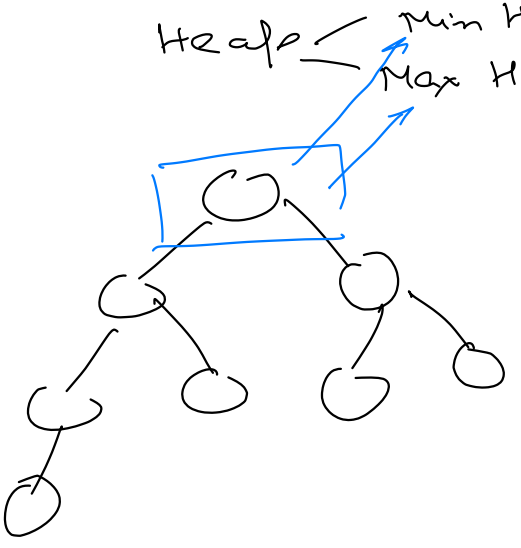


Heap Property

Parent data > MAX (child's data)

Largest Value always at top

Hash Table

→ Store data < Linear / Hierarchy

→ Search & Sort

O(n) O(log n)

Problem: Stream of numbers between 1 to N.
Stream ends when we get a 0.
When we get a number from stream, tell if it is duplicate.

Ex: 5 9 1 8 5 3 2 7 2 8 1 ...
Dup Dup Dup Dup

BST → insert O(log n)
→ search O(log n)

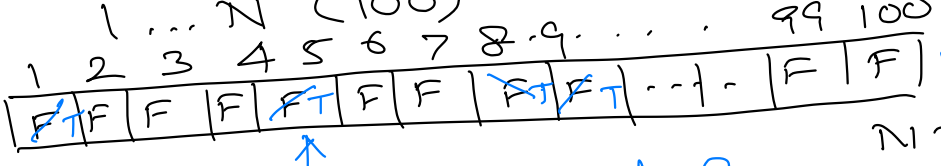
Linked → insert O(1)
List → search O(n)

BitMap → insert O(1)
~~XXX~~ search O(1)
Direct Address Table

T } 1 bit
F

integer → 4 bytes
4 × 8 bits
= 32 bits

1 ... N (100)



map [elem]
1 byte = 8 bits

N1 = integer
2^{31}

$\frac{4GB}{8} = 2^{31}$ bytes
$= 2$ 4GB

N = 100

```
bool mcp[100];
```

1 byte

100 bytes

1 ... 100

⇓

0 ... 99

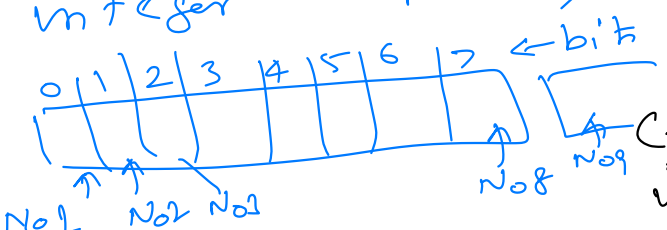
T/F 2 options ⇒ 1 bit

100 values ⇒ 100 bit

⇓

100/8 bytes.

integer mcp[4];



$\frac{100}{32}$ (4 × 8 bit) ≈ integer

bytes = 3.125 bytes

≈ 4 byt

Hash Table

↳ is a group of BUCKETS

↓

that store elements

Hash Function : Maps a key to a unique bucket.

What we store in hash table ⇒ KEY

```
int hashTable[10];
int hashFn(int key){
    return key % 10;
}
```

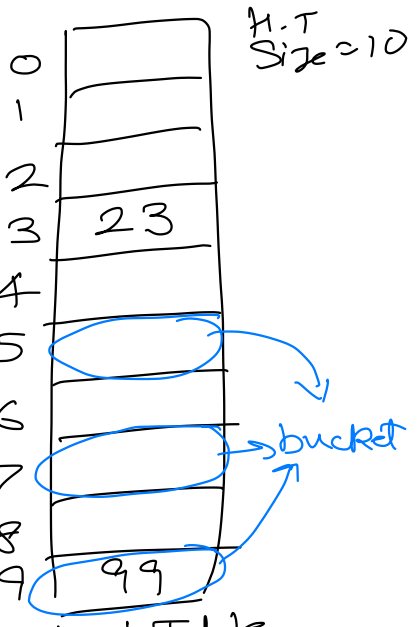
// MOD N

Size of Hash Table

0..9

Time ⇒ O(1)

H.T Size = 10



Add (elem)

1. bucketId = hashFn(elem)
2. hashTable[bucketId] = elem

23 → HF → 3

99 → HF → 9

Add (23)
Add (99)

Bucket Size = 1

Size of Hash Table = Number of Buckets in H.T.

Search(elem)

1. bucketId = hashFn(elem)
2. if (hashTable[bucketId] == elem)
   return true
3. return false.

Time = O(1)

Add(13)

13 → HF → 3

Bucket 3 already stores a elem (23) and it is not 13.

COLLISION

When hash function maps more than one key to a bucket id.

How to solve collision

→ Use a better hash function.

→ Handle collision

23 → HF → 3

13 → HF → 3

MOD N

Hash Function

1. MOD N

Key = 93367

N = 100

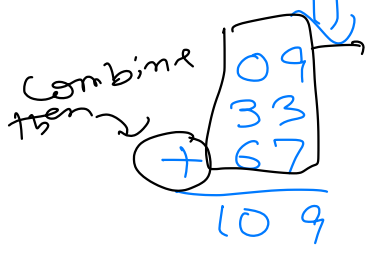
bucket id = 93067 MOD 100 = 67

9 6 7 3 3 MOD 33

5 7 6 7 MOD 100 = 67

2. Folding ⇒ Divide key into multiple parts & combine (fold) them together.

Key = 93367



96723

MOD N = 9

5767

57 + 67 = 124

MOD N = 24

3. Mid Square : Square the key & Pick digits from middle of square.

$9367 = 87773689$

87773689 bucket id 39

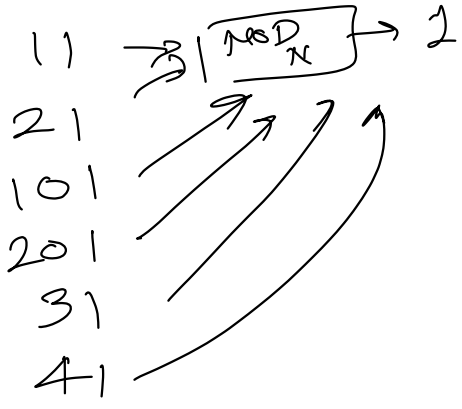
$5767 = 33258289$ bucket id 58

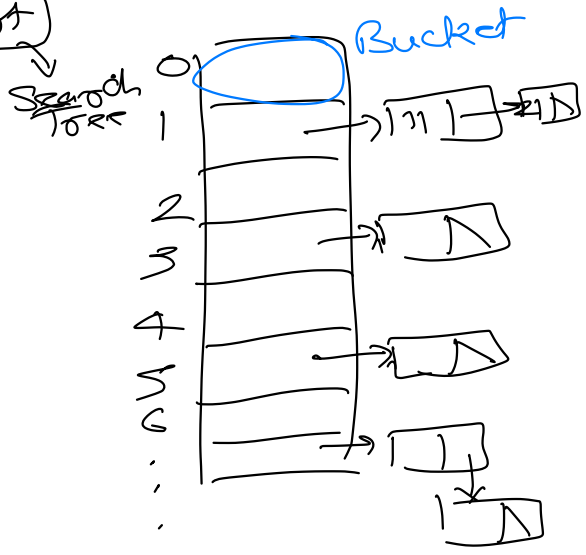
$96733 = 935727 3289$

bucket id = 27

Handling Colision

1. Chaining : Bucket is implemented as a [linked list]





2. Open Addressing

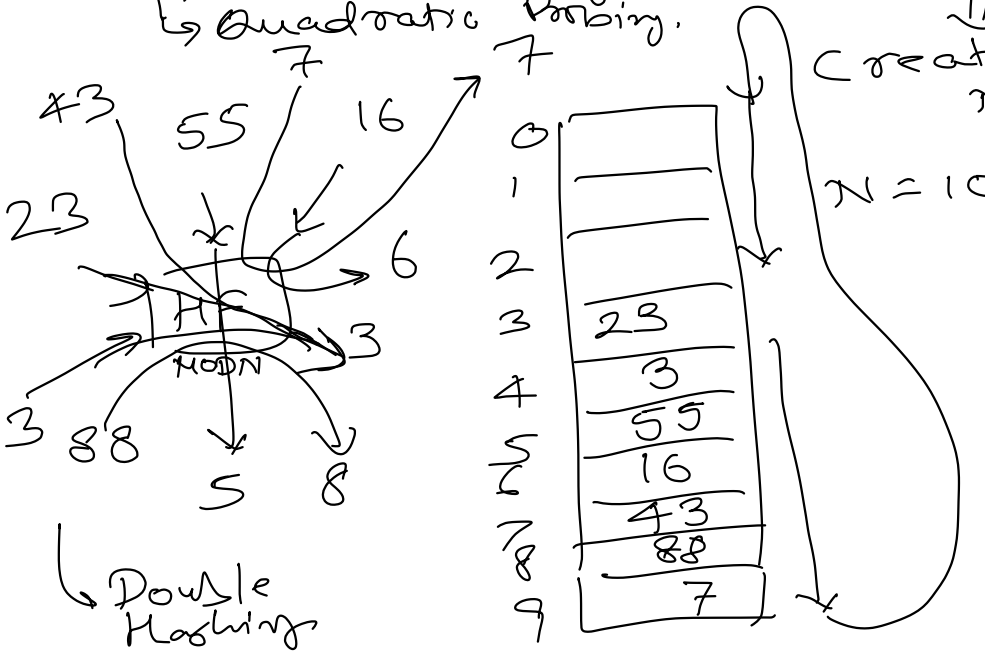
↳ Linear Probing.

↳ Quadratic Probing.

N elements to be stored in H.T.

⇓

Create a H.T. of N Buckets



↳ Double Hashing

quad eq.

n^2+n

Symbol Table

↓

H.T. of Keywords.

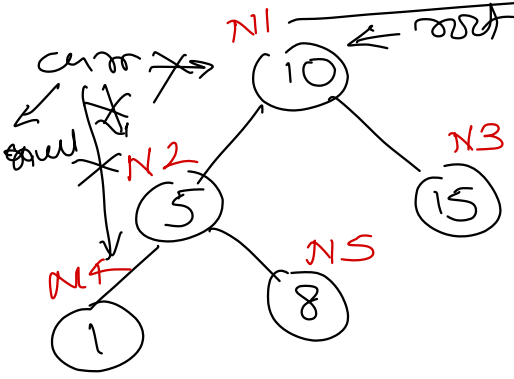
Chaining => we try to keep bucket size as small as possible.

Bucket size = R

$R << n$ → size of H.T.

$O(\log R)$

Inorder Traversal

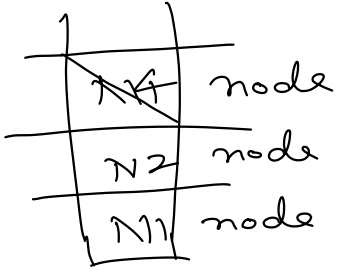



```
inorder (node)
1. if (node is null)
       STOP
2. inorder (node.left)
3. Process node
4. inorder (node.right)
```

curr → N4

~~N4~~	~~null~~
~~N2~~	~~N5~~
~~N3~~	~~null~~
~~null~~	~~N5~~
~~N3~~	null
null	

Impl Stack



BSTree<int> a = new
BST<int>();