	KL+MN
+	+	KL+MN*-
(	+(	KL+MN*-
O	+(	KL+MN*-O
^	+(^	KL+MN*-O
P	+(^	KL+MN*-OP
)	+	KL+MN*-OP^
*	+*	KL+MN*-OP^
W	+*	KL+MN*-OP^W
/	+/	KL+MN*-OP^W*
U	+/	KL+MN*-OP^W*U
/	+/	KL+MN*-OP^W*U/
V	+/	KL+MN*-OP^W*U/V
*	+*	KL+MN*-OP^W*U/V/
T	+*	KL+MN*-OP^W*U/V/T
+	+	KL+MN*-OP^W*U/V/T*+
Q	+	KL+MN*-OP^W*U/V/T*+Q
		KL+MN*-OP^W*U/V/T*+Q+

* Take values of matrix from user & find the sum of each row & column individually.

```
#include<stdio.h>
int main()
{
   int arr[3][3], i, j, sumOfROW,
                          sumOfCol;
   // taking input of matrix
   for(i=0; i<3; i++)
   {
      for(j=0; j<3; j++)
      {
         printf("\n Enter arr[%d][%d] item: ",
                                  i, j);
         scanf("%d", &arr[i][j]);
      }
   }
   // printing matrix
   printf("\n Your entered matrix are: ");
   for(i=0; i<3; i++)
   {
      for(j=0; j<3; j++)
      {
         printf("%d ", arr[i][j]);
      }
      printf("\n");
   }

   // calculating sum of rows & columns
   for(i=0, i<3; i++)
   {
      sumOfROW=0, sumOfCol=0;
```

```
    for(j=0; j<3; j++)
    {
        sumOfRow = sumOfRow +
                        (arr[i][j]);
        sumOfCol = sumOfCol +
                        (arr[j][i]);
    }
    printf("Sum of Row %d are: %d",
                        i, sumOfRow);
    printf("Sum of column %d are: %d",
                        i, sumOfCol);
}
printf("\n");
return 0;
}
```