Bubble Sort

- Bubble sort is a simple sorting algorithm. It works by comparing each pair of adjacent items and swapping thing them if they are in the wrong order.
- The passes of the list are repeated until no swaps are needed that indicates the list is sorted.
- Bubble sort has time complexity O(n2) which makes it inefficient for sorting large data.
- This algorithm is same as that of selection sort, but with a small difference that in case heavy particles (i.e. the largest elements will eventually come up and hence the name is bubble sort.

BUBBLESSORT (A,N)

Here A is a linear array with N filled elements and this algorithm sort the element in ascending order using Bubble Sort Technique .

Step 1: Repeat step 2 and 3 FOR I =1 to N-1

Step 2: Repeat step 3 for J=1 to N-I

Step 2: if A[J+1] THEN

Set Temp: A[J]

Set A[J+1]:A[J+1]

A[J+1]:TEMP

[End if]

Step 4: Exit

Bubble Sort	Time Complexity	Space complexity
Average Case	O(n2)	o
Worst Case	O(n2)	o

Advantages:

- Bubble sort is simple to implement and understand.
- It will be very quick on an already sorted list.

Disadvantages:

- Time complexity of bubble sort is O(n2) same as that of selection sort.
- It is probably the slowest sort ever found.

Insertion Sort

- In this algorithm, the element is inserted at a position where it gives sorted list.
- That is, we start with the element, compare it with the first and then the third is compared with the first two and so on.

This sorting algorithm is frequently used when n is small. The insertion sort algorithm is small. The insertion sort scans A from A [1]to A [n], inserting element A[k-1]into its proper position in the previously sorted sub-array A[1], A[2],…, A[k-1].

Pass 1. A[1] by itself is trivially sorted.

Pass 2 . A[2] is inserted either before or after A[1], A[2] is sorted.

Pass 3. A[3] is inserted either before or after A[1], A[2], that is, before A[1], between A[1] and A[2] , or after A[2] , so that: A[1] ,A[2] , A[3] is sorted.

Pass 4. A[4] is inserted into its proper place in A[1] , A[2] , .. ,A [n-1] so that :A[1] ,A[2], A[3], A[4] is sorted.

Pass N. A[N] is inserted into its proper place in A[1], .. , A[N-1]so that: A[1], A[2] , … , A[n] is sorted.

Insertion sort

This algorithm sorts the array A with N elements

Step 1: repeat steps 2 to 4 for K=2 to N

Step 2: set temp :a[k] and PTR :=K-1

Step 3. Repeat while TEMP <A [PTR] and PTR>=1

(A) set A [PTR+1] :=A [PTR]

(B) set PTR:=PTR-1

[End of loop , step 3].

Step 4 set a [ptr+1]:=temp

[End of loop, step1]

Step 5. Exit

Advantages and Disadvantages of Insertion Sort:

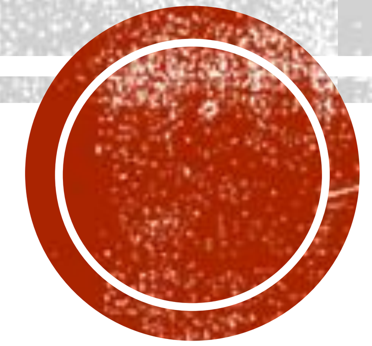
Advantages:

- **On the almost sorted arrays insertion sort shows better performance,upto o(n).**
- **It is also very easy to implement with a linear search.**

Disadvantage:

- **On an array,insertion is show becouse all the elements after it have to be moved up.**
- **This sort is applied to small data sets but not larges data sets.**

REPRESENTATION OF QUEUE IN THE MEMORY



•queue can be represented in the memory by two ways.

1.Queue using array

2. Queue using link list

(1) Queues using array: (static queue):

• The queue entered or data element can be stored in array.

•The queues represented using linear arrays are called as called as linear queues.

• operation mention on queue can be implemented on an array.

• The data we need for our array implementation of the queue are: an array and a count.

• The main property of a queue is that objects are inserted in the rear and come off from the front of the queue.

REPRESENTATION OF QUEUE IN THE MEMORY

- (2). Queue using link list:(Dynamic queue):

An alternative and efficient representation of queue is possible by using Linked lists for storing the data using dynamic memory.

- Advantage of storing queue in a linked list

The advantage of linked list representation of queue over array representation are:

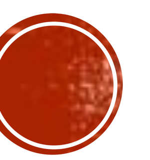
(1). There is no need of knowing the queue size before implementing it, as dynamic memory allocation technique helps helps the progammer to declare the memory space at run time.

(2). The queue is never full as long as the system has enough space for dynamic allocation. So there is no need to check theisfull or overflow condition of the concept of pointer in linked lists

Remember that a queue implemented using linked list is called as linked queue.

- **A linked list is a nodes *(a nodes can be a struct or a class)* defined arbitrarily in memory.**
- **A node has a special variable (pointer) of the same type as the structure/class that points to the next memory location thus making the list values continuous.**

The Frist node of a list is referred to as head/ start most of the time and the last node is referred to as tail/ end , the tail/ end of the list always points to NULL or 0, which is also referred to as grounding.

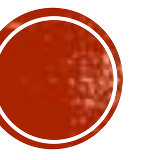


REPRESENTATION OF QUEUE IN THE MEMORY

- *Typedef stuct node*

```
{
    int data;
    Stuct node*next;
} node;

Typedef stuct Q
{
    node *R;
    node *F;
}
```



QUEUE OPERATIONS

- *1. Enqueue.*
- *2. Dequeue.*

(1). Enqueue:

- **it's very similar to the addition in a dynamic linked list.**
- **The only difference is that here you'll add the new element only at the end of the list, since a dynamic list is used for the Queue ,the queue is also dynamic,mean it has no prior size set.**
- **Aim is to entred the element in the queue.**
- **We don't have to check the Overflow condition in linked Queue.**

(2). Dequeue:

This is again very similar to deletion in linked list, but you can only delete from the head of the list, that makes it a Queue.

QUEUE OPERATIONS

- *Algorithm:*

(1).Enqueue opretion:

QUEHE_INSERT(Q, FRONT, REAR, N, ITEM)

STEP 1: If (REAR =N) the

write " Overflow" and Exit

[End If]

STEP 2: If (FRONT=NULL and REAR =NULL) then

Set FRONT: = 1, REAR :=1

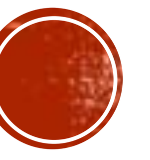
Eels

Set REAR = REAR +1

[End If]

STEP 3: Q [REAR]=ITEM

STEP4: EXIT



QUEUE OPERATIONS

- **Algorithm:**

(2). *DEQUEUE:*

QEUEUE_ DELETE (Q, FRONT, REAR, N, ITEM)

STEP 1: If (FRONT = NULL) Then

Write "UNDERFLOW" and Exit

[End If]

STEP 2: Set ITEM := Q[FRONT]

STEP 3: If (FRONT= REAR) Then

Set FRONT:= NULL

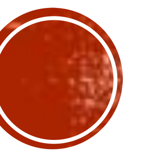
Set REAR := NULL

Else

Set FRONT:= FRONT +1

[END If]

STEP 4: EXIT



Stack

Definition of Stack

Stack is a list in which all the insertion and deletion are made at one end, called the top of the stack.

Example of Stack

- Real-life examples of a stack are a deck of cards, piles of books, piles of money, and many more. This example allows you to perform operations from one end only, like when you insert and remove new books from the top of the stack.

Representation of Stack in Memory

Usually stack can be represented by two ways:

- **Using Array**
- **Using Link List**

Representation of Stack using Array

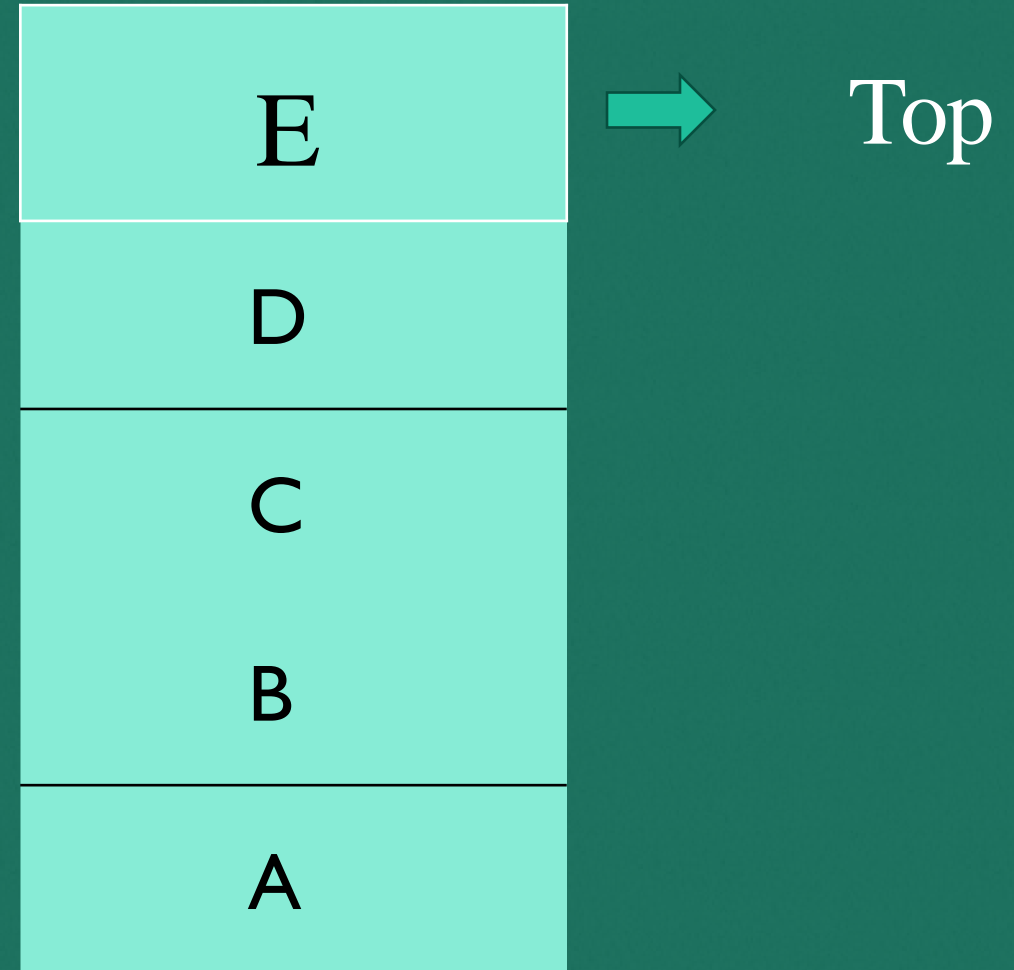
- In the representation of stack as an array the size of stack is fixed i.e. the maximum number of element it can accommodate must be known in advance.

- The top of the stack is the first element of array.

- The stack item are placed from item [0] to item [top-1].

```
# define MAX 50
Struct stack
{
                Int top ;
                Int item [MAX];
};
```

- A stack is represented in the memory as



- A stack is represented with the help of array in the memory as

index	0	1	2	3	4
	E	D	C	B	A

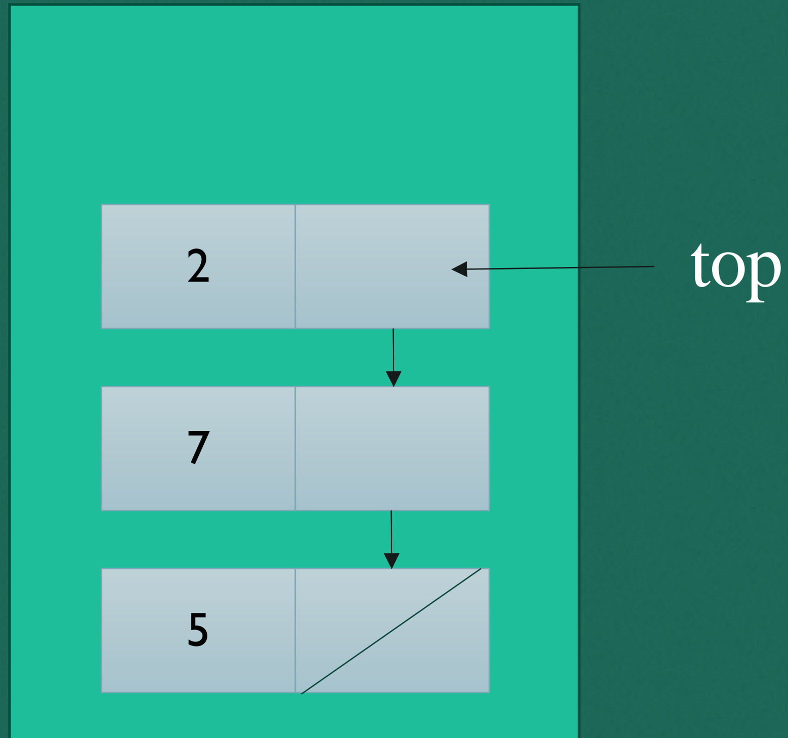
- Stack grows from L to R.

Representation of Stack Using Link List

- The major problem in the implementation of stack using array is that it suffers the basic limitation of an array that its size cannot be increased or decreased once it is declared.
- Stack can be represented efficiently using linked list.
- This problem can be solved by implementing stack using link list. For this type of implementation, let us use singly link list for storing stack elements.
- The top of stack is given by a pointer, the top pointing to first item of the list i.e. in the beginning of the list.
- The stack element ordered from top to bottom in link list from left to right.

In C the stack node can be defined as

```
Struct stack
{
        int data;
        Struct stack * link;
} node;
```



- Here top of the stack is a node having value 2, linked with the next node having value 7 and last node is having value 5 with its address field NULL.

Push operation

- To add new element in the stack, we must perform the two step which are as follows.

a) Increment top indicator
b) Put new element at new top.

Algorithm

```
PUSH  (STACK, TOP, MAXSTK, ITEM)
Step 1:    IF TOP = MAXSTK Then
                                 Print : Overflow and go to step 4
Step 2:   Else Set Top = Top + 1        // increment top by one
Step 3:    Set STACK [TOP] := ITEM       // insertion of an element in top of
stack
Step 4:    Stop
```

Operations of Stack

There are two basic operations on the stack:

i. Push (a, b) to insert the element "b" on the top of stack "a"

ii. Pop (a) to remove the top element of the stack "a" and return the removed element as a function value.

Explanation of Push Operation

Push mean Add an item on top

Step 1 Check whether the top of STACK is having the maximum number of element it can accommodate. If it contains maximum number then print overflow.

Step 2 Otherwise (Means if condition is false) then add an element in the top of the stack.

Step 3 Now the top element of the stack is the item that you have inserted recently.

To implement the push operation, there may be a situation when the size of stack is equal to the size of declared array size. Then, we cannot push any element on to the stack. Such type of operation is referred to as an Overflow.

Therefore to push the element, we must ensure that stack is not full.

Pop Operation

Algorithm : To delete or pop an item from the top

Pop (STACK, TOP, ITEM)

This algorithm delete or pop the top element from the stack and assign it to variable name item.

Step 1: If Top = 0 then
Print "underflow" and go to step 4
//(underflow means there is no item in the stack)

Step 2: Else Set ITEM = STACK [Top]

Step 3: Set top := top – 1

Step 4: Exit

Explanation of Pop Operation

Step 1 To check whether there is an element present in the stack or not. If no element is there in the stack then there is condition of underflow.

Step 2 Otherwise Assign the value of Top element to item.

Step 3 Perform the operation, Top = top – 1

Step 4 Stop

To remove an element from the top of stack, we must first check the possibility of underflow as it is quite possible that somebody tries to pop an element from an empty stack.

Thank You

Shivalik college naya nangal

<u>from BCA department</u>

Govt. Shivalik College Naya Nangal

Operation In Array...

Insertion and deleting from array

Inserting Element In The Array...

- Inserting means adding new element in the array.
- Insertion of an element at the end of linear array is at the end.
- Consider the following :

10	20	30	40	50			

Initial Array

10	20	30	40	50	60		

After insertion an element at end

Algorithm….

- An array named , A is linear array with an element in it.
- LB is lower bound and UB is upper bound.
- Data is a positive integer which is to be inserted at Kth position in array A.

1. Start
2. Assign the value of n to variable i.
3. Repeat step IV and V till I>=k
4. Move the ith element down ward by making assignment.

```
5    A[i+1] = A[i]
6     i = i-1
7    A[k] = data
8     N= n+1
9    Stop
```

Deleting Element from Array....

- To remove the element from array.
- An array named , A is linear array having N element in it.
- Consider the following :

10	20	30	40	50			

Initial Array

10	20	30	40	50	60		

After insertion an element at end

Algorithm....

1. Start
2. Data = A[k]
3. Repeat step V. Stating from initial value of variable i from K to final value say n-1.
4. Move (i+1) element upward by making assignment A [i] = [i+1]

5 $N = n - 1$

6 Stop

Sparse Matrix.....

- A matrix that has relatively higher number of zeros than the non zero elements is known as sparse matrix.
- A Matrix can be define as a two dimensional array having 'M' columns and 'N' representing m *n matrix.

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Fig. 3.10 (a)

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Fig. 3.10 (b)

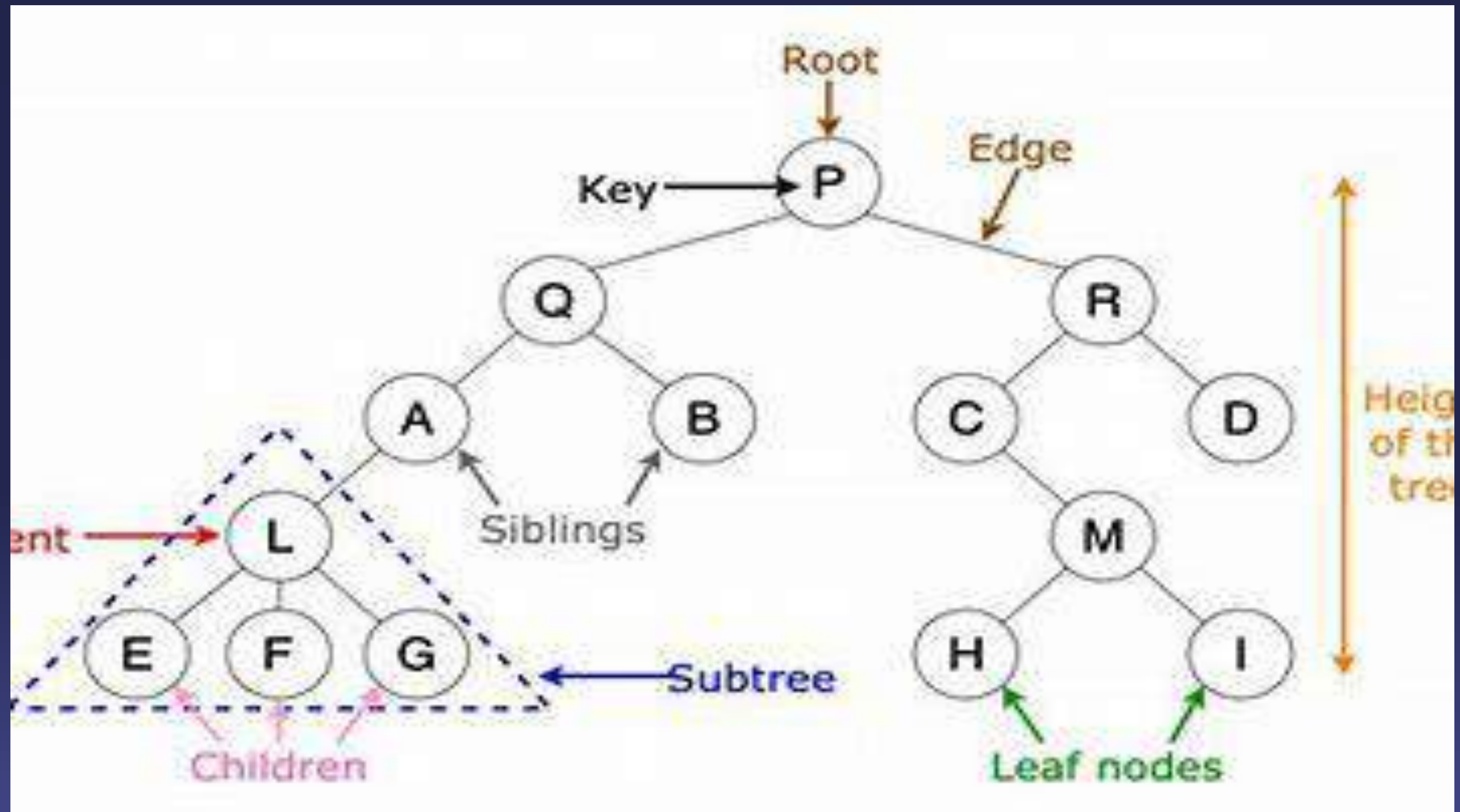
- In fig 3.10 only 2 memory unit utilized and other seven are Wastage of memory space.
- Similarly in fig 3.10(b) only 3 units are utilized and other 17 are zeros, hance there is wastage of memory space as well as time.

- Sparse matrix are those matrix that have the majority of their elemens equal to zero.

**** To save the space of memory and time , sparse Matrix requires some sparse representation ****

Thank you….

DATA STRUCTURE(TREES)



TREES

A tree data structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes. The topmost node of the tree is called the root, and the nodes below it are called the child nodes. Each node can have multiple child nodes, and these child nodes can also have their own child nodes, forming a recursive structure[1].

The tree data structure has roots, branches, and leaves connected with one another. The root node is the topmost node of a tree or the node which does not have any parent node. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree. The nodes which do not have any child nodes are called leaf nodes[1].

There are different types of trees such as general tree, binary tree, binary search tree, AVL tree, B-tree, etc.[2]

REPRESENTATION OF TREES IN MEMORY

- A binary tree can be represented in memory using pointers. Each node in the tree contains a data element and two pointers to its left and right child nodes. The root node is represented by a pointer to the topmost node of the tree. If the tree is empty, then the value of the root is NULL.

BINARY TREES

- A binary tree is a tree data structure where each node has at most 2 children. Since each element in a binary tree can have only 2 children, we typically name them the left and right child. A binary tree is represented by a pointer to the topmost node (commonly known as the “root”) of the tree. If the tree is empty, then the value of the root is NULL. Each node of a Binary Tree contains the following parts: Data. Pointer to left child. Pointer to right child.

BINARY TREE TRAVERSAL

- There are three types of binary tree traversal:
- In order Traversal
- Preorder Traversal
- Post order Traversal
- In order traversal is used to traverse the left sub tree first, then visit the root node, and finally traverse the right sub tree. Preorder traversal is used to visit the root node first, then traverse the left sub tree, and finally traverse the right sub tree. Post order traversal is used to traverse the left sub tree first, then traverse the right sub tree, and finally visit the root node.
- I hope this helps! Let me know if you have any other questions.

BINARY SEARCH TREES

- A binary search tree (BST) is a binary tree data structure in which the value of each node in the left sub tree is less than or equal to the values in the node itself, and the value of each node in the right subtree is greater than or equal to the values in the node itself.
- A binary search tree is a binary tree with the following properties:
- The left sub tree of a node contains only nodes with keys lesser than the node's key.
- The right sub tree of a node contains only nodes with keys greater than the node's key.
- The left and right sub tree each must also be a binary search tree.
- There must be no duplicate nodes.