PRACTICAL NO: 1

AIM :- Implement Linear Search to find an item in the list.

Theory :- LINEAR SEARCH

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

It is worst searching algorithm with worst case time complexity. It is a force approach. On the other hand incase of an ordered list; instead of searching the list in the sequence. A binary search is used which will start by examing 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, the algorithm returns that element found and its position is also found.

1) Unsorted :

Algorithm:

Step 1: Create an empty list and assign it to a variable.

Step 2: Accept the total no. of elements to be inserted into the list from the user say 'n'.

Step 3: User for loop for adding the elements into the list.

Step 4: Print the new list:

Step 5: Accept an element from the user that to be searched in the list.

Step 6: Use for loop in a range from '0' to the total no. of elements to search the elements from the list.

Step 7: Use if loop that the element in the list is equal to the element accepted from user

Step 8:- If the element is found then print the statement that the element is found along with the elements position.


```
Print("linear Search")
a = []
n = int(input("Enter a range:"))
for s in range(0,n):
    s = int(input("Enter a number"))
    a.append(s)
    print(a)
c = int(input("Enter a number to be searched"))
for i in range(0,n):
    if(a[i]==c):
        print("found at position",i)
        break
    else:
        print("not found")
```

Output:

```
linear search
enter search: 3
enter a number 1
[1]
enter a number 3
[1,3]
enter a number 4
[1,3,4]
enter a number to be searched 3
found at 1.
```

```
Print (" linear Search")
a = []
n = int(input("Enter a number"))
for S in range (0,n):
    S = int(input(" Enter a number")
    a = a.append
        a.sort()
        print(a)
c = int(input("Enter a number to be searched"))
for i in range (0,n):
    if (a[i] = c):
        print("found at position",i)
        break
    else:
        print("not found")
```

Output:-

linear Search

Enter a range = 4

Enter a number = 1

[1]

Enter a number = 3

[1,3]

Enter a number = 2

[1,2,3]

Enter a number = 4

[1,2,3,4]

Enter a number to be searched 1

Found at position 0.

Step 9: Use another if loop to print that the element is not found if the element which is accepted from user is not their in the list.

Step 10 :- Draw the output of the given algorithm.

Sorted Linear Search :-

Sorting means to arrange the element in increasing or decreasing order.

Algorithm :-

Step 1 :- Create Empty list and assign it to a variable.

Step 2 :- Accept total no. of elements to be inserted into the list from user, say 'n'.

Step 3 : Use for loop for using append() method to add the elements in the list.

Step 4 : Use sort() method to sort the accepted element and assign in increasing order the list then print the list.

Step 5 : Use if statement to give the range in which element is not found in given range then display " Element not found".

Step 6: Then use else statement. if element is not found in range then satisfy the given condition.

Step 7: Use for loop in range from 0 to the total no. of elements to be searched before doing this accept an search no from user using Input statement.

Step 8. Use if loop that the elements in the list is equal to the element accepted from user.

Step 9: If the element is found then print the statement that the element is found along with the element position.

Step 10: Use another if loop to print that the element is not found if the element which is accepted from user is not there in the list.

Step 11: Attach the input and output of above algorithm.

```
a[]
n=int(input("Enter the range"))
for b in range(0,n):
    b=int(input("Enter the number"))
    a.append(b)
    a.sort()
    print(a)
s=int(input("Enter the number to search"))
if(s<a[0] or s>a[n-1]):
    print("Element not found")
else:
    f=0
    l=n-1
    for i in range(0,4):
        m=int((f+l)/2)
        print m
    if(s==a[m]):
        Print("Element found at:",m)
        break
    else:
        if(s<a[m]):
            l=m-1
        else:
            f=m+1
```


PRACTICAL - 02

AIM: Implement Binary search to find a search no, in the list.

Theory: BINARY SEARCH

Binary search is also known as half interval search, logarithmic search or binary chop is a search algorithm that finds the position of a target value within a sorted array. If you are looking for the number which is at the end of the list then you need to search entire list in linear search which is time consuming. This can be avoided by using binary fashion search.

Algorithm:-

1) Create ~~Empty~~ list & assign it to a variable
2) Using input method, accept the range of given variable.
3) Use for loop, add elements in list using append() method.
4) Use sort() method to sort the accepted element & assign it is increasing ordered list. print the list after sorting.

P.E

5) Use if loop to give the range in which el... is found in given range then display a ... "Element not found".

6) Then use else statement, if statement if it ... found in range then satisfy the below condit...

7) Accept an argument & key of the element th... element has to be searched.

8 Initialize first to 0 & last to element of ... list as array is starting from 0 hence it ... Initialized 1 less than the total count.

9) Use for loop & assign the given range.

10) If statement in list & still the element to ... Searched is not found then find the middle ... element. (m)

11) Else if the item to be searched is still be... then two middle terms then.
Initialize last (m) = mid (m)-1
Else
Initialize first (f) = mid (m)-1.

12) Repeat till you found the element stick the ... & output of above algorithm.

Output:

>>> Enter the range: 3
>>> enter the number: 2
[2]
>>> enter the number: 1
[1,2]
>>> enter the number: 4
[1,2,4]
>>> Enter the number to search: 3
element not found.

40

```
print ("Bubble sort algorithm")
a = []
b = int(input ("Enter number of element:"))
for s in range (0,b):
    s= int (input ("Enter the elements"))
    a. append (s)
    print (a)
n= len (a)
for i in range (0,n):
    for j in range (n-1):
        if a[i] < a[j]:
            temp = a[j]
            a[j] = a[i]
            a[i] = temp
print ("elements after sorting are", a)
```


PRACTICAL - 3

Bubble Sort

AIM :- Implementation of bubble sort program on given list.

THEORY : Bubblesort is based on the idea of repeatedly comparing pairs of adjacent element and then swapping their position if they exist in the wrong order, this is the simplest form of sorting available. In this, we sort the given element in ascending or descending order by comparing two adjacent element at a time.

Algorithm :-

Bubble Sort algorithm starts by comparing the first two elements of an array & swapping if necessary.

- If we want to sort the elements of array in ascending order then first element is greater than second then, we need to swap the elements.
- If the first element is smaller than second then we do not swap the element.

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

5) There an 'n' elements to be sorted then process mentioned above should be repeated n-1 times to get the required result.

6) Stick the output & input of above algo. of bubble sort stepwise.

Bubble sort algorithm: 42

Enter number of elements: 3
enter the element: 8
[8]
enter the element: 9
[8,9]
enter the element: 1
[8,9,1]
elements after sorting are: [1,8,9]

mr
9/12/19

```
def quick(alist):
    help(alist, 0, len(alist)-1)
def help(alist, first, last):
    if first < last
        split = part(alist, first, last)
        help(alist, first, split-1)
        help(alist, split+1, last)
def part(alist, first, last):
    pivot = alist[first]
    l = first+2
    r = last
    done = False
while not done:
    while l<=r and alist[l]=pivot
    l = l+1
while alist[r]>=pivot and r>=1:
    r=r-1
    if r<1:
      done = True.
    else:
    t = alist[l]
    alist[l] = alist[r]
    alist[r] <t
t = alist[first]
alist[r]=t -> alist[first] = alist[r]
return r
r = int(input("Enter range for the list"))
alist[]
```


PRACTICAL: 4

Quick Sort

Aim:- Implement Quick Sort to sort the given list.

Theory: The quick sort is a recursive algorithm based on the divide and conquer technique.

Algorithm:

Step 1: Quick sort first select a value which is called pivot value, first element serve as our first pivot value, since we know that first will eventually end up as last in that list.

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

Step 3: Partitioning begins by locating two position markers - lets call them leftmark & rightmark at the beginning and end of remaining items in the list. The goal of the partition process is to move items that are on wrong side with respect to pivot value while also converging on the split point.

Step 4: We begin by incrementing leftmark until we locate a value that is greater than the P.V. We then decrement rightmark until we find value that is less than the pivot value. At this point we have discovered two items that are out of place with respect to eventual split point.

Step 5: At the point where rightmark becomes less than leftmark. We stop. The position of rightmark is now the split point.

Step 6: The pivot value can be exchanged with contents of split point and PV is now in place.

Step 7: In addition, all items to left of Split point are less than PV & all the items to the right of split point are greater the PV. The line can now be divided at split point & quick can be invoked on.

Step 8: The quickest function invokes a recursive function, quicksort helper

Step 9: Quicksort helper begins with same base case as merge sort.

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

Step 11: If it is greater than it can be partitioned & recursively sorted.

Step 12: The partition function, implement the procedure

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


```
for b in range (0, x)
    b = int (input("Enter the elements"))
    list.append
    n = len (alist)
quick (alist)
print (alist)
```

Output:

```
Enter range for the list : 4
Enter the element : 6
Enter the element : 8
Enter the element : 4
Enter the element : 9
[4, 4, 6, 9]
```

```
print "Nisha Rao"
class stack:
    global tos
    def __init__(self):
        self.l = [0,0,0,0,0,0,0]
        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
    def peek(self):
        if self.tos<0:
            print ("stack empty")
        else:
            p= self.l[self.tos]
            print ("top element =", p)
s = stack()
```


PRACTICAL - 5

AIM : Implementation of stacks using Python list.

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 file. The elements in the stack are added or removed only from one position i.e the topmost position. Thus the stack works on the LIFO (Last in First Out) principle, as the element that was inserted last will be removed first. A stack can be implemented using array as well as linked list stack has three basic operation : push, pop, peek. The operations of adding and removing the elements is known as Push & Pop element of stack.

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 then initialize to an empty list.

Step 3: Define methods push and pop under the stack.

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

Step 5: Or Else print statement as insert the element into the stack and initialize the values

Step 6: Push method used to insert element but pop method used to delete the element from the stack.

Step 7: If in pop method, value is less than 1 return the stack is empty or else delete the element from stack at topmost position.

Step 8: First condition checks whether the no: of element are zero while the second case whether tos is assigned any value. If top is not assigned any value, then this can be sure that stack is empty.

Step 9: Assign the element value in push method and print the given value is popped out.

Step 10: Attach the input and output of above algorithm.


```
>>> s.push(20)
>>> s.1
[20,0,0,0,0]
>>> s.pop()
Data = 20
>>> s.1
[0,0,0,0,0]
>>> s.push(10)
>>> s.push(20)
>>> s.push(30)
>>> s.push(40)
>>> s.push(50)
>>> s.1
[10,20,30,40,50]
```

s.peek()

06/01/2021

```
class queue:
    global r
    global f
    def __init__(self):
        self.r = 0
        self.f = 0
        self.l = [0,0,0]
def enqueue(self, data):
    n = len(self.l)
    if self.r < n:
        self.l[self.r] = data
        self.r = self.r + 1
        print("Element inserted ...", data)
    else:
        print("Queue is full")
def dequeue(self):
    n = len(self.l)
    if self.f < n
        print(self.l[self.f])
        self.l[self.f] = 0
        print("Elements deleted :- ...")
        self.f = self.f + 1
    else:
        print("Queue is empty").
```


PRACTICAL NO : 6

AIM: Implementing a queue using python list.

Theory : Queue is a linear data structure which has 2 references front and rear. Implementing a queue using python list is the simplest as the python list provides inbuilt functions to perform the ~~satisfied~~ specified operations of the queue. It is based on the principle that a new element is inserted after rear and element of queue is deleted which is at front. In simple terms, a queue can be described at a data structure based on first in first out (FIFO) principle.

Queue() : Creates a new empty queue.

Enqueue(): Insert an element at the rear of the queue and similar to that of insertion at linked using tail.

Dequeue(): Returns the element which was at the front. The front is moved to the successive element. A dequeue operation cannot remove element if the queue is empty.

Algorithm:

Step 1: Define a class Queue and assign global variable the define init() method with self assignment in init(), assign or initialize the init values with help of the self argument

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

Step 3: Use if statement that length is equal to zero then is full or else insert the element in the empty list or display that Queue element added successfully and increment by

Step 4: Define dequeue() with self argument under this use if statement that front is equal to length of list then display Q is empty or else give that front is at zero and using that delete the element from front side and increment it by 1.

Step 5: Now call the Queue() function and give the element that has to be added in the empty list by using enqueue() and print the list after adding and some for deleting and display the list after deleting the element from the list.

Output:

```
>>> Q.add(10)
    element insert .... 10
>>> Q.add(20)
    element insert ..20
>>> Q.add(3)
    element insert ...3
>>> Q.add(4)
    queue is full
>>> Q.remove
    10 element deleted.
```

Practical No: 7
Evalution of postfix expression.

Aim:- Program on Evaluation of given string by using stack in python environment.

Theory: The postfix expression is free of any parathensis further we took are priorities of the operators in the program. A given postfix expression can easily be evaluated using stacks. Reading the expression is always from left to right in Postfix.

Algorithm:-

Step 1: Define evaluate as 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 and print it.

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

Step 5: Scan the token list from left to right. If token is an operand, convert it from a string to an integer and push the value onto the 'p'

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

Step 7: Perform the Arithmetic operation. Push the result back on the 's'.

Step 8: When the input Expression has been completely processed the result is on the stack pop the 's' and return it value

Step 9: Print the result of string after the evaluation of postfix.

Step 10: Attach the output and input of above algorithm.

Output:

The evaluated value is 27.

Sakshi

```
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 add( 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):
    newnode = node(item)
    if self.s == None
      self.s = newnode
    else:
    newnode.next = self.s
    self.s = newnode
  def display (self):
```


PRACTICAL - 8

AIM: Implementation of single linked list by adding the nodes from last deposition.

Algorithm:

1] Traversing of linked list means using all the nodes in the linked list in order to perform some operation on them.

2] The entire linked list means can be acused as the first nodes of the linked list the first node of the linked list in terms of referred by the Head pointer of the linked list.

3] Thus the entire linked list can be traversed using the nodes which is returned by Head pointer.

4] Now that we know that we can traverse the entire linked list using the Head pointer we should only use it to refer the first node of list only.

5] We should not use the Head pointer to traverse entired linked list because the Head pointer . One only reference to the 1st node in the linked list, modifying the reference of the Head pointer can lead to changes which cannot revert back.

6] We may lose the reference to the 1st node in our linked list and hence cost of one list so in order to avoid value some unwanted changes to the 2nd node we will use the temporary node to traverse.

7] We will use this temporary node as a copy of the node we are currently traversing since we are making temporary node a copy of current node the datatype of the temporary node should also be node.

8] But 1st node is referred by current so we can traverse to 2nd node as next.

9] Similarly we can traverse rest of nodes in the linked list using same method by while loop.

10] One concern now is that to find terminating condition for while loop.

11] The last node in the linked list is referred to the tail of linked list. Since the last node of linked list does not have any next node the value in the next field of the last node:

12] We can refer to the last of linked list self.next.

13] Attach the coding, input and output of above algorithm.

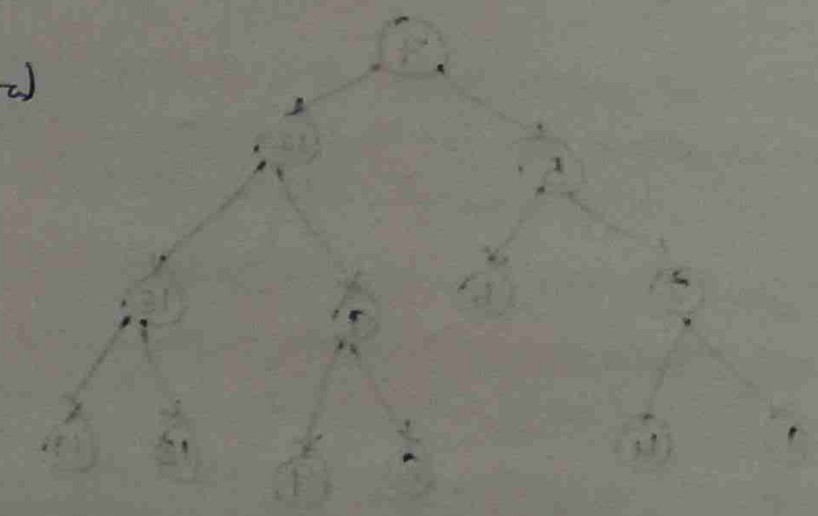
```
head = self.s
while head.next != None
    print(head.data)
    head = head.next
    Print(head.data)

start = linkedlist()
start.add(50)
start.add(60)
start.add(70)
start.add(80)
start.add(40)
start.add(30)
start.add(20)
start.display()
print("Sakshi")
```

Output:

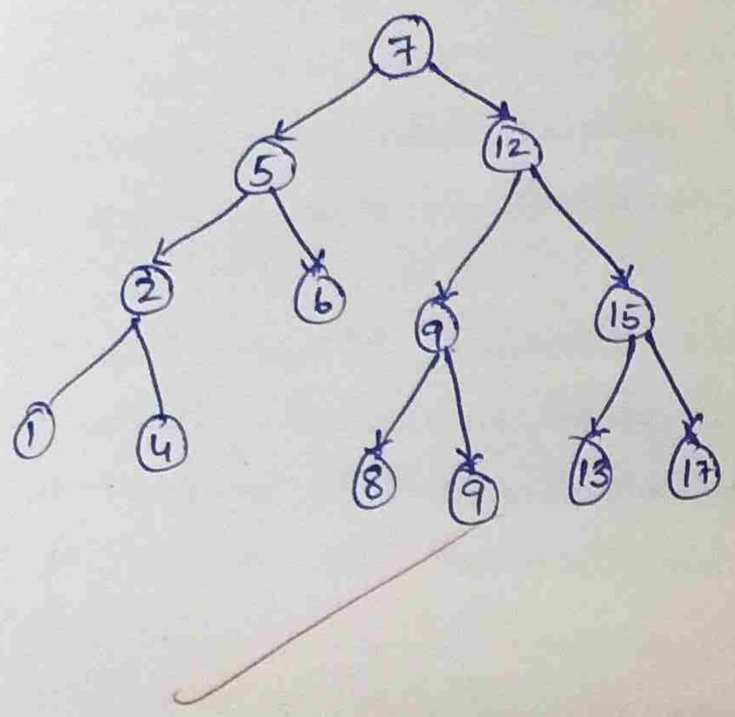
20
30
40
50
60
70
80

Sakshi



52

Binary Search Tree



Practical-9

AIM: Program based on binary search by implementing Inorder, Preorder and Postorder traversal.

Theory:

Binary tree is a tree which supports max of 2 children for any node within the Tree. Thus any particular node can have 0 or 1 or 2 children that is ordered such that child is identified as left child and other as right child.

* Inorder: i) Transverse the left subtree, the left subtree internal might have left and right subtree.
ii) Visit the root node.
iii) Transverse the right subtree and repeat it.

* Preorder: i) Visit root node.
ii) Transverse the left subtree. The left subtree intern might have left and right subtree

* Postorder: i) Traverse the left subtree. The left subtree intern might have left and right subtree.
ii) Traverse the right subtree.
iii) Visit the root node.

Algorithm:

1] Define class node and define init() method with 2 args. Initialize the value in this method.

2] Again Define a class BST that is Binary Search Tree with method with self argument and assign the root is none.

3] Define add() method for adding the node. Define a var. p that q = node(value).

4] Use if statement for checking the condition that root is none the use else statement. For if node is less than main node then put or arrange that in leftside.

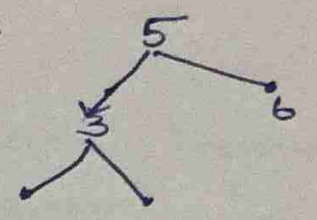
5] Use while loop for checking the node is less than or greater than the main node and break the loop if it is not satisfying.

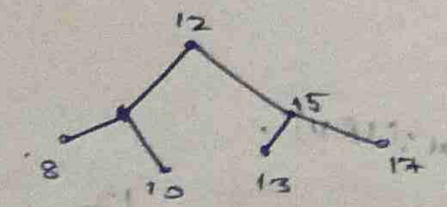
6] Use if statement within that else statement from checking that node is greater than main node side and put it into right side.

7] After this, left subtree and right subtree merge and use this method to arrange the node according to the Binary Search Tree.

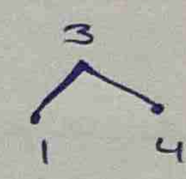
* Inorder = (LVR)

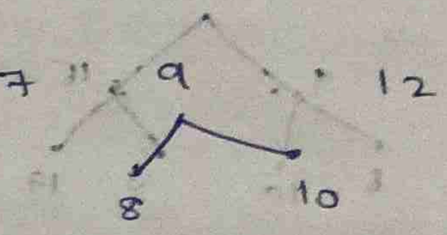
Step 1:

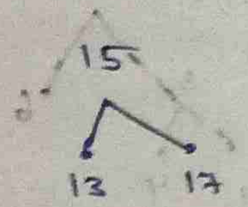




Step 2:



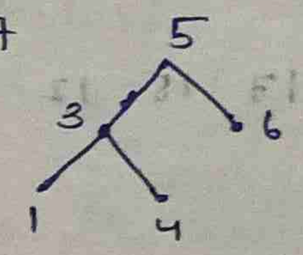


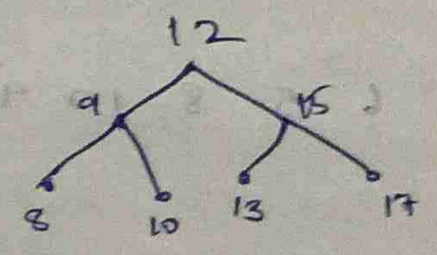


Step 3: 1 3 4 5 6 7 8 9 10 12 13 15 17

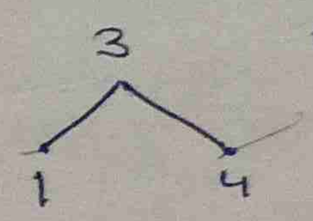
* Preorder: (VLR)

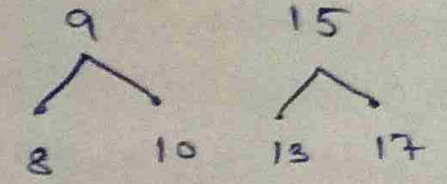
Step 1:





Step 2:





Steps: 7 5 3 1 4 6 12 9 8 10 15 13 17.

9) Define Inorder (), Preorder (), and Postorder () with root argument and use if statement that root is none and return that in all.

10) Inorder else statement use for getting that condition first left root and then right node.

11) For Preorder, we have to give condition in else that first root not, left and then right node.

12) For Postorder, In else part, assign left, the right and then go for root node.

13) Display the ~~root~~ output and input of above algorithm

& Postorder: (LRV):

Step 1: 5 12 7

Step 2: 3 6 5 9 15 12 7

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

PRACTICAL - 10

Aim:- Implementation of sets using Python.

Algorithm:-

1) Define 2 empty sets 0 and 1 and set 2 value. Use for statement providing the range of above 2 sets.
2) Now add() method used for addition the element wants to given range then print the after addition.
3) Find the union and intersection of above 2 sets by using update(), clear() method. Print the sets of union and intersection of set3.
4) Use if statement to find out the subset and superset of set 3 and set4. Display the above set.
5) Display element is set3 is not in set 4 using mathematical operation.
6) Use disjoint() to check anything is common an element is present or not. If not then display that it is mutually exclusive event.
7) Use clear() to remove or delete the sets and print the set clearing the element present in set.


```
print ("Satish 1832")
set1 = set()
set2 = set()
for i in range (2, 15):
    set1.add(5)
for i in range (1, n)
    set2. add (i)
print ("Set1 : ", set1) ;
print ("set2 :", set2) ;
print ("\n") ;
set3 = set1 | set1;
print ("Union of set1 and set2 : set3", set3),
set4 = set1 and set2
print ("Intersection of set1 and set2 : set4", set4),
print ("\n")
if set3 < set4:
    print ("set3 is support of set4")
elif set3 < set4:
    print ("set3 is same as set4")
else:
    print ("set3 is same as set4")
if set4 < set3:
    print ("set4 is subset of set3")
    print ("\n")
set5 = set3 - set4
Print ("Element in Set3 & not in set4: set5", set5),
Print ("\n")
```

```
if set4.isdisjoint(sets):
    print("set4 & set5 are mutually exclusive")
set5.clear()
print("after applying clear, set5 is empty set:")
print("set5:", set5)
```

Practical NO: 11

Aim: Program based on binary search due by implementing inorder, preorder, postorder.

Theory: Binary Tree is a tree which supports maximum of 2 children for any node within the trees. Thus, any particular node can have either 0 or 1 or 2 children

* Inorder: i) Traverse the left subtree. The left subtree return might have left and right subtree.
ii) Visit the root node.
iii) Traverse the right subtree and repeat it.

* Preorder: i) Visit the root node.
ii) Traverse the left subtree. The left subtree return might have left & right subtree.
iii) Traverse the right subtree, repeat it

* Post order: i) Traverse the left subtree. The left subtree return might have left and right subtrees.
ii) Traverse the right subtree.
iii) Visit the root node.

Algorithm:

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

Step 2: Again define a class BST that is Binary Search Tree with init () method with self argument and assign the root in none.

```
class node:
    global r
    global l
    def __init__(self, l)
        self.l = None
        self.d = a = 1
        self.r = None

class Tree:
    global root
    def __init__(self):
        if self.root = None
            self.root = node(val).
        else:
            new node = node(val)
            h = self.root
            while True:
                if newnode = "data en data":
                    if h.l != None:
                        h = h.l
                    else:
                        h.l = newnode
                        print (newnode.data, "data on left", h.data)
                        break
                else:
                    if h.r != None
                        h = h.r
                    else:
                        h.r = newnode
                        print (newnode.data " added to right", h.da
                        break
```

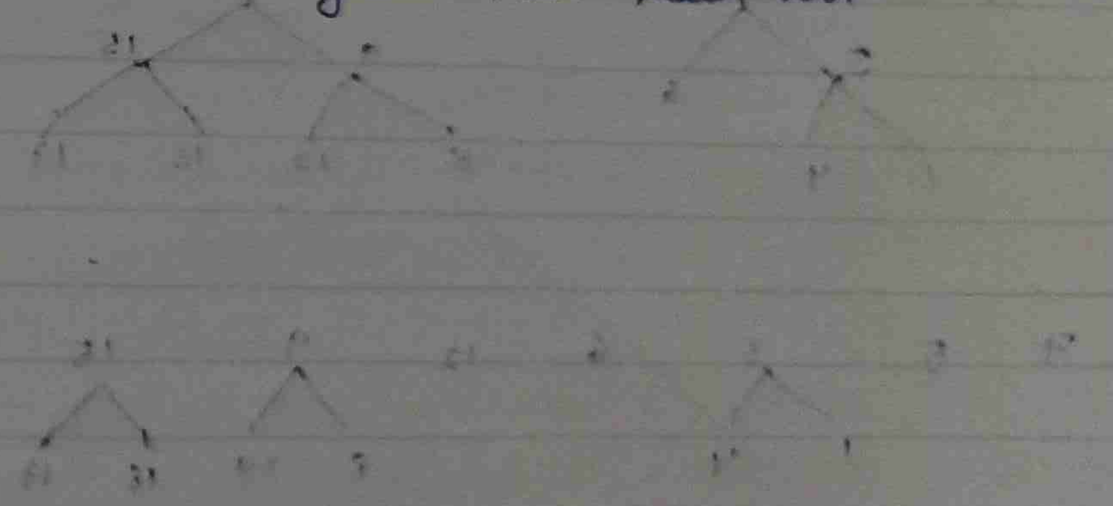
```
def preorder (self. start)
    if start = = None :
        print (start. data)
        self. preorder (start. l)
        self. preorder (start. r)
def inorder (self. start):
    if start = None;
        self. inorder (start. l)
        print (start data)
        self. inorder (start. r)
def postorder (self. start)
    if start ! = None:
        self. inorder (self. l)
        self. inorder (self. r)
        Print (start. data)
T = tree ()
T = add (150)
T = add (80)
T = add (10)
T = add (85)
T = add (60)
T = add (78)
T = add (80)
T = add (60)
T = add (12)
Print ( "preorder")
T. preorder ( T. root)
print ("inorder)
T. inorder (T. root)
print ("sakshi")
```


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

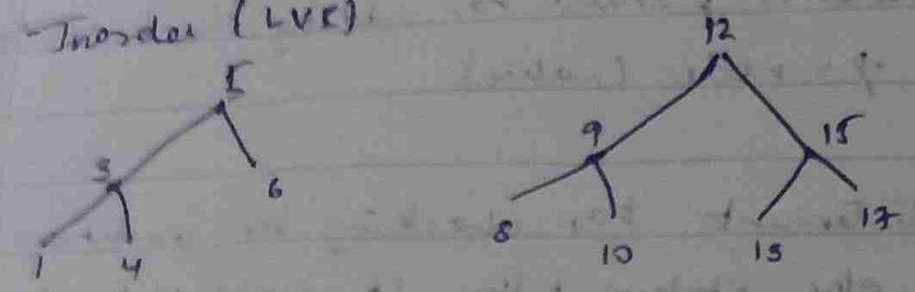
Step 4 : Use if statement for checking the condition that root is none the use else statement for if node is less than the main node then put or arrange them in left side.

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

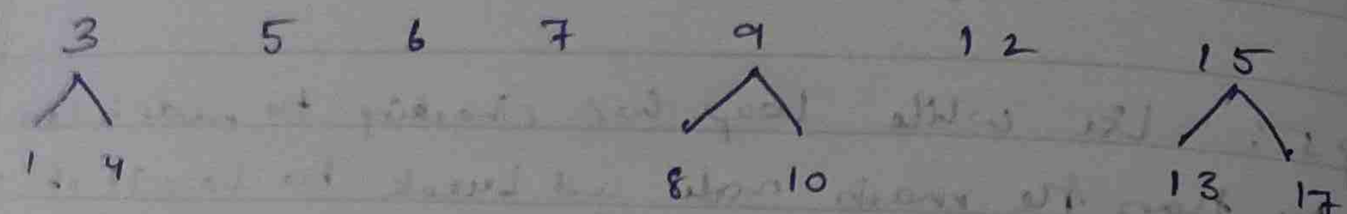
Step 6 : Use if statement within that else statement for checking that node is greater than main root.



Inorder (LVR)



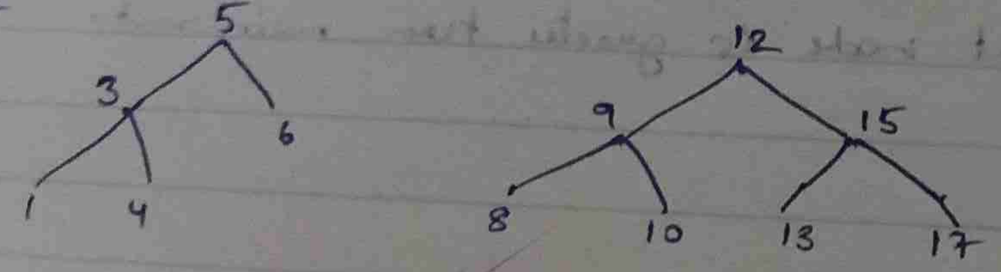
3 5 6 7 9 12 15



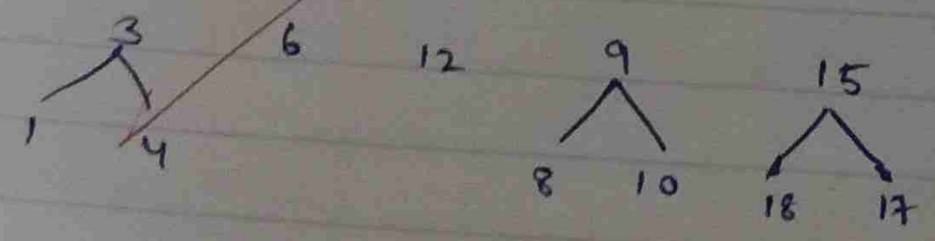
1 3 4 5 6 7 8 9 10 11 12 13 15 17

Preorder (VLR)

7 5



7 5 3 6 12 9 15



7 5 3 1 4 6 12 9 8 10 15 13 17.

Outputs :-

80 added on left of 100

70 added on left of 80

85 added on left of 70

10 added on left of 10

Preorder

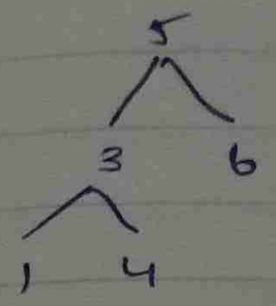
100
80
70
10
15
17
85

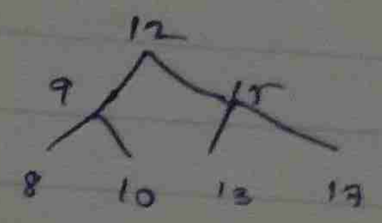
Inorder

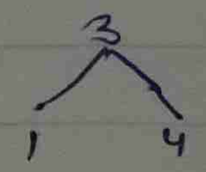
10
15
17
70
85
100

61

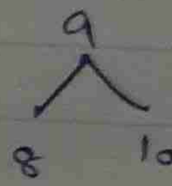
Postorder (LRV)

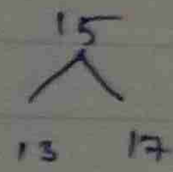




5 

6

12 



12

1 4 3 6 5 8 10 9 13 17 15

9/3/2