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

utput:-

linear search
Enter a range: 4
Enter a number: 1
[1]
Enter a number: 3
[1/3]
Enter a number to be searched
found at position 0

Practical no. 1

Aim:- Implement linear search to find an item in the list

Theory:-

linear Search

linear search is one of the impliest 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 approch. On the other hand incase an ordered list, instead of searching the 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, the algorithm return that element found and its possible is also found

No.

1. Unsorted

Algorithm:-

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

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

Step 3: Use for loop for adding the element into the list

Step 4: Print the new list

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

Step 6: Use for loop in rage from '0' to the total no. of statement to search the element from the list

Step 7: Use if loop that 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 element position


```
print ("linear search")
a=[ ]
n= int (input (" Enter a range:"))
for s in range (0,n):
    S= int (intput("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 position 1

step9: Use another if loop to print that the element is not found if the element which is accepter from user is not their in the list

step10: Draw the output of given algorithm

Sorted linear search

Sorting around mean to arrange the element is 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 element in the list

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

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

Step 6: The use else statement, if element is not found in range the satisfy the given condition

Step 7: Use for loop in range from 0 to the total number of element to be searched an search no from user using Input statement

step8: Use if loop that the element in the list is equal to the element accepted from user

step9: If the element is found then print the statement that the element is found along with the element

step10: Use another if loop to print that the element is not found if the element is not their in the list

step11: Attach the input and output of above algorithm

2/12/19

Practical no 2

Aim:

Implement Binary search to find an searched no in the list

Theory:

Binary search

Binary search is also known as f-interwalls search, logarithmic search or binary chop is a search algorithm that find 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

Create empty list and assign it to variable

using input method, accept the range of given list

use sort() method to sort the accepted element and assign it in increasing list after sorting


```
a = [ ]
n = int(input("Enter the range"))
for b in range(0,n):
    b = int(input("Enter the numbers"))
    a = append(s)
    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,n):
        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 no. 3

Aim:

Implementation of bubble sort program on given list

Theory

Bubble - Sort

Bubble sort is based on the idea of repeatedly comparing pairs of adjacent element and then swapping their position if they exist in the wrong order.

Algorithm

1. Bubble sort algorithm starts by comparing the first two element of an array and swapping if necessary
2. If we want to sort the element of array in ascending order then first element is greater than second than we need to scoap the element

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

Output

```
Bubble sort algorithm
Enter number of element: 4
Enter the elements: 3
[3]
Enter the elements: 1
[3,1]
Enter the elements: 5
[3,1,5]
Element after sorting are [1,3,5]
```

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

utput :

```
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]
```


until we find value that is less then the pivot value. At this point we have discovered two item 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 how the split point

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

Step 7:- In addition, all items to left of split point are less than PV & al the items to the right of splitpoint are greater the PV. The line can now be divided at split point & quick can be invoked on 2 h

Step 8:- The quicksort function invokes a recursive function, quicksorthelpe

Step 9:- quicksort helper begins with same base as the merge sort.

Step 10:- If the length of the list is ~~than~~ 0 or equal to one it is alr sorted

Step 11:- If it is greater then it can be partitioned & recursively so

Step 12:- The partition function, implement the process earlier

Step 13:- Display and stick the coding & output of above algori

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 pile. 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 principle as the element that was inserted last with be removed first. A stack can be implemented using array as well as linked list stack has three basic operating push, pop, peek. The operations of adding and removing the operations of addings and is known as Push & Pop

Algorithm :-

Step1 :- Create a class stack with instance variable item

Step2 :- Define the init method with self argument & initialize the initial value & then initialize to an empty list

Step3 :- Define method push and pop under the class stack

```
print "Aachal"
class stack:
    global tos
    def __init__(self):
        self.l = [0,0,0,0,0,0,0,0]
        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 empty")
        else:
            k = self.l[self.tos]
            print("data =", k)
            self.tos = self.tos - 1
stack()
    def peek(self):
        if self.tos < 0
            print("stack empty")
        else:
            p = self.l[self.tos]
            print("top element =", p)
```

06/01/20

```
S = stack()
```

Practical No. 7

Title: Implementing a Queue using python list

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

Queue(): Creates a new empty queue

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

```
class Queue:
    global r
    global y
    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()
```

Prac 8

1. Transversing of linked list mean using all the nodes in the linked list

2. The entire linked list mean can be accessed as the first nodes of the linked lst the first node of the link list in term of referene by the head

This the entire linked list can be traversed using the nodes which is refered

Now that we know that that can traverse the entire linked list using the head pointer

We should not use the head pointer to transverse entired list because the head pointer is our only refererece to 1st nodes

```
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):
  neonode = node (item)
  if self.s == Node
    self.s = neonode

else:
  head = self.s
  while head.next! = None
      head = head.next
  head.next = neonode
    neonode = node (item
    If self.s == None

else:
    neonode.next -- self.s
    self.s = neonode
 def display (self):
```

```
print("\n")
if set 4 is disjoint (set 5):
    print(" set 4 & set 5 are mutually exclusive \
set 5.clear()
print("after applying clear, set 5 is empty set:
print("set 5:", set 5)
```

Output

Enter the range : 4
Enter the number : 2
[2]
Enter the number : 8
[2,8]
Enter the number: 6
[2,8,6]
Enter the number to be searched : 4
Element found at : 1

5. Use if loop to given the range in which element is found

6. Then use else format, if statements is not found in range then satisfy

7. Accept one argument and key of the element that has to be searched

8. Use for loop and assign the given range

9. If statement in list and still the element to be searched is not found then find the middle element

11. Else if the item to be searched is still less than the middle term

12. Repeat the you find the element stick the input and output of above algorithm

2/1/21

4. Again second and third element are compared and swapped if it is necessary and this process goes on until last and second last element is compared and support

5. If there are n element to be sorted then the process mentioned above should be repeated n-1 times to get the required result

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

9/12/19

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:

Step1: Quick sort first seled 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

tep2:- 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

p 3: Partitioning begine by location two position markers let's call them leftmark & right mark at the beginning and end of remaining items in the list. The goal of the partition process is to move item that are on wrong side with respect to pivot value while also converging on the split point

+:- We begin by incrementing leftmark until we locate a value that is greater than the PV we then decrement


```
def quick(alist):
    help(alist,0,len(alist)-1)
def help(alist,first,last):
    if first<last
        split=port(alist,first,last)
        help(alist,first,split-1)
        help(alist,split+1,last)
def past(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[first]=alist[r]
    alist=[r]=t
    return r
r=lat(input("Enter rangefor the list"))
alist[]
```

Aachal Singh

```
>>> s.push(20)
>>> s.l
[20,0,0,0,0]
>>> s.pop()
Data = 20
>>> s.l
[0,0,0,0,0]

>>> s.push(10)
>>> s.push(20)
>>> s.push(30)
>>> s.push(40)
>>> s.push(50)
>>> s.l
[10,20,30,40,50]
>>>
```

s.peek()

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

5 Or else print statement as insert the element into the stack & initialize the value

6 Push method is used to insert the element pop metho is used to delete the element

7 If pop method in less than 1 then return the stack is e or else delete the element from struck at top position

8 First cond. check whether the no. of element are 0 while the second case whether to is assigned any value top is not assigned any value, then can be sure stack is empty.

9 Assign the element in push method to & print the value ~~is~~ poped not

10 Attack the Input & output of above algorithm

Practical 6.

Evalution of a Postfix Expression

- Aim:- Program on evaluation of given string by using stack in Python environment
  i.e. Postfix

- Theory: the postfix expression is free of any further ~~task~~ of the priorities of the operators in the program. A gives postfix expression can easily be evaluated using stacks. Reading the expression is always from left to right in Postfix

ep1: Define evaluate as function then create a empty stack in python

p2: Convert the string to a list by using the string method 'sput'

3: Calculate the length of string and print it

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

```
# code

def evaluate(s):
    k = s.split()
    n = length
    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)
print("The evaluate value is:",r)
```

output:

The evaluate value is: 62

Step5: Scan the token list from left to right. If token is an operand convert it from a string to an integer & push the value onto the 'p'

Step6: If the token is an operator *, /, +, -, ^, it will n two operands. pop the 'p' twice. The first pop is seco operand and the second pop is the first operands

Step7: Perform the arithmetic operation. Push the result back on the 'm'

Step 8: When the input expression has been completely processed the result is on the value

Step 9: Print the result of string after the evalua of postfix

Step 10: Attach output & input of above algorithm

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
```


Step1: Define a class queue and assign global va
& then define init() method with self argum
in init(), assign the help of self argum

Step 2: Define a empty list and define Queue
method with 2 argument assign the length o
list

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

Step 4: Define queue() with self argument
that front is equal to length of list then
queue empty or else give that front is
using that delete the elements from

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

start = linked list
start.addL(50)
start.addL(60)
start.addL(70)
start.addL(80)
start.addB(40)
start.addB(30)
start.addB(20)
start.display()
print("Sakshi")
```

Output :-

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


6. We may lose the reference to the 1st node in our linked list and hence most of our list so in order to avoid making some unwanted changes to the 2nd node we will use the temporary node to transver

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

8. But the 1st nodes is referred by current. so we can transvese to 2nd node as next

9. Similarly we can transverse rest of nodes in th linked list using same method by while loop

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

11. The last node in the linked list is refered to tail of linked list. Since the last node of linked li does not have any next node the value in the ne field of the last node

12. We can refer to the last of linked list

Practical No. 9

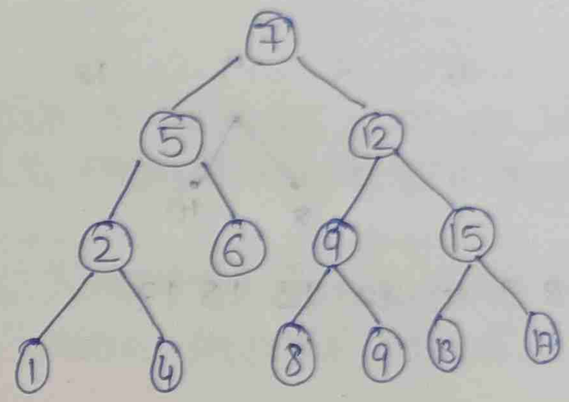
Aim: Program based on binary search by implemen
Inorders preorder and preorder Transversal

* Inorder :- 1. Transverse the left structure, the left subtree inturn might have left and right subtree
ii sort nodes
iii Transverse the right subtree & respect it

* Preorder :- 1. Visit sort node
2. Transverse the left subtree the left subtree inturn might have and right subtree
3. Transverse the right subtree repeat it

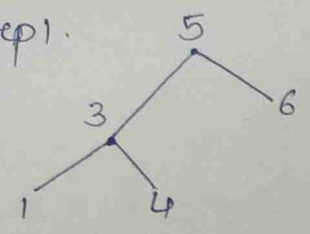
* Postorder :- 1. Tranverse the left subtree. The left subtree inture might have left and right subtree
2. Transverse the right subtree
3. Visit the root node

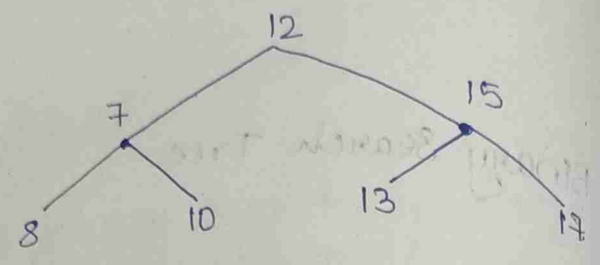
Binary Search Tree



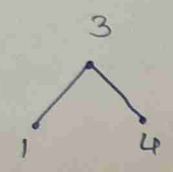
* Inorder

Step 1.

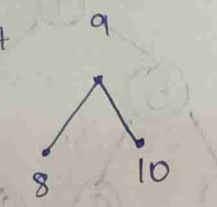


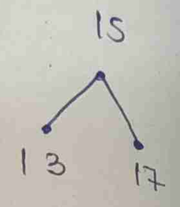


Step 2



5

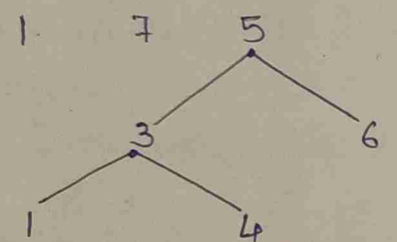


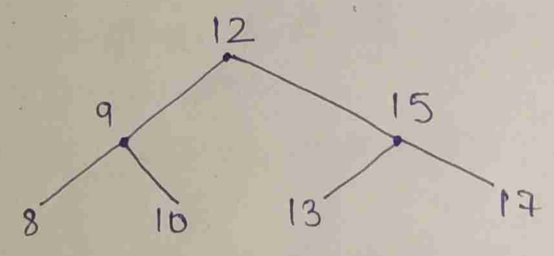


Step 3 1 3 4 5 6 7 8 9 10 12 13 15 17

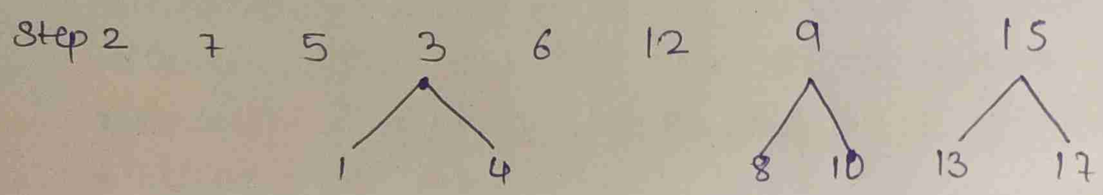
* Preorder

Step 1





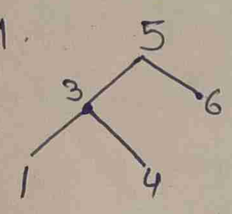
Step 2

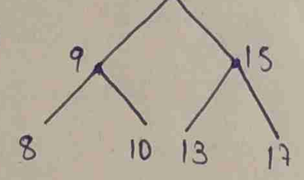


Step 3: 7 5 3 1 4 6 12 9 8 10 15 13 17

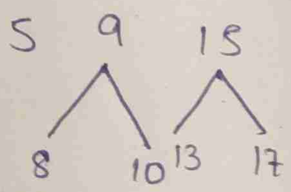
* Postorder

Step 1.





Step 2:



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

Algorithm

Define class node & define init() method with 2 argument

Again define a class BST that is binary search Tree with init() method with self argument

Define add() method for adding the node define a variable p that p= node(value)

Use if statement for checking the node is less th or greater than the main loop

Use if statement within that else statem for checking that nodes is greater

After this left subtree & right subtree merge a use this method to arrange the node accordi to the binary search Tree

For Preorder, we have to give condition in el

For preorder, In else part, assign left then and there go for root node

Display the output & input

Practical 10.

Aim: Implementation of sets using python

Algorithm

1. Define 2 empty set of 1 and set 2 now. Use for statement
2. Now add() method used for addition the elem according to given range then print the after addition
3. find the union & intersection of above 2 sets by using n(and), (or) method
4. Use if statement to find out the subset & superset of set 3 & set 4 display above set
5. Display element in set 3 is not in set y using operation
6. use disjoint() check anything is common or element is present
7. Use clear() to remove or delete the set & print the set

```
# code
print ("Aachal 1837")
set1 = set()
set 2 = set()
for i in range (8,15)
    set1.add (i)
for i in range (1,12)
    set 2.add (1)
print ("set 1:", set 1)
print ("set 2:", set 2)
print ("\n")
set 3 = set1 | set 2
print ("Union of set 1 & set 2: set 3", set 3)
set 4 = set 1 & set 2
print ("Intersection of set 1 & set 2 : set 4", set 4)
print ("\n")
if set 3 > set 4:
    print ("set 3 is support of set 4")
elif set 3 < set 4
    print ("set 3 is same as 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")
Set 5 = set 3 - set 4
print (" Element in set 3 & not in set 4: set 5", set 5
```

Practical no.11

Aim: Program based on binary search tree by implenting

Algorithm:

Define class node and define init() method with 2 argument.

Step2: Again define a class BST that is Binary search Tree with init() method

Step3: Define add() method for adding the node define a variable p that p=node(value)

Step4: Use if statement for checking the condition that is none the use else statement for if node is less than the main node then put or arrange them in left side

Step5: use while loop for checking the node is less than greater the main node & break the loop if it not satisfying

Step6: Use if statement within that else statement for checking that node is greater than main root

```
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 delete)
        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(180)
T=add(80)
T=add(10)
T=add(85)
T=add(10)
T=add(78)
T=add(60)
T=add(80)
T=add(80)
T=add(15)
T=add(12)
```

```
print("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("Aachal")
```

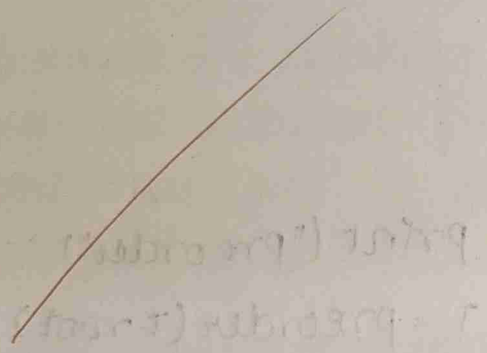
80 added OR left of 100
70 added OR left of 80
85 added OR left of 80
30 added OR left

Preorder

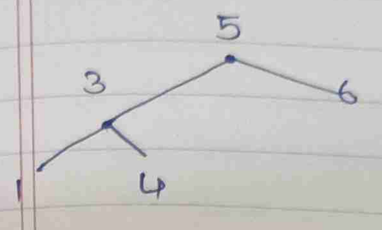
100
80
70
60
65
11
85

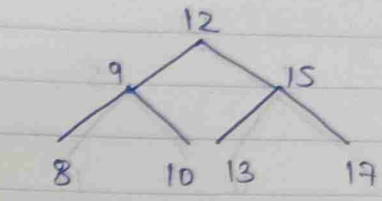
Inorder

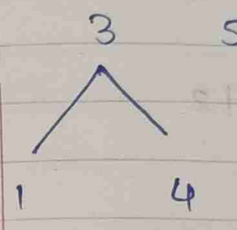
10
15
17
10
85
100

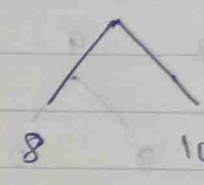


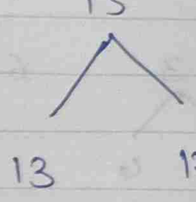
Inorder (LVR)





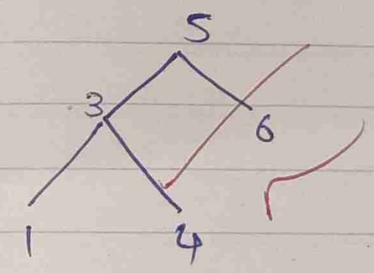


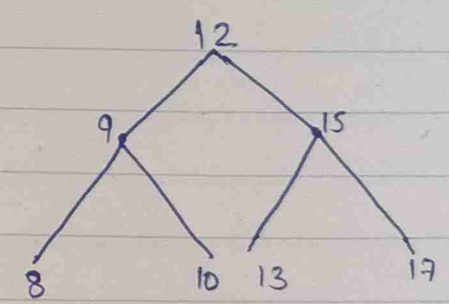


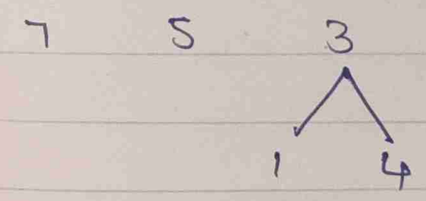


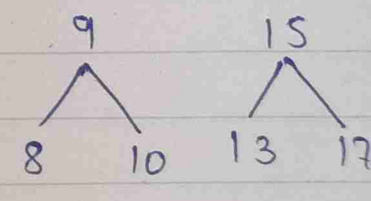
1 3 4 5 6 7 8 9 10 11 12 13 15 17

Preorder (VLR)









7 5 3 1 4 6 12 9 8 10 15