

UNIT- NINE

SEARCHING

Topics to be Covered

- Introduction to searching Techniques
- Essential of Search
- Sequential Search
- Binary Search
- Tree Search
- General Search Tree
- Hashing: hash function and hash table
- Collision resolution techniques
- Efficiency comparision of different Search techniques

Introduction to searching techniques

- The process of finding the location of a specific data item or record with a given key value or finding the location of all records, which satisfy one or more conditions in a list is called Searching.
- If the item exist in the given list then it is said to be successful other wise it is said unsuccessful.

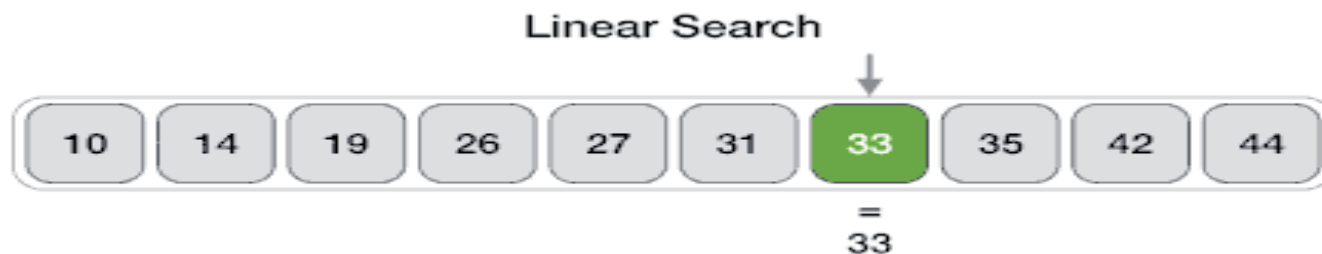
Searching falls into two categories:

External Searching: it means searching the records using keys where there are many records, which reside in files stored on the disks.

Internal Searching: It means searching the records using keys where there are less number of records residing entirely within the computer's main memory.

Linear Search

- Linear search is also called as **sequential search algorithm**. It is the simplest searching algorithm. In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL.
- It is widely used to search an element from the unordered list, i.e., the list in which items are not sorted.





Algorithm:

Linear Search (Array A, Value x)

Step 1: Set i to 1

Step 2: if $i > n$ then go to step 7

Step 3: if $A[i] = x$ then go to step 6

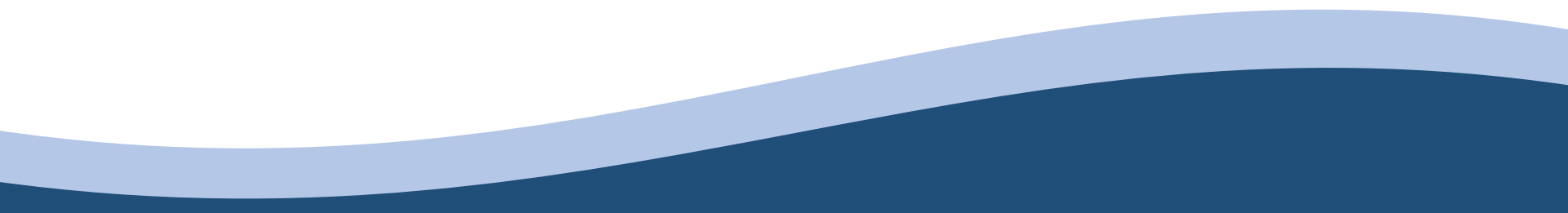
Step 4: Set i to $i + 1$

Step 5: Go to Step 2

Step 6: Print Element x Found at index i and go to step 8

Step 7: Print element not found

Step 8: Exit



Time Complexity:

Case	Time Complexity
Best Case	$O(1)$
Average Case	$O(n/2)$
Worst Case	$O(n)$

Space Complexity:

Space Complexity	$O(1)$
-------------------------	--------

Binary Search

Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to $O(\log n)$.

Binary Search Algorithm:

1. Begin with the mid element of the whole array as a search key.
2. If the value of the search key is equal to the item then return an index of the search key.
3. Or if the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
4. Otherwise, narrow it to the upper half.
5. Repeatedly check from the second point until the value is found or the interval is empty.

1. Iteration Method:

```
binarySearch(arr, x, low, high)
```

```
  repeat till low = high
```

```
    mid = (low + high)/2
```

```
    if (x == arr[mid])
```

```
      return mid
```

```
    else if (x > arr[mid]) // x is on the  
      right side
```

```
      low = mid + 1
```

```
    else // x is on the left side
```

```
      high = mid - 1
```

1. recursive Method:

```
binarySearch(arr, x, low, high)
```

```
  if low > high
```

```
    return False
```

```
  else
```

```
    mid = (low + high) / 2
```

```
    if x == arr[mid]
```

```
      return mid
```

```
    else if x > arr[mid] // x is on the right side
```

```
      return binarySearch(arr, x, mid + 1, high)
```

```
    else // x is on the left side
```

```
      return binarySearch(arr, x, low, mid - 1)
```


Binary Search

Search 23

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91

23 > 16
take 2nd half

L=0	1	2	3	M=4	5	6	7	8	H=9
2	5	8	12	16	23	38	56	72	91

23 < 56
take 1st half

0	1	2	3	4	L=5	6	M=7	8	H=9
2	5	8	12	16	23	38	56	72	91

Found 23,
Return 5

0	1	2	3	4	L=5, M=5	H=6	7	8	9
2	5	8	12	16	23	38	56	72	91

General tree Search

Whenever an element is to be searched, start searching from the root node, then if the data is less than the key value, search for the element in the left subtree. Otherwise, search for the element in the right subtree.

Algorithm:

If root.data is equal to search.data

 return root

else

 while data not found

 If data is greater than node.data

 goto right subtree

 else

 goto left subtree

 If data found

 return node

 endwhile

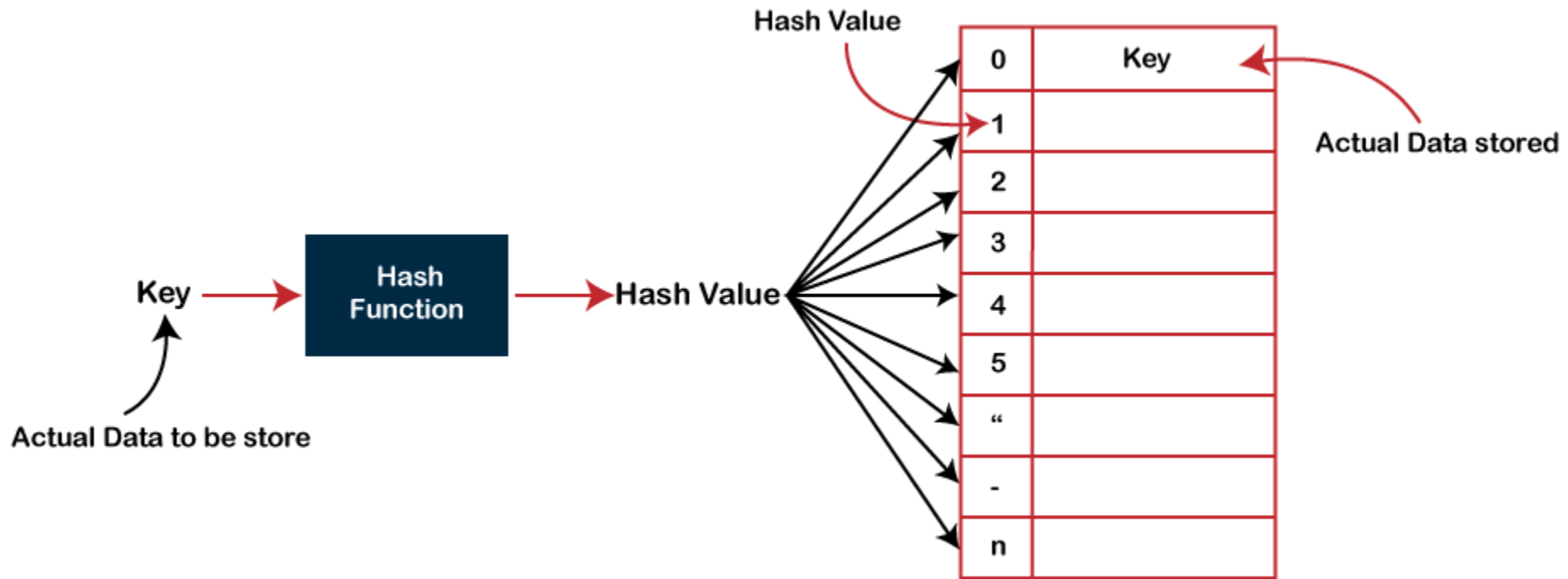
 return data not found

end if

Hashing

- Hashing is one of the searching techniques that uses a constant time. The time complexity in hashing is $O(1)$.
- The worst time complexity in linear search is $O(n)$, and $O(\log n)$ in binary search. In both the searching techniques, the searching depends upon the number of elements but the technique that takes a constant time is hashing technique.
- In Hashing technique, the hash table and hash function are used. Using the hash function, 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 algorithm computes the index at which the value would be stored. It can be written as:

$$\text{Index} = \text{hash}(\text{key})$$



There are three ways of calculating the hash function:

- **Division method**
- Folding method
- Mid square method

Hash Table

- Hash table is one of the most important data structures that uses a special function known as a hash function that maps a given value with a key to access the elements faster.
- A Hash table is a data structure that stores some information, and the information has basically two main components, i.e., key and value. The hash table can be implemented with the help of an associative array. The efficiency of mapping depends upon the efficiency of the hash function used for mapping.

$\text{Hash}(\text{key}) = \text{index};$

Drawback of Hash function

A Hash function assigns each value with a unique key. Sometimes hash table uses an imperfect hash function **that causes a collision** because the hash function generates the same key of two different values.

Hash Functions: Division method

- This is the most simple and easiest method to generate a hash value. The hash function divides the value k by M and then uses the remainder obtained.

Formula:

$$h(K) = k \bmod M$$

Here,

k is the key value, and

M is the size of the hash table.

- It is best suited that M is a prime number as that can make sure the keys are more uniformly distributed. The hash function is dependent upon the remainder of a division. Example:

$$k = 12345$$

$$M = 95$$

$$h(12345) = 12345 \bmod 95 \\ = 90$$

$$K = 546$$

$$M = 95$$

$$h(546) = 546 \bmod 95 \\ = 71$$

Hash value	Key
1	
2	
3	
.	
.	
.	
71	546
72	
.	
.	
90	12345
.	
.	
94	

Fig: Hash Table

Collision and resolution techniques

Collision: When the two different values have the same value, then the problem occurs between the two values, known as a collision.

In the above example, the value 546 is stored at index 71. If the key value is 926, then the index would be:

$$h(926) = 926 \bmod 95 = 71$$

Therefore, two values are stored at the same index, i.e., 71, and this leads to the collision problem. To resolve these collisions, we have some techniques known as collision techniques.

The following are the collision techniques:

Open Hashing: It is also known as closed addressing.

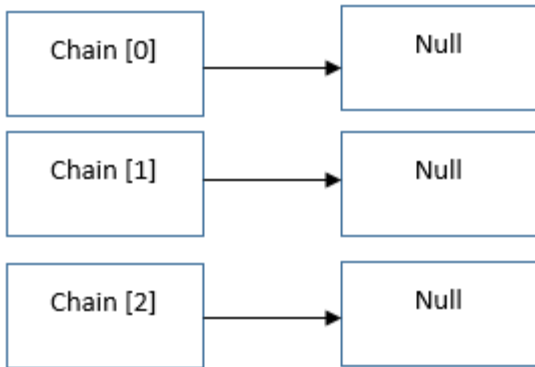
Closed Hashing: It is also known as open addressing.

Open Hashing

- Open hashing is a collision avoidance method which uses array of linked list to resolve the collision.
- It is also known as the **separate chaining method** (each linked list is considered as a chain).
- It is one of the most used techniques by programmers to handle collisions. Basically, a linked list data structure is used to implement the Separate Chaining technique. When a number of elements are hashed into the index of a single slot, then they are inserted into a singly-linked list. This singly-linked list is the linked list which we refer to as a chain in the Open Hashing technique.

Example:

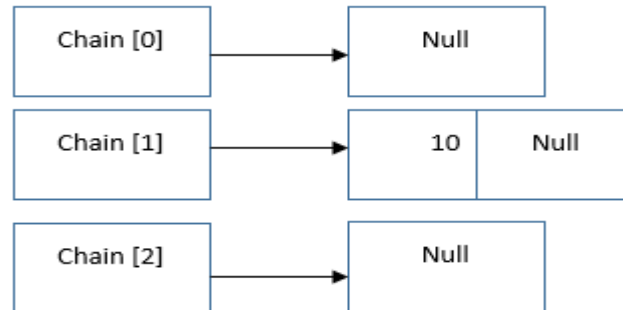
Let us assume a hash table of size 3. Then the array of linked list will be:



Insert key 10:

$$10 \text{ MOD } 3 = 1$$

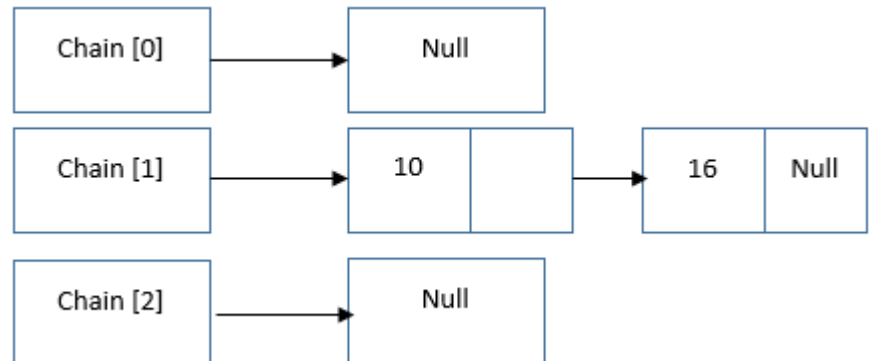
Hence add the node with data 10 in the chain[1].



Insert key 16:

$16 \text{ MOD } 3 = 1$ Collision occurred ,Both 10 and 16 points to the hash index 1.

We can avoid the collision by adding data 16 at the end of the chain[1].



Insert data into the separate chain

Algorithm:

1. Declare an array of a linked list with the hash table size.
2. Initialize an array of a linked list to NULL.
3. Find hash key.
4. If `chain[key] == NULL`
 Make `chain[key]` points to the key node.
5. Otherwise(collision),
 Insert the key node at the end of the `chain[key]`.

Searching a value from the hash table

Algorithm:

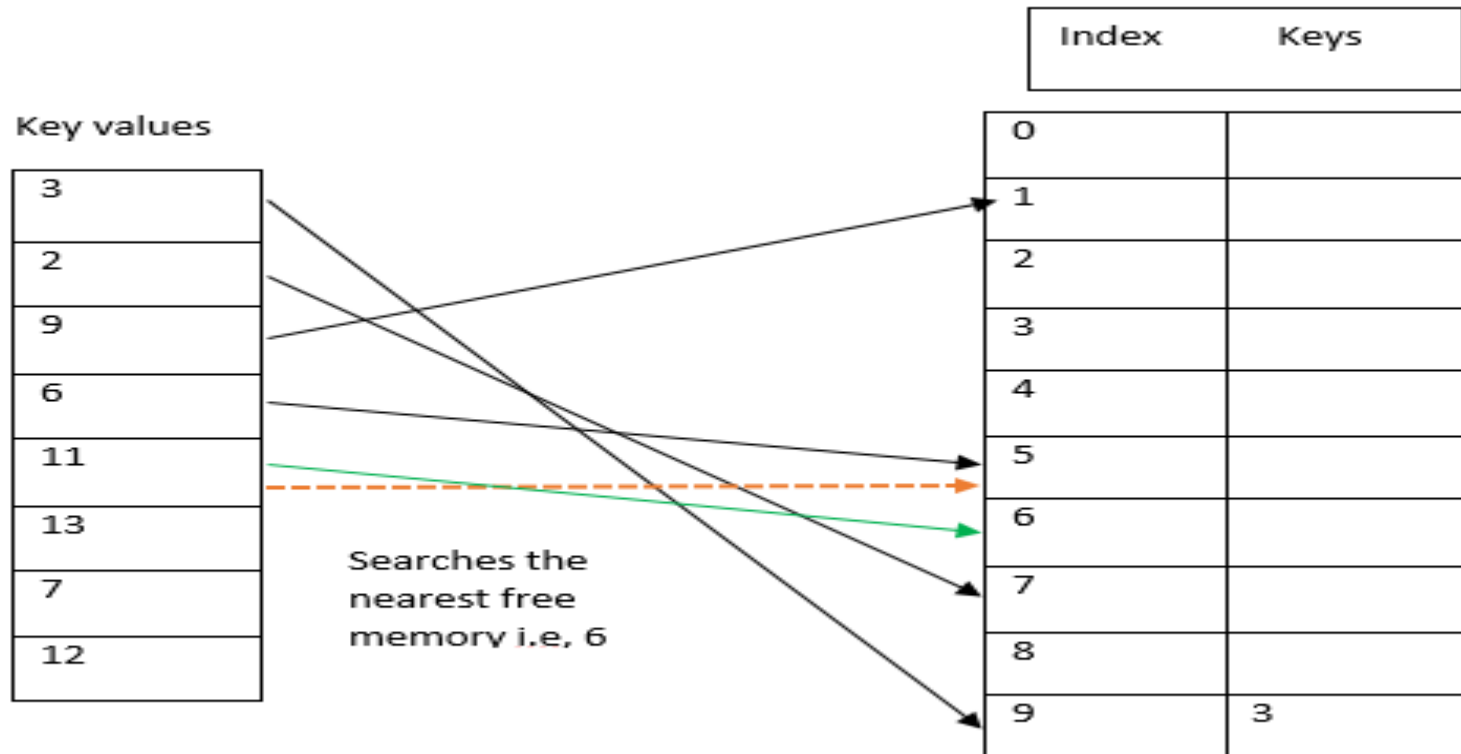
1. Get the value
2. Compute the hash key.
3. Search the value in the entire chain. i.e. `chain[key]`.
4. If found, print "Search Found"
5. Otherwise, print "Search Not Found"

Closed Hashing

In Closed hashing, three techniques are used to resolve the collision:

1. Linear probing
2. Quadratic probing

Linear Probing: Linear probing is one of the forms of closed hashing (open addressing). As we know that 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 searches for the closest free locations and adds a new key to that empty cell**. In this case, searching is performed sequentially, starting from the position where the collision occurs till the empty cell is not found.



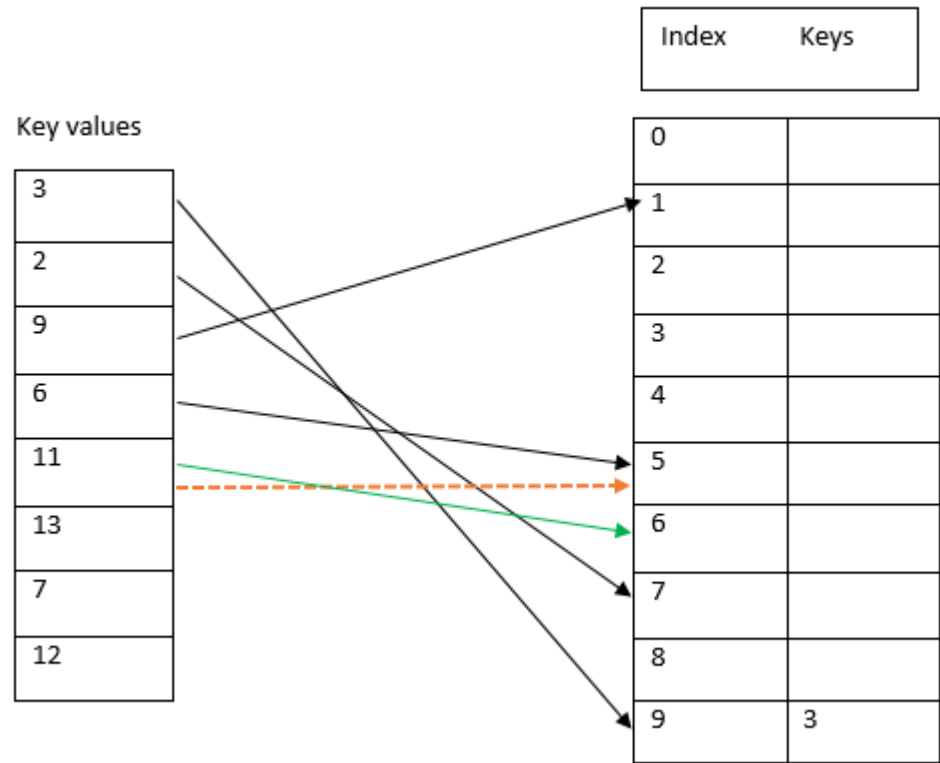
Let the hash function be $2K+3$ and $m=10$

Figure: Collision resolution using linear probing

Quadratic Probing:

quadratic probing is an open addressing technique that uses quadratic polynomial for searching until a empty slot is found.

It can also be defined as that it allows the insertion k_i at first free location from $(u+i^2)\%m$ where $i=0$ to $m-1$.



Let the hash function be $2K+3$ and $m=10$

The index value of 11 is 5, but this location is already occupied by the 6. So, we apply the quadratic probing technique.

When $i = 0$: $\text{Index} = (5+0^2)\%10 = 5$

When $i=1$: $\text{Index} = (5+1^2)\%10 = 6$

Since location 6 is empty, so the value 11 will be added at the index 6.

Comparisons of different Search Algorithms

Sequential Search	Binary Search
Time complexity is $O(n)$	Time complexity is $O(\log n)$
Finds the key present at first position in constant time	Finds the key present at center position in constant time
Sequence of elements in the container does not affect.	The elements must be sorted in the container
Arrays and linked lists can be used to implement this	It cannot be implemented directly into the linked list. We need to change the basic rules of the list to implement this
Algorithm is iterative in nature	Algorithm technique is Divide and Conquer.
Algorithm is easy to implement, and requires less amount of code.	Algorithm is slightly complex. It takes more amount of code to implement.
N number of comparisons are required for worst case.	Log n number of comparisons are sufficient in worst case.

Basis of comparison	Binary Tree	Binary Search Tree
Definition	A nonlinear data structure known as a Binary Tree is one in which each node can have a maximum of two child nodes.	A BST is a binary tree with nodes that has right and left subtrees that are also binary search trees.
Types	<ul style="list-style-type: none"> Complete Binary Tree 	<ul style="list-style-type: none"> AVL Trees
Structure	There is no ordering in a Binary Tree in terms of how the nodes are arranged.	In a BST, the left subtree contains elements that are less than the nodes element, while the right subtree contains elements that are greater than the nodes element.
Duplicate Values	Duplicate values are permitted in binary trees.	Duplicate values are not permitted in the Binary Search Tree.
Speed	Because it is unordered, the speed of deletion, insertion, and searching operations in Binary Tree is slower than in Binary Search Tree.	Because it has ordered properties, the Binary Search Tree performs element deletion, insertion, and searching more quickly.
Complexity	Typically, the time complexity is " $O(n)$ ".	Typically, the time complexity is " $O(\log n)$ ".

Any Query?