DS practical no 1

Aim: Linear search

1) Sorted array

algorithm

Step1:- Define a function with two parameter use for conditional statement with range ie length of array to find index

Step2:- now use if conditional statement to check whether the given number by user is equal to element in the array

Step3:- if conditional in step2 satisfies return the index no of the given

Step4:- print the new list

Step5:- accept an element from the user that to be searched in the list

Step6:- use if loop in a range from 0 to the total no of element to search the element from the list

Step7:- use if loop that element in the list is equal to the element accepted from user

Step8:- if the element is found then print the statement that the element is found along with the element position

Coding

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

inp = input(" Enter element in array :"))
array = []
for ind in inp:
    array.append(int(ind))
print ("Element in array are: ", array)
array.sort()
x1 = int(input (" Enter Element to be satisfied :"))
x2 = linear (array, x1)
if x2 == x:
    print(" Element found at position", x2)
else:
    print (" Element not found")
```

output

>>> Enter element in array 1 2 3 4 5

>>> Element in array are : [1, 2, 3, 4, 5]

>>> Enter element to be searched : 2

The element is found at position 1

coding

```
# unsorted array
def linear (arr,x):
    for i in range (len(arr)):
        if arr[i] == x:
            return i
inp = input ("enter element in array :").split()
array()
For ind in inp:
        array.append (int(ind))
print ("element in array are :", array)
x1 = int(input ("enter the element to be searched:"))
x2 = linear (array, x1)
if x == x :
    print ("Element found at location", x2)
else :
    print ("element not found")
```

output

```
>>> enter element in array: 3 2 8 5 1
>>> Element in array are : [3, 2, 8, 5, 1]
>>> Element to be searched 8
    Element found at location 2
```

step IX use another if loop to print that the element is not found if element which is accepted from user is not their in list

step X draw the output of given algorithm

3) Stored Linear search :
storing means to arrange the element in increasing or decreasing order
algorithm
step1 create empty list and assign it to a variable

step2 accept the ~~total~~ total no of search to be inserted into the list from user sarn

step 3 use for loop for using in step2 satisfies return the index no of the given array if condition doesn't satisfy then get of loop

step4 : now initialise a variable to enter element in the array form user now use split() method to split the values

step5 : now initialise a variable as array i.e empty

step6 : now use for conditional statement to append the elements given as input by user in the empty array

Step8 :- again initialise a variable to call the defined function

Step9 :- Use if conditional statement to check if variable in step 8 matches with the element you want to find then print the index corresponding to element if the conditional doesn't satisfy then print that element is not found

38

```
a= (3,7,5,6,7,8,9)
b = int (input ("Enter a number"))
n= len(a)
for b in range (0,n):
    a.append(-b)
    a.sort()
    print ("The element list is",a)
s = int(input("enter a number to be searched"))
if (s<a[0] or s>(n-1)):
    print ("Element not found")
else:
    b=0
    i=n-1
    for i in range (0,n):
        m = int (f+1)/2
        print (m)
        break
    if (s==a[m]):
        print ("found Element")
    else:
        if (s<a[m]):
            i=m+1
        else:
            b=m+1
```

Aim:- implement binary search to find an searched number 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 find the position of a target value which 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

step 1:- create Empty list and assign it to a variable

step 2:- using input method accept the range of given list

step 3:- use for loop add element in list using append () method

step 4:- use sort () method to sort the accepted element and assign it in increasing ordered list print the list after sorting

step 5:- use if loop to give the range in which element is found in given range then display a message

"Element not found"

Step6 :- Then we use else statement iib statement is not found in range then satisby the below condition

Step7 :- accept an argument and key of the element that element has to be searched

Step8 :- intialise first to 0 and last to last element ob the list as array is starting from 0 hence it is intialised 1 less than the total count

Step9 :- use for loop and assign the given range

Step10 :- if statement in list and still the element to be searched is not found then find the middle element

Step11 :- Else if the item still less than the middle item to be searched is then intalise

last n = mid (n) - 1

Else

initialise first (h) = mid (n) - 1)

Step 12 :- Repeat till you found the element stick the input and output of above algorithm

Coding

```
# bubble sort
inp = input("Enter the Element ").split()
a=[]
for ind in inp:
    a.append(int(ind))
print("Element before sort ",a)
n=len(a)
for i in range(0,n):
    for j in range(n-1):
        if a[j]<a[j]:
            temp = a[j]
            a[j] = a[i]
            a[i] = temp
print("Element after sorting ",a)
```

output

Enter the Element 7 8 2 1 3

Element ~~bbo~~ before sort [7, 8, 2, 1, 3]

Element after sort [1, 2, 3, 7, 8]

bubble sort

Aim :- implement of bubble sort program on given list

Theory :- bubble sort is based on the idea of repeatly comparing pairs of adjacent element and then swapping their position if they exists in the wrong letter & This is the simplest form of sorting available in ascending or decending order by comparing two adjacent element at a time

algorithm

Step 1 :- bubble sort algorithm starts by comparing the first two element of an array and swapping if necessary

Step 2 :- if we want to sort the element of array is ~~as~~ ascending order then first element is greater than second then we need to swap the element

Step 3 :- if we first element is smaller than second then we do not need to swap the element

Step 4 :- again second and third element are compared and swapped if it is neccessary and this process go on until last and last second last element we compared and swapped

Step 5 :- if there are n number to be sorted then the process mentioned above

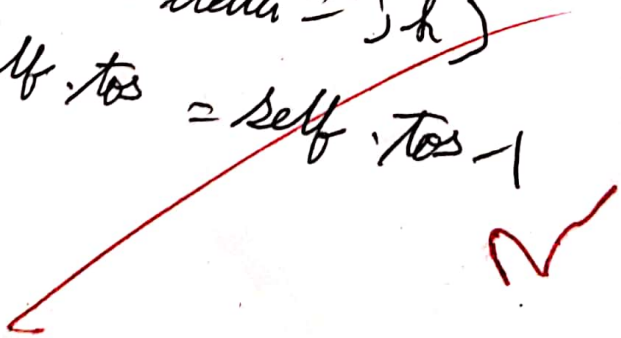
Should be repeated n-1 times to get the required result

Step 6:- stick the output and input of above algorithm of bubble sort step wise

rm
21/12/23

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("stack is full")
        else:
            self.tos = self.tos + 1
            self.l[self.tos] = data
    def pop(self):
        if self.tos < 0:
            print("stack empty")
        else:
            k = self.l[self.tos]
            print("data =",k)
            self.tos = self.tos - 1
```



aim : implementation of a stack using python list

Theory : a stack is linear data structure that can be represented in a real world from by physical stack or pile the element in the stack are the ofered the stack are the position this it was in the like principle (last in first out) it has 3 basic operator namely push pop back

algorithm

1) Create a class stack with instance variable

2) define the init main method using self argument and initialize the initial value and then initialize the to an empty list

3) define method push and pop under the class stack

4) Use if statement do give the condition that if length of given list is greater then the range of list then print stack and initialize the the value

5) push method used to insert the left pop method used to delete the element from the stack

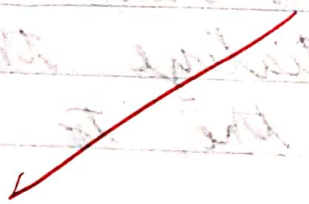
6)

7) it is non method value is to then then return is empty or else the element from stack of uppermost position

8) assign the element to values in push method to print the given value

9) attach the input and output of algorithm

10) if top is -1 not assigned any value then print that the stack is empty



output

```
>>> s.push(10)
>>> s.push(20)
>>> s.push(30)
>>> s.push(40)
>>> s.push(50)
>>> s.push(60)
>>> s.push(70)
>>> s.push(80)
>>> s.push(90)
>>> Stack is full
>>> s.pop()
    data = 80
>>> s.pop()
    data = 70
>>> s.pop()
    data = 60
>>> s.pop()
    data = 50
>>> s.pop()
    data = 40
>>> s.pop()
    data = 30
>>> s.pop()
    data = 20
>>> s.pop()
    data = 10
```

mg 09/01/2021
S. peek()

coding

```
def quicksort ( alist):
    quicksort helper( alist ,0, len( alist )-1)
def quicksort help (alist , first, last) :
    if first < last:
        splitpoint = partition (alist, first, last)
        quicksort helper (alist, first splitpoint -1 ) last)

def partition (alist first , last):
    pivotvalue = alist [ first]
    left mark = first +1
    rightmark = last
      done= false
    while not done:
        while leftmark <= rightmark and alist [leftmark]
                                    <= pivot value
            leftmark = leftmark +1
        while alist (rightmark] >= pivot value and right >=left
            rightmark < leftmark -1
        if rightmark < leftmark
                done= true
        else:-
            temp =alist [ leftmark ]
            dist [leftmark] = alist (rightmark]
            alist [rightmark ]= temp
            return rightmark
```

Topic :- implement quick sort to sort the given list

Theory :- the quick sort is a recursive the given list based on the divide and conquer technique

algorithm

step 1:- quick sort first selects a value first value element selve as out first pivot value since we knows 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 item to appropriate side of list

step 3:- positioning begins by locating two position maker lets call them left mark and right mark at the beginning and end of remaining item in the list the goal of the platform processing to more item left are on wrong side

step 4:- we begin by increasing left mark until use locate a value that is greater then the pv we then decrement right mark until we find value less than pv

step 5:- with the point where rightwork become less than leftmark we stop rightwork is now split point

step 6 :- the pv can be exchanged with the content of split point and pv is now in place

step 7 :- in addition all item let of split point are less then n.v and all the item to left to the right split point are greater then n.v

step 8 :- the quickest function involves of recursive function quick sort help

step 9 :- quick sort helper begins with same base as the merge sort

step 10 :- if length of list is less than 0 or equal one if is already sorted

step 11 :- if it is greater that it can be position and recursive function

step 12 :- the position function implement the process describe earlier

step 13 :- display and write the coding and output of above algorithm

alist = [82, 58, 95, 67, 89, 66, 55, 80, 100]
quicksort (alist)
print (alist

output

[82, 95, 58, 55, 66, 89, 67, 80, 100]

coding

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

practical no :-6

aim :- implementing a queue using python

theory :- a queue is a data structure which has 2 reference front and rear implement a queue using python list is the simplest as the python list provide inbuilt function

create :- creates a new empty queue

enqueue :- insert an element at the rear the queue and similar to that of insertion of linked using tail

Dequeue :- return the element which was at the front the font is moved to the successive element a dequeue operation cannot remove element the queue is empty

algorithm

step 1 :- define a class equeue and assign global variable then define init() method with self argument in init() assign or initialise the initial variable with the help of self argument

step 2 :- define an empty list and define enqueue method with 2 argument assign the length of empty list

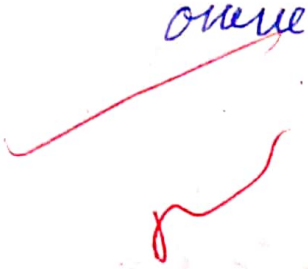
Step3 :- use if statement that length is equal to rear then queue is full or else insert the element is empty else or display that queue element added successfully and increment by 1

Step4 :- define dequeue() with self argument under this use if statement that front is empty or else give that front is at zero and using that delete the element from the front side and increment is by 1

Step5 :- now call the queue() function and give the element that has to be added in the list after adding and some first 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
```



AP coding

```
def evaluate():
    k = s.split()
    n = len(k)
    stack = []
    for i in range(n):
        if (k[i] is digit ( )):
        stack.append = (int(k[i]))
        elif k[i] == +
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) + int(a))
        elif k[i] == +('x'):
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) * int(a))
            return stack.pop()
        elif k[i] == '-'
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) - int(a))
        return.stack.pop
        s = '86g*+'
        g = evaluate(s)
print("the evaluated value is:", g))
```

output:-

> the evaluated value is 62

aim:- evaluation of given string using string i.e postfix

theory:- this postfix expression is free any parenthesis further we took end of the priorities of operation in program a given postfix expression is always from left to right

algorithm

step 1:- declare evalute 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 the string and print it

step 4:- use for loop to assign the range of string then given condition using if statement

step 5:- scan the token list from left to right if token is an operand convert if a string to integer and push the value onto then:-

step 6:- if the token is an operator it will need two operand pop the 'p' twice the first pop is second operand and the second pop is the first operand

step 7:- perform the arithmetic operation push the result back on the 'in'

step 8:- when the input expression has been completed processed the result is as the stack pop the 'p' and return the value

step 9:- attach output and input of above algorithm

coding

```
class node:
    global data
    global next
    def __init__(self, item):
        self.data = item
        self.next = none
class linked list:
    global a
    def __init__(self):
        self.s = none
    def add n (self, item):
        newnode = node(item):
        if self.s == none (item):
            self.s == newnode
        else:
            head = self.s
            while head.next != none:
                head = head.next
            head.next = newnode
    def add(self, item):
        newnode = node(item):
        if self.s == none
            self.s == newnode
        else
            newnode.next = self.ss
            self.s = newnode
def display(self):
    head = self.s
    while head.next = none:
        print(head.data)
        head = head.next
```

aim:- implementation of single linked list by adding the nodes from position

theory :- a linked list is a linear data structure which store the elements in a node the individual element of linked list called node node comprisises of 2 parts
1) data 2) next
data stores all the information with respect to the element for eg:- roll no, names, address, etc, where next refers to the next nodes

algorithms
step 1:- transversing of a linked list means visiting all nodes in linked list in order to perform some operation on them

step 2:- the entire linked list can be transvered using the node which is refered by the head pointer of the linked list

step 3:- thus the either linked list can be transfered using the node which is refered by the head pointer of the linked list

step 4:- now that we know that we can transver the entire linked list using the head pointer we should only use it to refer the first node of list only

step 5:- we should not use the head pointer to transverse the entire linked list because the head pointer is our only reference to the list node in the linked list modifying reference of head pointer can lead to change which we cannot revert back

step 6:- we may may lose the reference to the list node in out linked list and hence most of our linked list so in order to avoid making some unwanted changes to the list node we will use temporary node to transver the entire linked list

step 7:- will use this temporary node as a copy of the node we are currently transver since we are making temporary node a copy of current node the datatype of temporary node should be also be node

step 8:- now that current is refered in the first node if we want to access end node of list we can refer it as the next node of list node

step 9:- but 1st node is referred by current so we can transver to 2nd node as h = h.next

step 10:- similarly we can transverse nested node in the linked list using some method by while loop

print (head.data)

start = linked list-()

output:

start.addLC(50)
start.addLC(60)
start.addLC(70)
start.addLC(80)
start.addBC(40)
start.addBC(30)
start.addBC(20)
start.addDisplay()

20
30
40
50
60
70
80

step 11:- our concern now is to find terminating condition for the while loop

step 12:- the last node in the linked list is referred by tail of linked list since the last node of linked list does not have any next node the value in the next field of last nodes is is none

aim :- implementation of mergesort by using python

theory :- mergesort is a divide and conquere algorithm it divides input array into two halves itself for the two halves and the merge the two sorted halves the merge() function is used for merging two halves

algorithms

step 1 :- the list is divided into left and right in a recursive call until two adjacent element all often

step 2 :- now begin the sorting process the i and j iteration traverse the two halves in each call the k iteration traverse the whole lists and makes changes along the way

step 3 :- if the values at i is smaller then the value at L[i] is assigned to the arr[i+1] slot and is incr to the arr[i+1] slot and to is incre to the i chosen

step 4 :- this way the values being assigned through k[i+1] are all sorted

step 5 :- at the end of this loop one of the halves may not have been transeven completely remaining and both in the list

```
#
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[l+i]
    for j in range(0,n2):
        R[j] = arr[m+1+j]
    i = 0
    j = 0
    k = l
    while i<n1 and 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<m1:
        arr[k] = L[i]
        i = i+1
        k = k+1
    while j<n2:
        arr[k] = R[j]
        j+=1
        k = k+1

def mergesort(arr,l,r):
    if i<r:
        m = int((l+(r-1)/2)
        mergesort(arr,l,m)
        mergesort(arr,m+1,r)
        sort(arr,l,m,r)
```

Q8

```
arr = [12, 23, 34, 56, 9, 80, 98, 72]
print (arr)
  n = len (arr)
  mergesort (arr, 0, n-1)
  print (arr)
output
[12, 23, 34, 56, 78, 45, 86, 98, 72]
[12, 23, 56, 56, 72, 45, 78, 86, 98]
```

step 6:- Thus this the merge sort has been implementation

Practical 10:

Aim: implementation of sets using python

Algorithm:

Step 1: define two empty set as set1 and set2 now use for statement provide the range of above 2 sets

Step2:- now add() method is used for addition the elements according the element according to given range and print the sets for addition

Step3:- find the union and intersection of above 2 sets by using and method print the sets of union and intersection of set 3

Step4:- use if statement to find out the subset and superset of set 3 and set 4 display the above sets

Step5:- display that element in set 3 is not in set4 using mathematical operations

Step6:- use clear() to remove or delete the sets and print the set and print the set after clearing the element present in the set

```
# code :
set 1 = set()
set 2 = set()
for i in range (0, 15):
    set 1 add (i)
for i in range (1, 12):
    set 2 add (i)
print ("set 1 :", set 1)
print ("set 2 :", set 2)
print ("\n")
set 3 = set 1 | set 2
print ("union of set 1 and set 2 : set 3", set 3)
set 4 = set 1 & set 2
print ("intersection of set 1 and set 2 : set 4 ", set 4)
print ("\n")
if set 3 > set 4 :
    print ("set 3 is superset of set 4")
elif set 3 < set 4 :
    print ("set 3 is subset of set 4")
else :
    print ("set 3 is same as set 4")
if set 4 < set 3
    print ("set 4 is subset of set 3")
    print ("\n")
sets = set 3 - set 4
print ("element in set 3 and not in set 4 : sets ", sets)
print ("\n")
if set 4 . is disjoint (sets):
    print ("set 4 and sets are mutually exclusive \n")
    set. clear()
    print ("after applying clear ,sets is empty set:")
    print ("set 5", = set 5)
```

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 and set2 {8, 9, 10, 11}

set3 is superset of set4

element in set3 and not in set4 : set5

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

set4 and set5 are mutually exclusive

after applying clear set5 is empty set

set5 = set()

aim :- program based on binary search tree by implementing in order recorder and postorder transversal

theory :- binary tree is a tree which support maximum of 2 children for any node within the tree thus any particular node can have either 0 or 1 or 2 children there is another identity of binary tree that is ordered such that one child is identified as left child and other as right child

in order: 1) tranverse the left subtree the left subtree infrom might have left and right subtree

recorder: 1) visit the root node traved the left subtree and right subtree tranverse the right subtree

postorder : 1) traverse the left subtree the left subtree intern might have left and right subtree

de algorithm :-

step 1 : define class node and define init () with 2 argument initialise the value in the method

step 2 :- again define a class bst that is binary search tree with init () with self argument and assign the root is none

Step 3:- define addl() for adding the node defining a variable N that N= node (value)

Step 4: use if statement for checking the node is less then or greater than the main node and break the loop if it is not satisfying

Step 5;- use while loop for checking the node is less then 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 then main loop then put it into rightside

Step 7:- after this left side tree and right subtree to repeat this method to arrange the node according to binary search tree

Step 8:- define inorder() preorder() and postorder() with root as argument and use if statement that root is none and return that all

Step 9:- inorder else statement used for giving the condition if first left root and then right node

Step 10:- for preorder we have to give condition inside that first root left and the right side

```
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):
        n = node(value):
        if self.root == none:
            self.root = n
            print("Root is added successfully", n.val)
        else:
            h = self.root
            if n.val < h.val:
                if h.val = none
                    h.left = n
                    print(n.val, "none is added successfully")
                    break
                else:
                    h = h.left
            eseelse:
                if h.right = none
                    h.right = n
                    print(n.val, "none is added to rightside successfu
                    break
                else:
                    h = h.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
    else:
        postorder(root.left)
        postorder(root.right)
        print(root.val)
t=BST()
```

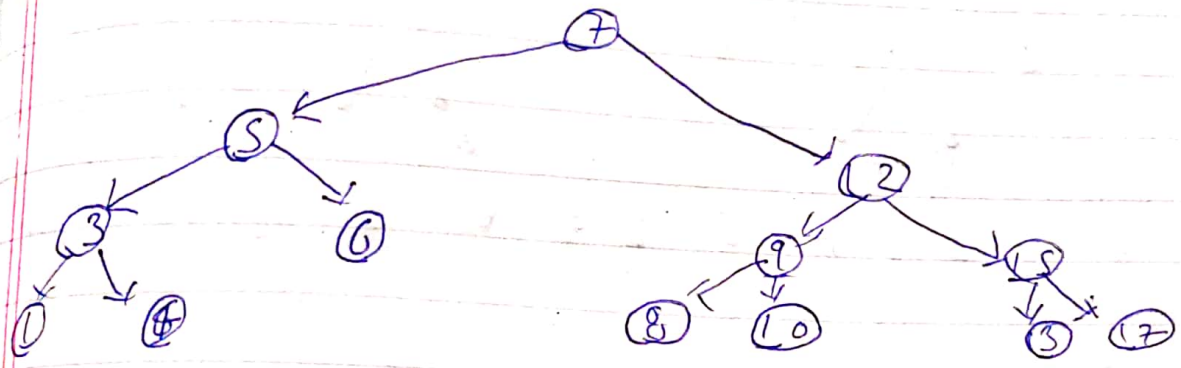
output:

```
t.add(1)
  root is added successfully
t.add(2)
  2 node is added to rightside successfully
t.add(4)
  4 node is added to rightwise successfully
t.add(5)
  5 node is added to rightside successfully
t.add(3)
  3 node is added to leftside successfully
print("\n inorder:", inorder(t.root))
  inorder:
1
2
3
4
5
inorder: None
print("\n preorder :", preorder(t.root))
  preorder:
1
2
3
4
5
preorder: None
print("\n postorder:", postorder(t.root))
3
5
4
2
1
postorder: none
```

step 11 : both root order in else part assign left and rightside

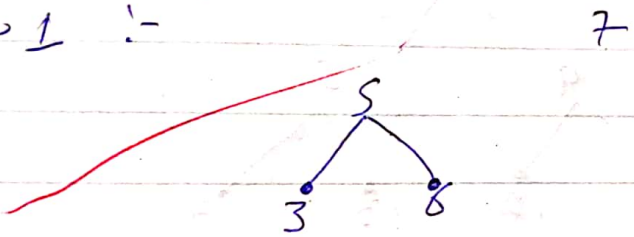
step 12: display the output and input

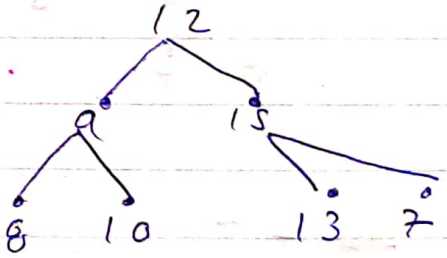
Binary Search Tree



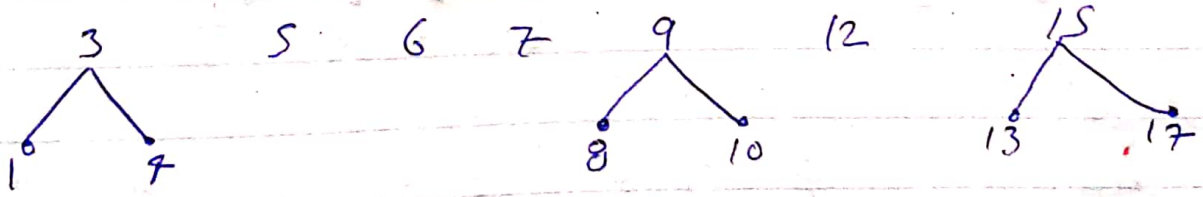
inorder : (L V R)

step 1 :-





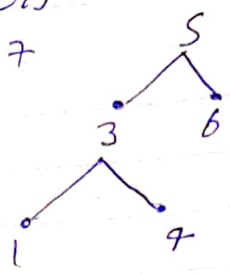
step 2 :-

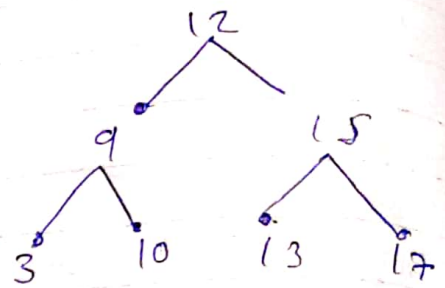


step 3 1 3 4 5 6 7 8 9 10 12 13 15 17

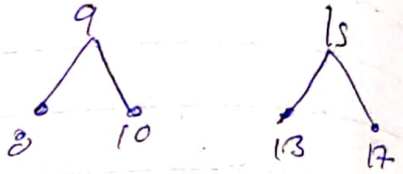
preorder (VLR)

step 1 :- 7





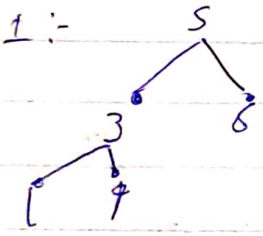
step 2: 7 5 3 6 12 9 15

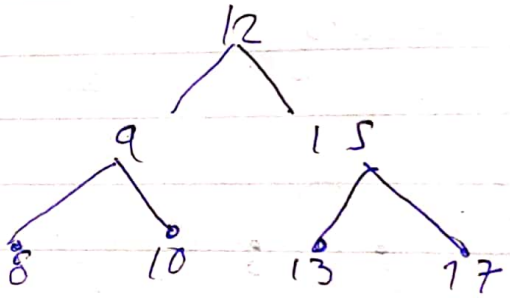


step 3 :- 7 5 3 1 4 6 12 9 8 10 15 13 17

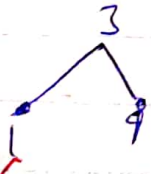
postorder: (LRV)

step 1 :-

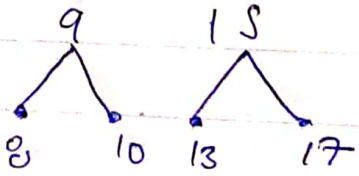




step 2 :-



6 5 9 15 12 7



step 3 :- 1 4 3 6 5 8 10 9 13 17

15 7