Practical - I

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

Theory:- Linear search

Linear search is one of the simplest searching algorithm 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 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 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 element to be inserted into the ~~first~~ 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 user that to be searched in the list.

Step:6 → Use for loop in a range from '0' to the total no. of element 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 element position.

Sorted:- Algorithm

Step:1 → Create empty list and assigned to a variable

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

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

Step:4 → Sort the list to sort the accepted element and assign in increasing order in the list.

Step:5 → Use if statement to give the range on which the element is found in given range then display

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 statement to be searched before doing this accept on search no from user using input statement

Step:8 → Use another if loop to point that the element is not found. ~~User~~ If the element which is

Coding

```
print("linear search")
a=[]
n=int(input("enter the range"))
for s in range(0,n):
    s=int(input("enter the no"))
    a.append(s)
    a.sort()
    print(a)
c=int(input("enter search no"))
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:5
enter a no: 4
 [4]
enter a no: 2
 [2,4]
enter a no: 1
 [1,2,4]
enter a no: 5
 [1,2,4,5]
enter a no: 6
 [1,2,4,5,6]
enter a search no: 8
not found
```

Coding :-

```
print("linear search")
a=[]
n=int(input("enter a range"))
for s in range(0,n):
    s=int(input("enter no"))
    a.append(s)
    a.sort()
    print(a)
c=int(input("enter search no:"))
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:5
enter no:6
[6]
enter no:2
[2,6]
enter no:5
[2,5,6]
enter no:3
[2,3,5,6]
enter no:1
[1,2,3,5,6]
enter search no:5
found a position:4
```

exception of num was or not there in the list.

Step 12 Draw the output of given algorithm.

Practical no: 2

Aim:- Implement Binary Search to find an searched no in the list.

Theory:- Binary Search:

Binary search is also known as Half interval search, logorithm 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 no which is at the end of the 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 for i/p method loop add, accept the range of given list.

Step:3 → Use for loop add elements 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 msg "element not found".

Step:6 → Then use else statement, if statement is not found in range

```
print("Binary Search")
a=[]
n=int(input("enter the range"))
for b in range(0,n):
    b=input("enter no:")
    a=append(b)
    a.sort()
    print(a)
s=input("Enter number to searched:")
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
```

Output

Binary Search

enter a range : 5

enter no : 3

['3']

enter no : 2

['2', '3']

enter no : 5

['2', '3', '5']

enter no : 8

['2', '3', '5', '8']

enter no : 6

['2', '3', '5', '6', '8']

enter number to searched 5

2

elements found at : 2

Step:7 → Accept a argument &key of the element that element has to be searched.

Step:8 → Initilize first to 0 and last to last element of the list_ax array is starting from 0 hence it is initialized 1 less than the total count

Step 9: Use for loop & assign the given range.

Step 10: If statement in first and still the element to be searched is not found then find the middle element (m).

Step 11:- Else if the item to be searched is still less then middle term then

initialize last (h) = mid (m) - 1

else

Initialize first (l) = mid (m) - 1

Step 12:- Repeat till you found the element while the i/p & o/p of above algorithm.

Practical-3

Bubble sort

Aim:- Implementation of Bubble sort program on given list.

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

Algorithm:-

Step:1- Bubble sort algorithm start by comparing the first two element of any array and swapping if necessary.

Step:2:- If we want to sort the element of array in ascending order then first element is greater then second then we need to swap the element:-

Step:3 → If the first element is smaller than second element then we do not swap the element

Step:4 → Again second & third element are compared and swapped if it is necessary and this

Coding

```
print("Bubble sort")
a=[]
b=int(input("enter the range"))
for s in range(0,b):
    s=input("enter no")
    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("enter after sorting=",a).
```

Output

Bubble sort

enter the range 4

enter no: 4.

[4]

enter no: 6

[4, 6]

enter no: 5

[4, 6, 5]

enter no: 1

[4, 5, 6, 1] [4, 6, 5, 1]

Sorted array [1, 4, 5, 6]

process goes on until the last and second last element is compared and swapped.

Step 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

Step 6 → Display the o/p of the above algorithm of bubble sort step 5.

Practical =4

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 eventual end up as last in that list

Step 2:→ The position 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 marker let's 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.

Coding

```
def quicksort(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+1
    r=last
    done=False
    while ~~ter and alist[l]<=pivot~~
          not done:
        while l<=r and alist[l]<=pivot
            l=l+1
        while alist[r]>=pivot and r>=l:
            r=r-1
        if r<l:
            done=True
        else:
            Temp=alist[l]
            alist[l]=alist[r]
            alist[r]=temp

    temp=alist[first]
    alist[first]=alist[r]
    alist[r]=temp
    return r
x=int(input("enter a range of list:"))
alist=[]
for b in range(0,x):
    b=int(input("Enter elements:"))
    alist.append(b)
    n=len(alist)
quicksort(alist)
print(alist
```

Output

enter a range of list : 5
enter element : 8
enter element : 3
enter element : 4
enter element : 7
enter element : 9

[2, 4, 7, 8, 9]

Step4:- We begin by incrementing leftmark until we locate a value that is greater than the pv. We then decrement rightmark until we find value that is less than the pivot value. At the 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 the content of split point p.v is now in place.

Step:7 → In addition all the items to left of split point are less than p.v & all the items to the right of split point are greater than pv. The line can be invoked recursively on the two halves.

Step:8 → The quicksort function invokes a recursive function quicksortthelper

Step:9 → quicksortthelper begins with same based as the merge sort.

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

Step11 → If it is greater their it can be partitioned and recursive sort

Step12 → The partition function implement the process described earlier.

Step:13 → Display and check the coding and o/p of above Algorithm

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 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 3 basic operation- push, pop peek. The operations of adding and removing the element is known as push & pop.

Algorithm

step:1 Create a class stack with instance variables.

step:2 Define the init method with self argument and initialize the inital value and then initialize to an empty list.

Code

```
Print("Nishant Tiwari")
class stack:
    global tos
    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 is empty")
        else:
            k = self.l[self.tos]
            print("data =", k)
            self.tos = self.tos - 1
    def peek(self):
        if self.tos < 0
            print("stack empty")
        else:
            a = self.l[self.tos]
            print("data =", a)
s.stack()
```

06/01/2020

>>> Vishal Tiwari
>>> s
[0, 0, 0, 0, 0]
>>> s.push(10)
>>> s.push(20)
>>> s.push(30)
>>> s.push(40)
>>> s
[10, 20, 30, 40, 0]
>>> s.peek()
data = 40
>>> s.pop()
data = 40
>>> s
[10, 20, 30, 0, 0]

06/01/2020

Step 3: Define method push and pop under the class stack.

Step 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.

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

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

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

Step 8 → Assign the element values in push method to and print the given value is popped.

Step 9 → Attach the input & o/p of above algorithm.

Step 10 → First condition checks whether the no of element an zero while the second case whether top is assigned any value. If top is not assigned any value then can be store that stack is empty.

Practical=6

Aim:- Implementing a Queue using python list

Theory:- Queue is a linear data structure which h 2 reference front and rear Implementing a queu using python list is simplest as python the specific operation of queue. If based on principal that new element is inserted after rear and element of queue is deleted which is at front. In simple terms a queue can be described as data structure based on first out FIFO principal.

Queue():- Create a new empty queue.

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

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

Class queue

```
    global r
    global x
    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] = data
            self.r = self.r + 1
            print("Element inserted...", data)
        else:
            print("Queue is full")
            self.n = 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().
```

O/P:

Q.add(10)

Element inserted: 10

Q.add(20)

Element inserted: 20

Q.add(30)

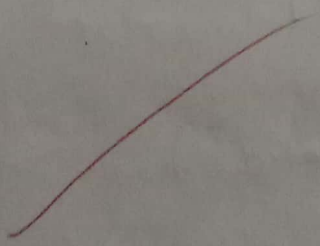
Element inserted: 30

Queue is full

Q.remove()

20

elemeted deleted:



Algorithm

Step:1 → Define a class Queue and assign global variance term define init() method the init() value with the help of self arrangement.

Step:2 → Define a empty list & define enqueue() method with 2 arrangement assign the length of empty list.

Step:3 → Use if statement that length is equal to rear than Queue is full or else insert the element in empty list or display that Queue element added automatically incremented by 1.

Step:4 → Define deQueue() with self argument value this if statement that front is equal to length of the list than display Queue is empty or else give that front is at zero and using that delete that element from front side and increment it by 1.

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

def

Practical = 7

Aim:- Program an evaluation of given string by using stack in python environment i.e. postfix

Theory:- The postfix expression is free of any parenthesis further we look case of priorities of operation in the prog. A given postfix expression can easily be evaluated using stack reading that expression is always from left to right in postfix

Algorithm

Step:1 → Define element as function the create a empty stack in python.

Step:2 → Convert the string to list by using the string method split.

Step:3 → Calculate the length of string and print it.

Step:4 → Use for loop to origin the range of string the gave condition using if statement

Step:5 → Scan the taken list from left to right taken is an integer and push the value into y.

Step:6 → If the taken is operator *, /, +, -, ^, it will need to operand pop and the 'p' turns. The first pop is the and second pop in first operand

```
def evaluate(s):
    lt = s.split(' ')
    n = len(lt)
    stack = []
    for i in range(n):
        if (lt[i].isdigit()):
            stack.append(int(lt[i]))
        elif (lt[i] == '+'):
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) + int(a))
        elif (lt[i] == '-'):
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) - int(a))
        elif (lt[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 = "8 6 a * + "
r = evaluate(s)
print("The evaluate value:", r)
```

Step:7 → Perform the arithematic operation, push result back on the m.

Step:8 → When the o/p expression has been compeletly processed, the result is on stack. pop the p and return the value.

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

Step:10 → Attach o/p and i/p of above algorithm.

Practical -8

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

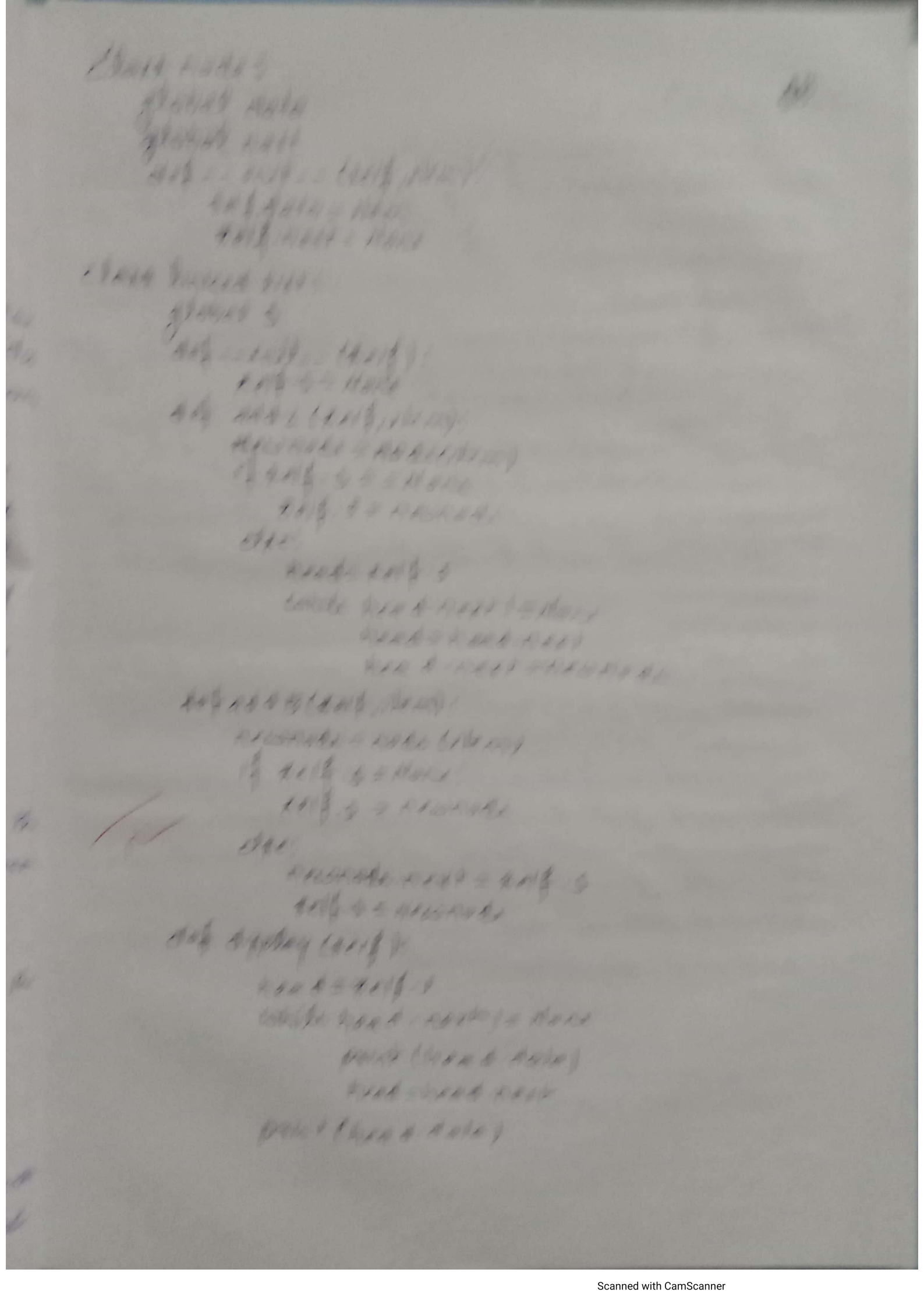
Theory: A linked list is a linear data structure which stores the elements in a node on a linear fashion but no necessarily contiguous. The individual element of the linked list called a Node. Node consist of 2 parts ① Data ② Next. Data stores all the information with the element, for example roll no, name, address, etc, whereas next refers to the next node. In case of larger list all the elements of list has to adjust itself every time we add it is very tedious task so linked list is used to solving this type of problem.

Algorithm

Step:1 → Traversing of a linked list means visiting all the nodes in the linked list in order to perform some operation on them.

Step:2 → The entire linked list can be accessed from first node of the linked list. The first node of the linked list in turn is referred by the Head pointer of the linked list

Step:3 → Thus, the entire linked list can be traversed using the node which is referred by the Head pointer of the linked list.



```
def delete(self):
    if self.s == None
        print("List is empty")
    else:
        head = self.s
    while True:
        if head.next != None:
            d = head
            head = head.next
        else:
            d.next = None
            break
s = linkedlist()
s.addL(50)
s.addL(60)
s.addL(70)
s.addL(80)
s.addB(40)
s.addB(30)
s.addB(20)
s.display
```

Output

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

Step: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.

Step:5 → We should not use the head pointer to traverse the entired linked list because the head pointer is our only reference to the 1st node in the linked list, modifying the reference of the head pointer can lead to change which we can't revert back.

Step:6 → We may lose the reference to the 1st node in our linked list and hence most of our linked list - so in order to avoid making some unwanted change to the 1st ~~linked~~ node list we will use a temporary node to traverse the entire linked list.

Step: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.

Step:8 → Now that current is referring to the first node, if we want to access 2nd node of list we can refer it as the next node of the 1st node,

Step:9 → But the 1st node is referred by current. So we can traverse to 2nd nodes as h = h.next.

Step:10 → Similary we can traverse next of n on the linked list using same method by whil loop.

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

Step:12 → The last node in the linked list is refer by 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 is None.

Step:13 → So we can refer the last node of link list as self.s = None.

Step:14 → We have to now see how to start trave the linked list & how to identify whether we h reached the last node of linked list or not

Step:15 → Attach the coding or input and outp of above algorithm.

Code

```
# Merge Sort
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+l+j]
    i = 0
    j = 0
    k = l
    while i < n1 and j < n2
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += l
        else:
            arr[k] = R[j]
            j += 1
            k += 1
        while l < n1:
            arr[k] = L[i]
            l += 1
            k += 1
        while j < n2:
            arr[k] = R[j]
            j += 1
            k += 1

def mergesort(arr, l, r):
    if l < r:
        m = int((l + (r-1))/2)
        mergesort(arr, l, m)
        mergesort(arr, m+l, r)
        sort(arr, l, m, r)
arr = [
                    ]
print(arr)
n = len(arr)
mergeshort(arr, 0, n-1)
print(arr);
```

Practical 19

Aim:- Implementation of merge sort.

Theory:- like Quicksort mergesort is a Divide & conquer algorithm. It divides input array in two halves. calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves. The merge (arr, l, m, r) is key process that assumes that arr[l, m] and arr[m+1, r] are sorted & merges the two sorted sub-arrays into one.

Algorithm

1) Define the sort (arr, l, m, r).
2) Stores the starting position of both parts in temporary values.
3) check iff first part comes to an end or not
4) checks if second part comes to an end or not
5) check which parts has smaller element
6) Now the real array has element in sorted manner including both part.
7) define the correct array in 2 parts.
8) sort the 1st part of array

g) sort the 2nd part of array

h) merge the both parts by comparing elem. of both the parts.

Output

[12, 11 13 5 6 7]

[5 6 7 11 12 13]

```
print("Nishant Rain.  1828")
set1 = set()
set2 = set()
for i in range(8,15):
    set1.add(i)
for i in range(1,12):
    set2.add(i)
print("set1:",set1)
print("set2:",set2)
print("\n")
set3 = set1 | set2
print("Union of set1 and set2: set3", set3)
set4 = set1 & set2
print("Intersection of set1 and set2: set4", set4)
print("\n")
if set3 > set4:
    print("set3 is superset of set4")
elif set3 < set4:
    print("set3 is subset of set4")
if set4 < set3:
    print("set4 is subset of set3")
    print("\n")
set5 = set3 - set4
print("Element in set3 and not in set4: set5", set5)
print("\n")
if set4.isdisjoint(set5):
    print("set4 and set5 are mutually exclusive\n")
set5.clear()
print("After applying clear, set5 is empty set: ")
print("set5 =", set5).
```

Practical :10

Topic: Implementation of sets by using python

Algorithm:

Step:1→ Define two empty set as set1 and set2 now use for statement providing the range of above 2 sets.

Step:2→ Now add() method used for adding the element according to given range then print the sets after adding

Step:3→ Find the union and intersection above 2 sets by using &(and), |(or) method. print the sets of union and intersection as set3, set2,

Step:4→ Use if statement to find out the subset and superset of set3 and set4 display the above set.

Step:5→ Display that statement in set3 is not in set4 using mathematical operation

Step:6→ Use disjoint() to check that anything is common element present or not. If not then display that it is mutually Exclusive event.

Step:7→ Use clear() to remove or delete the sets and print the set after clearing the element present in the set.

output

Nishant Tiwari

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

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

Union of set1 and set2 : set3 {1,2,3,4,5,6,7,8,9,10,11,12,13,14}

Intersection of set1 and set2 : set4 {8,9,10,11}

step 3 is superset of set4

step 4 is subset of set3

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()

Practical - 11

Aim: Program based on Binary search tree by implementing Inorder, Preorder & Postorder Transversal

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

⇒ **Inorder:** (i) Transverse the left subtree. The left subtree in turn might have left and right subtrees.
(ii) Visit the root node.
(iii) Transverse the right subtree and repeat it.

⇒ **Preorder:** (i) Visit the root node
(ii) Tranverse the leftsubtree. The left subtree in turn might have left and right subtree.
(iii) Transverse the right subtree repeat it.

Postorder: (i) Transverse the left subtree. The left subtree inturn might have left and right subtree
(ii) Tranverse the right subtrees.
(iii) Visit the root node

```
class Node
    global r
    global l
    global data
    def __init__(self,d):
        self.l = None
        self.data = d
        self.r = None
class Node
    global root
    def __init__(self)
        self.root = None
    def add(self, val):
        if self.root == None
            self.root = Node(val)
        else:
            newnode = Node(val)
            h = self.root
            while True
                if newnode.data < h.data:
                    if h.l != None
                        h = h.l
                    else:
                        h.l = newnode
                        print(newnode.data," added on left of "h.da
                        break
                else:
                    if h.r != None
                        h = h.r
                    else:
                        h.r = newnode
                        print(newnode.data," added on right of ", h.data)
                        break
```

Algorithm:

Step:1 → Define class node and define init() method wi
2 arguement. Initialize the value in this metho

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

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

Step:4 → Use if statement for checking the condition t
root is none then use else statement for if
node is less than the main node then put
or arrange that in leftside

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

Step:6 → Use if statement within that else statement for check
that node is greater than main root then put it
into rightside

Step:7 → After this, leftsubtree and rightsubtree, repeat
this method to arrange the node accordiongly
to Binary search Tree

```
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(start.l)
        self.inorder(start.r)
        print(start.data)
```

```
T = Tree()
T.add(75)
T.add(5)
T.add(12)
T.add(3)
T.add(6)
T.add(9)
T.add(15)
T.add(1)
T.add(4)
T.add(8)
T.add(10)
T.add(13)
T.add(17)
Print("peroder")
T.peroder(T.root)
Print("Inorder")
T.Inorder(T.root)
```

```
print("postorder")
T:postorder(T.root)
  print("NIL")
```

777

5 added on left of 7
12 added on left of 7
8 added on right of 5
6 added on left of 5
9 added on right of 12
15 added on right of 12
1 added on left of 12
4 added on left of 3
8 added on left of 9
10 added on right of 9
13 added on left of 15
17 added on right of 15

Preorder	Inorder	postorder
7	1	1
5	3	3
3	4	4
1	5	5
4	6	6
6	7	8
12	8	9
9	9	10
8	10	12
10	12	13
15	13	15
13	15	17
17	17	7

Step:8 → Define Inorder, Preorder() and postorder() with root arguement and use if statement that root is node and return that in all.

Step:9 → In inorder, else statement used for giving that condition first left, root and then right node.

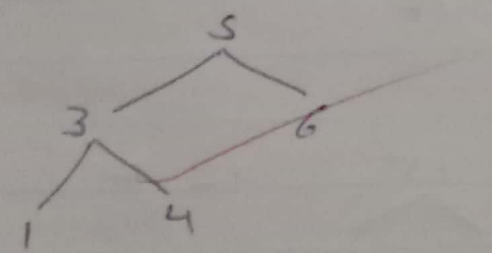
Step:10 → For preorder, we have to give condition in else that first root, left and then right node.

Step:11 → For postorder, In else part, assign left then right and then go for root node.

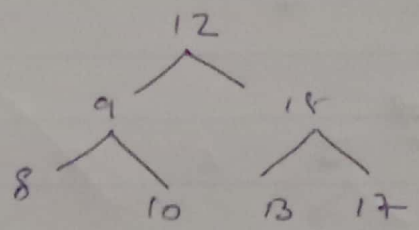
Step:12 → Display the o/p and i/p of above algorithm.

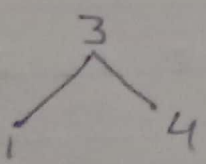
- Inorder: (LVR)

1)



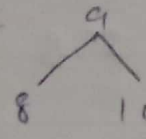
7



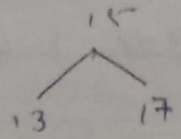


<

6 7



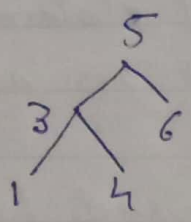
12

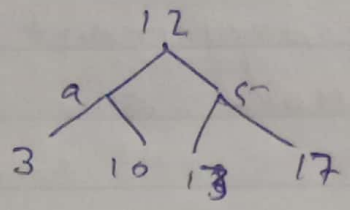


1 3 4 5 6 7 8 9 10 11 12 13 15 17

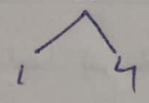
Preorder: (VLR)

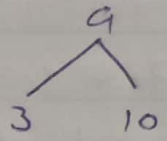
Step: 1

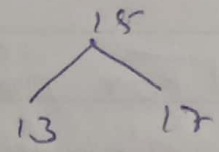




Step 2: 7 5 3 6 12



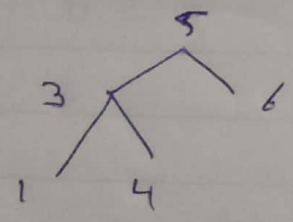


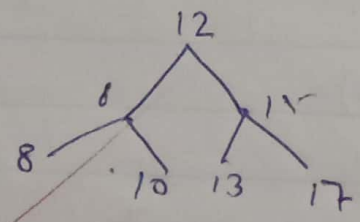


Step: 3 7 7 5 3 1 4 6 12 9 8 10 15 13 17

- Postorder (LRV)

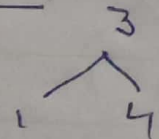
Step: 1





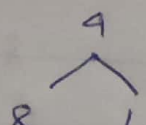
7

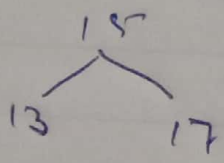
Step: 2 =>



6

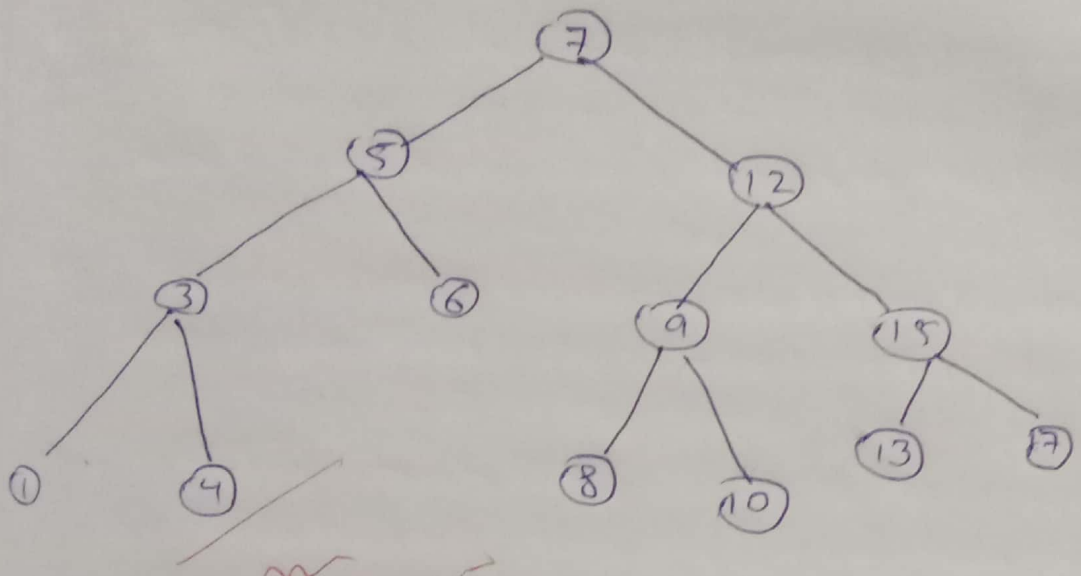
5





12 7

Step: 3 => 1 4 3 6 5 8 16 9 13 17 15 12 7



10/02/20