Practical 1:

Aim: Linear Search

Q1] Unsorted Array:

Algorithm:-

Step 1: Define a function with 2 parameters. Use for condi statement with range i.e. length of array to find index.

Step 2: Now use if conditional statement to check whether the given no. by user is equal to the elements in the arrays.

Step 3: If the condition in step 2 is satisfied then return index of given array. If the the condi satisfies then get out of loop

Step 4: Now initialize a variable to enter elements in the arrays from user. Now use split() to split values

Step 5:- Now initialise a variable as empty array.

Theory:-

Linear search is one of the simplest searching algorithm in which each items is sequentially matched with each item in the list.

It is worst searching algorithm with worst case time complexity. It is a fore approach on the other hand instead of searching list in sequence. A binary search is used which will start by examining the middle term. Linear search is a technique to compare each and every element with the key element to be found if both of them matches then its position is found.

sorted array
code:

```
def linear(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i

inp = (input("Enter elements in array:")).split()
array = []

for ind in inp:
    array.append(int(ind))
print('Elements in array are:", array)
array.sort()
x1 = int(input("Enter element to be searched:"))
x2 = linear(array, x1)

if x2 == x1:
    print("Element found at position", x2)

else:
    print("Element not found")
```


Step 6: Now use for conditional statement to append the element given as input by user in the empty array.

Step 7:- Now again initialise another variable to ask user to find the element in the array.

Step 8:- Again initialise a variable to call the defined function.

Step 9:- Use if conditional statement to check if variable in step 8 matches with the element your want to find the print the index corresponding to the element. If condition not staisfied print element not found.

Q3] Unsorted Array:

Algorithm:-

Step 1:- Define a function with 2 parameters use for conditional statements with range i.e. length of array to find index.

Step 2:- Now use if conditional statement to check if & whether the given statement is equal to the elements in arrays.

Step 3:- If the condn in step 2 is satisfied return the index no of given array. If the condn doesn't satisfy then get out of the loop

Step 4:- Now initialise a variable to enter elements in the arrays from user Now use split() to split the values.

Unsorted array

code:-

```
def linear(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i

inp = input("Enter elements in array:").split()
array=[]
for ind in inp:
    array.append(int(ind))
print("Elements in array are:", array)
x1 = int(input("Enter element to find"))
x2 = linear(array, x)
if x2 == x1:
    print("Element found at location:", x2)
else:
    print("Element not found:")
```

Output for sorted array :-

>>> Element in array : 1 2 3 5
>>> Elements in array are : [1, 2, 3, 5]
>>> Enter element to search : 2
Element found at position : 1
>>> Elenent in array are : [1, 2, 3, 4, 5]
>>> Element to search : 7
Element not found.

Output for unsorted array:

>>> Elements in array : 3, 2 4 5 1
>>> Elemlent in arrayare : [3, 2, 4, 5, 1]
>>> Element to be searched : 4
found at location 2.

>>> Element in array : 2 4 5 3 1
>>> Elements in arrays are [2, 4, 5, 3, 1]
>>> Element to be searched : 6
Not found.

Steps :- Now initialize a variable as array i.e. empty

Steps :- Now use for conditional loop statement to append the element as input by user in empty array

Step 3 :- Now again initialize another variable to ask for user to find element in the array

Step 8 :- Again initialize a variable to call the function.

Step 9 :- to check using if condn statement to see if variable in step 8 matches the elements in the list. If not satisfied print not found.

8.8 Practical 2:

Aim: Binary search

Algorithm:

Step 1: Define a function with 2 parameters. Now initialize variable with 0 and while loop to found the mid value

Step 2:- Use if condition statement to determine at which position the mid value should point

Step 3:- If the cond doesn't satisfies return -1

Step 4:- Now initialise a variable to enter elements in array

Step 5:- Use for condition statement to append the element in empty array.


```
# binary search
# code
def binary (arr, key):
    start = 0
    end = (en(arr)
    while start < end:
        mid = (start + end) // 2
        if arr[mid] > key:
            end = mid
        elif arr[mid] < key:
            start = mid + 1
        else:
            return mid
    return -1
arr = input ('Enter element in sorted:').split()
arr = []
for ind in array:
    arr.append (int (ind))
key = int (input ('Enter element to search:'))
index = binary (arr, key)
if index < 0:
    Print ('Element not found:')
else:
    Print ('Element not found:', index)
```

>>> Enter the elements in array:
3 5 10 12 15 20

>>> Element to be searched: 12
12 element was found at index 3

>>> Enter the elements in array
-3, 0 1 5 6 7 8

>>> Element to search: 2
Element was not found

Steps: Now initialise a variable to find element in array

Step7:- Now calls the function

~~Step8:~~

Theory:- Binary search is also known as half interval search logarithmic search or binary search algorithm that finds the position of the target. If you are looking for no. which is at end of list then you need to search at the entire list.

Practical 3:

Aim: Implementation of Bubble sort program on given list.

Theory:- Bubble sort is based on the idea of repeatedly comparing pairs of adjacent elements and then swapping their position. If the simplest form of sorting is available. In this we sort the given elements in ascending order or descending order by comparing adjacent element at a time.

Algorithm:-

Step 1:- Bubble sort algorithm starts by comparing 1st 2 elements of an array and swapping them.

Step 2:- If we want to sort the elements of array in assendig order then 1st element is greater than second then we need to swap the element.

Step 3:- If the 1st element is smaller than second element we do not swap the element.

#Bubble sort
code

```
inp = input('Enter Elements').split()
arr = []
for ind in inp:
    arr.append(int(ind))
Print('Element of array before sorting them:",arr)
n = len(arr)
for i in range(0,n):
    for j in range(n-1):
        if arr[i] * arr[j]:
            temp = arr[j]
            arr[j] = arr[i]
            arr[i] = temp
    Print('Element of array after bubblesort:",arr)
```

>>> Enter element : 2 3 6 1 8 5

>>> Enter Elements of arrays before sorting=[2,3,6,1,5]

>>> Sorted list[1,2,3,5,6]

24/12

Step 4: Again second and 3rd third element are compared and swapped if it is necessary and this process go on until last & second second last element is compared and swapped.

Step 5: If there are n elements to be sorted then the process mentioned above should be mentioned to get required ~~its~~ result.

Step 6: Stick the output & input of above algorithm of bubble sort stepwise.

Practical 4: Stack

Aim:- Implementation of stack using Python list

Theory:- A stack is linear data structure that can be represented in a real world form by physical stack or pile. The elements in the form of stack at the top most position. Therefore it works in LIFO (Last in first out). It has 3 basic operations namely: push, pop, peek

Algorithm:-

Step 1:- Create a class stack with instance variable items

Step 2:- Define the init method with self argument and initialize the initial value and the initialize the initial value and then intake to an empty value.

Step 3:- Define method push & pop and Class stack

Step 4:- Use if statement to give the condition that if length of given list is given range of list then print stack is

```
# code
class Stack():
    def __init__(self):
        self.l = [0, 0, 0, 0, 0]
        self.tos = -1
    def push(self, data):
        n = len(self.l)
        if self.tos == n-1:
            Print("The stack is full")
        else:
            self.tos += 1
            self.l[self.tos] = data
    def pop(self):
        if self.tos < 0:
            Print("The stack is empty")
        self.l[self.tos] = 0
        self.tos -= 1
x = stack()
```

output

```
>>> x.push(10)
>>> x.push(9)
>>> x.push(8)
>>> x.push(7)
>>> x.push(6)
The stack is full
```

```
>>> x.pop
7
>>> x.l
[10, 9, 8, 6]
```


Step5:- Else print Enter element.

Step6:- Push method to insert element & pop is used to delete the element from stack.

Step7:- If pop ~~value~~ method value is less than 1 then return stack is empty or else delete the element from stack at top most position

Step8:- Assign the element values in push, method & print given

Step9:- Attach the inp & output of above algorithm

Step10:- First condition ~~the~~ checks whether the no of elements are zero while 09/09/22 second ~~value~~ case whether top is assigned any value. If top is not assigned any value then print stack is empty.

Practical No. 5

Q1) Aim - Implement Quick sort to sort the given

Theory :- The quick sort is a ... algorithm then based on quick ... conquer technique.

ALGORITHM :-

Step 1 :- Quick sort first selects a value which is called as pivot value first value element. It serve as ... first pivot value. Since we know ... first will ~~ever~~ eventually endup as last in that list.

Step 2 :- The partition process will happen next. It will find the split point and at same time move other items to appropriate side of the list either less than or greater than pivot value.

Step 3 :- Partioning begins by locating ... markers lets call them left mark ... right mark at the begging ... end of remaining items in the ... The goal of partition process is ... move items that are on the wrong ... with respect to ... values ...

```
# code
print ("Quick Sort")
def partition (arr, low, high):
    i = low-1
    pivot = arr[high]
    for j in range (low, high):
        if arr[j] <= pivot:
            i = i+1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i+1], arr[high] = arr[high], arr[i+1]
    return i+1
def quicksort (arr, low, high):
    if low < high:
        pi = partition (arr, low, high)
        quicksort (arr, low, pi-1)
        quicksort (arr, pi+1, high)
x1 = input ("Enter element in the list").split()
alist1 = []
for b in x1:
    alist1.append (int(b))
print ("Element in list are:", alist1)
n = len (alist1)
quicksort (alist1, 0, n-1)
print ("Elements after quick sort are:", alist1)
```

output:-

>>>

Quick sort

Enter elements in the list 21, 22, 20, 30, 24, 34

[OUT] elements in list are: [21, 22, 20, 30, 24, 34]

elements after quicksort: [20, 21, 22, 24, 30, 34]

converge on the split point.

Step 4:- we begin by increasing the leftmark ~~use~~ it locates a value greater than P.V. Then decrement rightmark until we find value i.e less than P.V. At this point we have found 2 items that are out on place.

Step 5:- ~~At~~ the point the right rightmark becomes less than leftmark we stop. The position of rightmark is now the split point.

Step 6:- The pivot point can be ~~split~~ exchanged with content of split point & P.V. is now in place.

Step 7:- In addition ~~all~~ all the elements to the left of ~~p.v~~ ~~leftmark~~ p.v is less than the pivot mark. All items to the right is greater than P.V.

Step 8:- The quick() invokes a recursive()

Step 9:- quick sort helper, begins with same basic as the merge sort.

Step 10:- If length of list is less than 0 or equal one it is already sorted.

Step 11:- If it is greater than it can be partitioned & recursive ()

Step 12:- The partition() ~~process~~ implements the described earlier.

Step 13:- Display & stick the coding of above algorithm.

Q II]

Aim:- Implementation of Stack using

Theory:- A stack is a linear data structure that can be represented in the real world in the form of a physical stack or a pile. The elements in a stack are added or removed only from one position i.e. the top position. Thus stack work LIFO principle as the element that was inserted last will be removed first. A stack can be implemented using Array as well as linked. STACK has 3 basic operations:

```
CODE
class Stack:
    global tos
    def __init(self):
        self.l=[0,0,0,0,0,0]
        self.tos = -1
    def push (self, data):
        n = len (self.l)
        if self.tos ==n-1:
            print ("Stack is full")
        else:
            self.tos = self.tos+1
            self.l [self.tos] =data
    def pop (self):
        if self.tos<0:
            print ("Stack is empty")
        else:
            k = self.l [self.tos]
            print ("data =", k)
            self.l [self.tos] = 0
            self.tos = self.tos -1
```

push, pop, peek. The operation of adding & removing the elements is known as Push & Pop.

ALGORITHM:-

Step 1:- Create a class with instance variable items.

Step 2:- Define the init() with self argument & initialize the initial value to -1. Then initialise to an empty list.

Step 3:- Define methods push & pop under class stack.

Step 4:- Use if condition to give the condition that if length of given list is greater than range of list then print: stack is full

Step 5: Else print statement as insert the element into the stack & initialise the value.

Step 6:- Push() used to insert the element put POP() use to delete the element from stack

Step 7: If in POP() value is less than
print The stack is empty or
delete the element for stack
topmost position.

Step 8:- First condition checks whether
no of elements is zero while
second case whether tos is
to any value. If tos is
to any value. So we can be
sure that stack is empty.

Step 9:- Assign the elements values
Push() to add & print the
is popped.

Step 10:- Attach the input & output
above Algorithm.

Output:-

= stack()
.push(10)
.push(20)
.push(30)
.push(40)
.push(50)
s. push(60)
s. push(99)
s. push(80)

Input:-

s.POP()
s.POP()
s.POP()
s.POP()
s.POP()
s.POP()
s.POP()
s.POP()

Output:-

Stack is full
data = 70
data = 60
data = 50
data = 40
data = 30
data = 20
data = 10
stack is empty

II Code:

```
Class Queue:
    global r
    global f
    def __init__(self):
        self.r = 0
        self.f = 0
        self.l = [0, 0, 0, 0, 0, 0]
    def add(self, data):
        n = len(self.l)
        if self.r < n-1:
            self.l[self.r] = data
            self.r = self.r + 1
        else:
            print("Queue is full")
    def remove(self):
        n = len(self.l)
        if self.f < n-1:
            print(self.l[self.f])
            self.f = self.f + 1
        else:
            print("Queue is empty")
Q = Queue()
```


Practical 6:-

Title: Implementing a Queue using python list

Theory: Queue is a linear data structure which has 2 ~~queues~~ front & rear implementing a queue using python list is the simplest as the as python list provides inbuilt functions to perform the specified operations of the queue. It is based on principle that a new element is is inserted after rear end element of the queue is deleted which is at front. In simple term, a queue can be distributed as a data structure based on first in first out.

Queue(): Creates a new empty queue.

Enqueue():- Insert an element at the ~~top~~ rear of the queue and similar to that of insertion of linked list by "tail".

Dequeue():- Returns the element which was at front the front is moved to successive element & dequeue operations cannot remove element if queue is empty.

Algorithm

Step 1:- Define a class Queue & assign global variables then define __init__ with self argument in init assign or initialize the initial val with help of self argument.

Step 2:- Define a empty list and define enqueue() method with 2 arguments assign the length of empty list

Step 3:- Define Queue() with self argument use if statement that front is equal to length of list then display Queue is empty or else give that front is at 0 & using that delete the element from front side & increment it by 1.

Step 3 Use if statement that length is equal to rear then queue is full or else insert the element in empty list or display that Queue element added successfully & increment by it

Step 5:- Now all the Queue() function be given the element that has to be added in the empty list by enqueue & Print list after [illegible]

Output :-

Q.add(30)
Q.add(40)
Q.add(50)
Q.add(60)
Q.add(70)
Q.add(80)
Q.add(90)
Queue is full
Q.remove()
30
Q.remove()
40
Q.remove()
50
Q.remove()
60
Q.remove()
70
Q.remove()
80
Q.remove()
Queue is empty

```
# code
def evaluate(s):
    k = s.split()
    n = len(k)
    stack = []
    for i in range(n):
        if k[i].isdigit():
            stack.append(int(k[i]))
        elif k[i] == '+':
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) + int(a))
        elif k[i] == '-':
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) - int(a))
        elif k[i] == '*':
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) * int(a))
        else:
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) / int(a))
    return stack.pop()
s = "869*+"
r = evaluate(s)
```


Practical No 7

Aim:- Program on Evaluation of given string by using stack in python enviornment i.e Postfix

Theory:- The postfix expression is free of any parantheses. Further we took care of the properties of the operators in ~~code~~ program. A postfix expression can be easily be evaluated using stack. Reading the expression is always from left to right in postfix.

Algorithm:-

Step 1:- Define evaluate as a function then Create a empty stack in python

Step 2:- Convert the string to a list by using the string method 'split'

Step 3:- Calculate the length of string & print it

Step 4:- Use for loop to assign the range of string then give condition using if statement.

Steps:- Scan the token list from le
right. If token is an operan
convert it from a string to int
& push the value onto p

Step 6:- If the token is an operator *,
+ - , ^ it will need two
operands pop the 'p' twice. The
pop is second operand & the
pop is the first operand

Step 7:- Perform the arithemetic opera
Push the result result back on
the 'm'

Steps:- When the input expression ha
been completely processed the
is on the value

Step 9:- Print the result of string aft
the evaluation of postfix

Step 10:- Attach output & input of ab
algorithm

print ("The evaluated value is:", r)

output:

The evaluated value is: 62

```
# code
class Node:
    global data
    global next
    def __init__(self, item):
        self.data = item
        self.next = None
class LinkedList:
    global s
    def __init__(self):
        self.s = None
    def addL(self, item):
        newnode = node(item):
        if self.s == none:
            self.s = newnode
        else:
            head = self.s
            while head.next != None:
                head = head.next
            head.next = newnode
    def addB(self, item):
        newnod = node(item)
        if self.s == none
```


Practical 8:

Aim:- Implementation of single linked list by adding the nodes from last position

Theory:- A linked list is a linear data structure which is storing the elements in a node in a linear fashion but no nessarily continues. The individual element of the linked list called as as a Node comprises of 2 parts ① Data ② Next Data stores all the information wrt the element whereas next refers the next node

Algorithm:

Step1:- Traversing of a linked list means positioning all the nodes in the linked list in order to perform some operation on them.

Step2:- The entire linked list means can be accessed as the 1st node of the linked list.

Step3:- Thus the entire linked list can be traversed using the node which is reffered by the head pointer of linked list.

Step4:- Now that we know that we can traverse the entire linked list using head pointer we should only modify the first node of the list.

Step5:- We should not use the head pointer to traverse the entire linked list because the head pointer is our only reference to 1st node.

Step6:- We may lose the reference to nodes in our linked list if most of our linked list. So in order to avoid making some unwanted changes to the 1st node we will use temporary node to terminate the entire linked list.

Step7:- We will use this temporary node as a copy of the node we currently traversing since we are making temporary node as copy of current node the data of the temporary node should also not

Step8:- Now that current is referring to the 1st node if we want to access 2nd node of list we need

```
        self.s = newnode
    else:
        newnode.next = self.s
        self.s = newnode
def display(self):
    head = self.s
    while head.next != None:
        print(head.data)
        head = head.next
    print(head.data)
start = LinkedList()
print("output:")
start.addL(80)
start.addL(70)
start.addL(60)
start.addL(50)
start.addL(40)
start.addB(30)
start.addB(20)
start.display()
```

```
20
30
40
50
60
70
80
```

Step4:- Now that we know that we can traverse the entire linked list using head pointer we should only maintain refn the first node of the list.

Step5:- We should not use the head pointer to traverse the entire linked list because the head pointer is our only reference to 1st node

Step6:- We may lose the reference to in our linked list & most of our linked list. So in order to avoid making some unwanted changes to the 1st node we will use temporary node to iterate the entire linked list.

Step7:- We will use this temporary node as a copy of the node we currently have. Since we are making temporary node copy of current node the data of temporary node should also

Step8:- Now that current is referring to the 1st node if we want to access 2nd node of list we have to refer as next node of 1st node

```
        self.s = newnode
    else:
        newnode.next = self.s
        self.s = newnode
def display(self):
    head = self.s
    while head.next != None:
        print(head.data)
        head = head.next
    print(head.data)
start = LinkedList()
# output:
>> start.addL(80)
>> start.addL(70)
>> start.addL(60)
>> start.addL(50)
>> start.addL(40)
>> start.addB(30)
>> start.addB(20)
>> start.display()
>> 20
   30
   40
   50
   60
   70
   80
```

Step 9:- But the 1st node is reffred by current. so we can traversed to 2nd nodes as nth = h.next

Step 10:- Similarly we can traverse rest of nodes in the linked list.

Step 11:- Our concern now is to find terminating condition for the while loop.

Step 12:- The least node in the linked list is reffred by tail of linked list.

Step 13:- So we can refer to last node of linked list.

Step 14:- We have to now see how to start traversing the linked list & how to identify by whether then we have reached the last node.

Step 15:- Attach the coding or input & output of above algorithm.

Practical 9

Aim. Implementation of merge sort by using python

Theory. Merge sort is a divide & conquer algorithm. It divides input array into 2 halves, calls itself for 2 halves & merges the 2 sorted halves. The merge() is used for merging 2 halves.

Algorithm:

Step 1 - The list is divided into left & right in each recursive call until 2 adjacent elements are obtained.

Step 2:- Now begins the sorting process. The i & j iterators traverse the 2 halves in each call. The k iterator traverses the whole lists & makes changes along the way.

Step 3:- If the value at i is smaller than the value at j L(i) is assigned to the arr[i+1] slot & i is incremented. If not then R[j] is chosen

code

```
def sort (arr, l, m, r):
    n1 = m-l+1
    n2 = r-m
    L = [0]* (n1)
    R = [0]* (n2)
    for i in range (0,n1):
        L[i] = arr (i+1]
    for j in range (0,n2)
        R[j] = arr [m+1+j]
    i = 0
    j = 0
    k = L
    while i < n1 && j < n2:
        if L[i] <= R [j]:
            arr[k] = L[i]
            i = i+1
        else:
            arr (k] = R[j]
            j = j+1
        K = K+1
    while i < n1:
        arr [k] = L[i]
        i = i+1
        k = K+1
```

```
while j
    arr[k] = R[j]
    j += 1
    k = k+1
def mergesort(arr, i, u):
    if i < u:
        m = int((l+(r-1))/2)
        mergesort(arr, l, m)
        mergesort(arr, m+1, r)
        sort(arr, l, m, r)
arr = [12, 23, 34, 56, 78, 56, 98, 42]
Print(arr)
n = len(arr)
mergesort(arr, 0, n-1)
Print(arr)
```

Output:

[12, 23, 34, 56, 78, 45, 86, 98, 42]

[12, 23, 56, 56, 42, 45, 78, 86, 98]

Step 4:- This way, the values being assigned through n[i+1] are all sorted

Step 5:- At the end of this loop, one of the halves may not have been traversed completely remaining slots in the lists

Step 6:- Thus, the merge sort has been implemented.

Practical 10:

Aim:- Implementation of sets using ...

Algorithm:-

Step 1:- Define 2 empty sets set1, set2. Now use for statements ... the range of above 2 sets.

Step 2:- Now add() is used for adding of elements in the given ... then print sets for adding.

Step 3:- Find the union & intersection of above 2 sets by using AND, OR methods Print sets of union & ... as set3.

Step 4:- Using if statement find out subset superset of set3 & set4. Display above set.

Step 5:- Display the element in set3 is ... set4 using mathematical op.

Step 6:- Use clear() to remove or delete ... & print set after clearing the elements present in the set.

```
code
1 = set()
2 = set()
i in range(9,15):
    set1.add(i)
  i in range(1,12)
    set2.add(i)
int("set1:", set1)
int("set2:", set2)
int("\n")
et 3 = set1 / set2
rint("Union of set1 & set2 : set3 ", set3)
et 4 = set1 & set2
rint("Intersection of set1 & set2: ~~set4~~ ", set4)
Print("\n")
  set 3 > set4:
      Print("set 3 is super set of set4")
lif set3 < set4:
      Print("set3 is subset of set4")
lse:
   Print("set3 is same as set4")
f set4 < set3:
    Print("set4 is subset of set3")
    Print("\n")
et 5 = set 3 - set4
```

```
Print("Elements in set 3 & not in set4:
Print("\n")
if set4.isdisjoint(set5):
        Print("set4 & set5 are mutually
                exclusively")

set.clear()
Print("After applying clear, set5 is empty
Print("set5 =", set5)
```

Output:

set1: {8, 9, 10, 11, 12, 13, 14}

set2: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

Union of set1 & set2: set3

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

Intersection of set1 & set2 {8, 9, 10, 11}

set 3 is super set of set4.

Elements in set 3 & not in set4: set5

{1, 2, 3, 4, 5, 6, 7, 12, 13, 14}

set4 & set5 are mutually exclusive

after applying clear set5 is empty set

set5 = set()

Practical 11:-

Aim:- Program based on binary search tree by implementing inorder, preorder, postorder transversal

Theory:- Binary tree is a tree which supports maximum of 2 children for any node in the tree. Thus any particular node have either 0 or 1 or 2 children. There is another identity of binary tree that it is ordered such that one child is identified as left child & other as right child.

- **Inorder:-** i) Traverse the left sub tree. The left sub tree in turn might have left & right sub trees.
- **Pre-order:-** i) Visit the root node ii) Traverse the left subtree & right sub tree
- **Post order:-** Traverse the left sub tree the left subtree in turn might have left & right sub trees

CODE

```
class node:
    def __init__(self, value):
        self.left = None
        self.val = value
        self.right = None

class BST:
    def __init__(self):
        self.root = None

    def add(self, value):
        p = node(value)
        if self.root == None:
            self.root = p
            print("Root is added sucessfully", p.val)
        else:
            h = self.root
            if p.val < h.val:
                if h.val == None:
                    h.left = p
                    print(p.val, "Node is added successfully")
                    break
                else:
                    h = h.left
            else:
                if h.right == None:
```

```
        n.right
        Print(p.val, "is added to right side successfully")
        break
    else:
        n = n.right
def Inorder(root):
    if root == None:
        return
    else:
        Inorder(root.left)
        Print(root.val)
        Inorder(root.right)
def Preorder(root):
    if root == None:
        return
    else:
        Print(root.val)
        Preorder(root.left)
        Preorder(root.right)
def postorder(root):
    if root == None:
        return
```


Algorithm:

Step 1: Define class node & define init() with 2 arguments. Initialize the value in this method.

Step 2:- Again, define a class BST that is binary search tree with init() with self argument & assign the root is none

Step 3: Define add for adding the node Define a variable p that p = node(value)

Step 4: Use if statement for checking the condition that root is none then use else statement for if node is less than then main node then put on arrange them in left side

Step 5:- Use while loop for checking the node is less than or greater than the main node & break the loop if it is not satisfying.

Step6:- Use if statement within that else statement for checking that node is greater than main root then put it to right side

Step7:- After this, left side tree & right subtree repeat this method into arrange the node as to binary search tree.

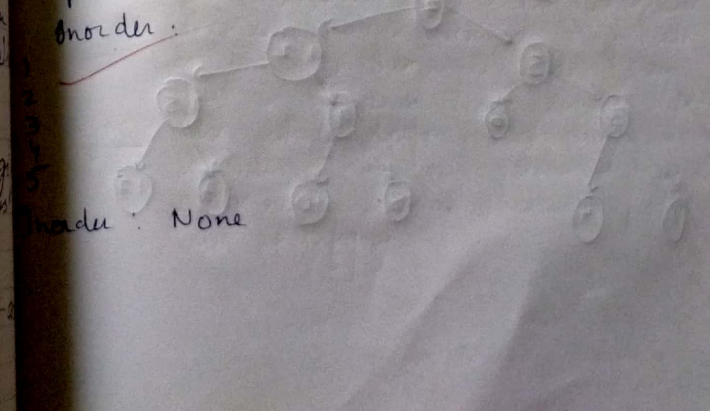
Step8:- Define Inorder() preorder() & postorder with root as argument & use if statement that root is none & return thats all

Step9:- In order else statement used for giving that condition if first the root, & then right node.

Step10:- For preorder, we have to give condition in else that first root, left & right node

Step11:- for post order in else part left & right & root

```
        else:
            Post order (root.left)
            Post order (root.right)
            Print (root.val)
t = BST()
output:
t.add(1)
root is added successfully !
t.add(2)
2nd node is added successfully to the right
t.add(4)
4 node is added successfully to the right
t.add(5)
5 node is added successfully to the right
t.add(3)
3 node is added to left side successfully
print("In Inorder:", Inorder(t.root))
Inorder:
1
2
3
4
5
Inorder: None
```



>>> Print("\n Preorder: ", Preorder(root))

Preorder

1
2
3
4
5

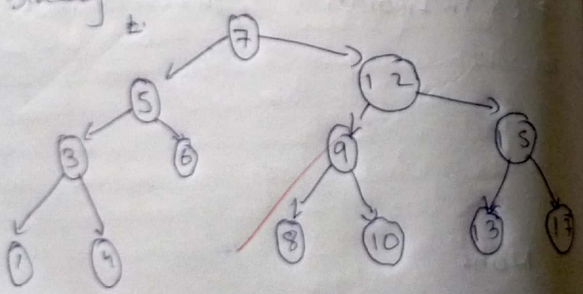
Preorder: None

>>> Print("Post order:", Postorder(t.root))

3
5
4
2
1

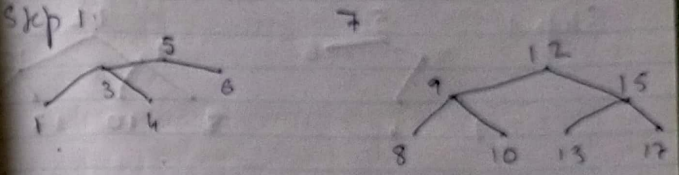
Postorder: None

* Binary search tree:

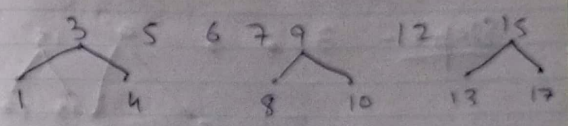


InOrder (LVR):

Skp 1:



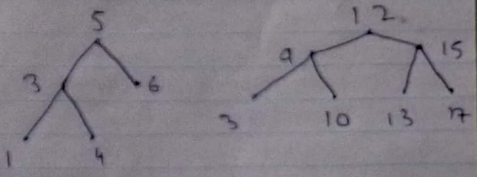
Skp 2:



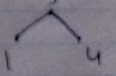
Skp 3: 1,3,4,5,6,7,8,9,10,~~12,13~~,15,17.

Pre order: (VLR)

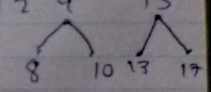
Skp 1: 7



Skp 2:- 7 5 3



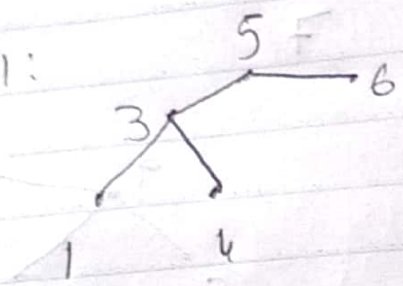
6 12 9

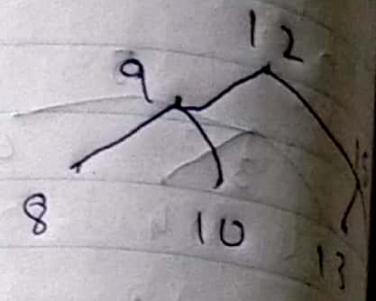


Skp 3: 7, 5, 3, 1, 4, 6, 12, 9, 8, 10, 15, 13, 17,

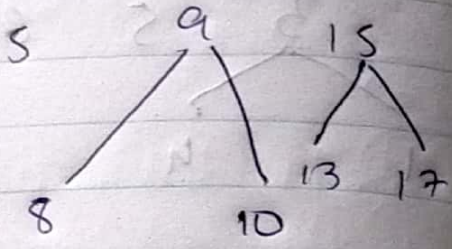
- POST ORDER : (L R V)

Step 1:





Step 2:- 3 6 5



Step 3: 1, 4, 3, 6, 5, 8, 10, 9, 13, 17