THAKUR COLLEGE OF SCIENCE & COMMERCE

NAAC Accredited with Grade "A" (3rd Cycle)

THAKUR TRUSTS

ISO 9001 : 2015 Certified

Degree College

Computer Journal

CERTIFICATE

SEMESTER II **UID No.** _____

Class F.Y.B.Sc(C.S) **Roll No.** 1810 **Year** 2019-20

This is to certify that the work entered in this journal is the work of Mst. / ~~Ms.~~ ~~AYUSH~~ AYOSH KUMAR MISHRA who has worked for the year 2019-20 in the Computer Laboratory.

03/1

Teacher In-Charge

Head of Department

Date : _____

Examiner

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
	Sem-2.			
1.	Implement Linear Search to find an item the list	31	29/11/19	
2.	Implement binary Search to find Searched no in the list	35	6/12/19	23/12/19
3.	Implement of bubble sort Program of given list	37	11/12/19	
4.	Implement quicksort the given Statement	39	20/12/19	
5.	Implementation of stack using python list.	41	3/1/20	03/01/20
6.	Implementing a Queue using python.	43	10/1/20	10/01/20
7.	Evaluation of Postfix Expression	46	17/1/20	17/01/20

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
8.]	Implementation of single linked list by adding the node from last Position	48.	24/01/20	
9.]	Program Based on Binary search tree by implementing Inorder, Preorder & Postorder Transversal	51	7/02/20	
10.]	To sort a list using merge sort.	56	14-02-20	
11.]	To demonstrate the use of circular queue	58	14-02-20	

Practical - 1.

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

Theory :->

Linear Search

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

It is worst searching algorithm with worst case time complexity it is a force approach. On the other hand in case of an ordered list, instead of searching the list in sequence, A binary search is used which will start by examining 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.

(i) Unsorted :->

Algorithm :->

Step1:-> Create an empty list and assign it to a variable

Step2:-> Accept the total no; of elements to be inserted into the list from the user, say 'n'.

Step3:-> Use for loop for adding the elements into the list.

Step4:-> Print the new list.

Step5:-> Accept an element from '0' to the new no. of elements to search the elements in the list.

Step6:-> Use FOR loop in a range from '0' to the new no. of elements to search the elements from list.

Step7:-> Use if loop that the elements in the list 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 desired Positions.

```
s= int (input ("Enter the requried number"))
a=[10,12,9,14,17,9]
for i in range (len)(a)):
    if (a[i]==s):
        print("Required number found in Position")
        break

if (!s) = a[i]):
    Print(" The required no. not not found")
```

Output

Enter the Required number 9

Required number found in Position 2

```
# sorted .. Program
s=sorted(list(input(" Enter the numbers:")))
s.sort()
print(s)
a=int(input("Enter the no. to be searched:"))
for i in range(len(s)):
    if (a==s[i]):
        print("number found in position", i)
        break
else:
    print("No not found")
```

Output:→

Enter the numbers: 2,5,8,9,1

[1, 2, 5, 8, 9]

Enter the no. to be searched: 5

Number Found is Position 2

Enter the numbers: 2,5,9,3,1

[1, 2, 3, 5, 9]

Enter the NO to be Searched: 7

NO. not found.

Use another if loop to point that the element is not found if the element which is accepted from use is not there in the list.

Draw the output of the given algorithm.

2.] Sorted Linear Search:

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

Algorithm :->

:-> Create Empty list and assign it to a Vari

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

-> User from loop for using append() method

Step6:- Then use else statement if element is not found in range then satisfy the given condition

Step7:-> Use for loop in range from 0 to the total no. of elements to be searched before doing this accept on search.

no. of user using input statement

Step8:-> use if ~~condition~~ loop that the elements in the list is equal to the element from user.

Step9:- If the element is found then print the statement that the element is found along with the element position

Step10:-> Use another if loop to print that the element is not found if the element which is accepted from user is not their in the list

Step11:-> attach the input and Output of above algorithm.

```
# source code :-
a = list(input("enter list: "))
a.sort()
e = len(a)
s = int(input("enter search no: "))
if (s>a[e-1] or s<a[0]):
    print("not is a list")

else:
    first + last = 0, e-1
    for i in range(0,e)
m = int('first + last)/2
print("first no at", m)
if se e, a[m]:
print("no found")
    break

else:
    if s<a[m]:
        last = m - 1

else:
        first = ~~not~~ m+1
```

Aim :-> Implement binary search to find an searched no in the list.

Theory :

Binary Search

Binary Search is also known as the half interval search Logarithmic, search as binary chop is a search algorithm that finds an position of a target value within a sorted array. If you are looking for the no. which is at the end of the the list. Then you to need to search entire list in linear search which is time assuming this can be avoided by using binary fashion search.

Algorithm :-

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

Steps 2 :-> Using input method accept the Range of given list.

Step 3 :-> Use for loop, add elements in list using append() method

Step 4 :-> Use Now sort the accepted element assign it in increasing ordered list print the list after sorting.

Step 5:-> Use if loop to given the range in which element is found in given range then display a menage elements not found.

Step 6:-> Then use use the else statement if the is not found in Range then satisfy the below condition.

Step 7:-> Accept an Argument & key of the element that element has to be sorted

Step 8:-> Initialize first to 0 and last to last element of the list as 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 list and still the element to be searched not found then find the middle element (m)

Step 11:-> else if the item to be searched is still less than the middle term then initialize last (h) = mid (m) - 1

Else

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

Step 12:-> Repeat till you found the element the input is output of above algorithm.

Output :

```
>>> Enter a range: 4
    enter a number : 2
    [2]
    enter a number: 1
    [1, 2]
    enter a number : 3
[1,2,3]
```

```
# Bubble Sort
s= list(input("Enter list of numbers: "))
for i in range(len(s)-1):
for b in range(len(s)-1-i):
    if s[b]>s[b+1]:
    d=s[b]
    s[b]=s[b+1]
    s[b+1]=d
print(s)
```

Output:-

Enter the list of no: 1,2,5,97,80,34,1017

[1,2,5,34,80,97,1017]

Practical no: 8

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

Theory:- bubble sort is based on the idea of repeatedely comparing pairs of adjacent elements and then swapping their position if they exist in the wrong order. This is the simplest form of sorting available. In this we sort the given elements in ascending or descending order by comparing two adjacent elements at a time.

Algorithm:->

Step 1:- Bubble Sort Algorithm Start by comparing the first two element of an Array and swapping if necessary.

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

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

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

Step 5:-> There are n elements to be sorted then the process mentioned above should be repeated n-1 to get required result.

Step 6:- Show the output & input of an algorithm of bubble sort stepwise.

```
def quick(alist):
    help(alist, 0, len(alist)-1)
def help(alist, first, last):
    if first<last:
        split = part(alist, first, last)
        help(alist, first, split-1)
        help(alist, split+1, last)
def part(alist, first, last):
    pivot = alist[first]
    l = first+1
    r = last
    done = false
    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:
    t = alist[l]
    alist[l] = alist[r]
    alist[r] = t
    t = alist[first]
    alist[first] = alist[r]
    alist[r] = t
    return r
x = int(input("Enter range for list:"))
alist = []
for b in range(0, x):
```

Practical no:-4

Aim:-> Implement quick sort to sort the given list.

Theory:-> The quick sort is a recursive algorithm based on the divide and Computer technique

Algorithm:->

Step 1:-> Quick sort first select a value which is called point value first element serve as over first point value. Since we know that first will eventually end up as last in that list.

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

Step 3:-> Partioning begins by ~~locating~~ locating two positions marks left's call them leftmark & right mark at the beginning and end of the remaining items in the list. The goal of the Partition process is the move items that are on wrong side with respect to point value which also converging on the split point.

Step 4:-> we begin by incrementing leftmark until we locate a value that is greater than p.v we than decrement rightmark until be find value that less than pivot value at this point we have discovered two items that are out of place with

respect to essential 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 contents of split point and p.v (pivot value) is now in place.

Step 7:-> In Addition all the items to left of split point are less than pv and the items to the right of split point are greater than p.v the list can now be divided at split point & quick so can be invoked ~~Respect~~ recursively on the two halves.

Step 8:-> The quicksort function invoke a recursive function quicksorthelp.

Step 9:-> Quicksorthelper begins with same base as the merge sort.

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

Step 11:-> If it is greater then it can be partitioned recursively sorted

Step 12:-> The Partition function implement the process ~~Step~~ desired earlier.

Step 13] display and trace the coding and Output of above algorithm.

```
b= int (input ("Enter element :"))
alist. append (b)
n= len (alist)
quick (alist)
print (alist)
```

Output:-

```
Enter range for list : 5
    Enter element 4
    Enter element 3
    Enter element 2
    Enter element 1
    Enter element 8
[1, 2, 3, 4, 8]
```

23/12/17

Code:

```
print ("Nitin Shah")
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.l[self.tos] = 0
            self.tos = self.tos - 1

    def peek(self):
        if self.tos < 0:
            print ("Stack empty")
        else:
            s = self.l[self.tos]
            print ("data", s)

s = stack()
```

Aim :->

Theory :
can b
of a
in the
one
the
out)
last
imple
list
peek
the e

Algor
Step
V

Step

Step

Step

Practical No. 5

Aim :-> Implementation of stack 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 (Last In First Out) principle as the element that was inserted last will be removed first. A stack can be implemented using Array as well as linked list. Stack has three basic operation Push, pop, peek, The operations of adding and removing the elements is known as Push Pop.

Algorithms :->

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

Step 2 :-> Define the init method with self Argument and initialize the intial value and then initialize to an empty list.

Step 3 :-> Define methods push and Pop under the class Stack.

Step 4 :-> Use if Statement to given the condition that if length of given list is greater than the range of list then print Stack is full.

Step5:-> Or else print Statement as insert the element into the stack and initialize the value.

Step6:-> Push Method used to insert the element but Pop method used to delete an element from the stack.

Step7:-> If in pop method, value in less than then returns the stack is empty or else delet the element from stack at top position.

Step8:-> First condition checks whether the no. elements are zero while the second case which to p is assigned any value → If top is not assigned any value, then can be sure that stack is Empty

Step9:-> Assign the element value in push method to add and print the given value poped

Step10:-> Attach the Output of the and the Input of above algorithm.

Output:
Nith
Dat
>>> 5
[10,
>>> 5
>>>5
>>>5
[10
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
[

Output:

Nithin Stack

Data = 50

```
>>> s.d
[10, 20, 30, 40, 50]
>>> s.pop()
>>> s.pop()
>>> s.d
[10, 20, 30, 0, 0]
>>> s.pop()
>>> s.pop()
>>> s.pop()
>>> s.d
[0, 0, 0, 0, 0]

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

s.peek()

03/01/24

Code →

```
class Queue:
    global r
    global f
    global a
    def __init__(self):
        self.f=0
        self.r=0
        self.a=[0,0,0,0,0]
    def enqueue(self, value,):
        self.n=len(self.a)
        if(self.r==self.n):
            print("queue is full")
        else:
            self.a[self.r]=value
            self.r+=1
            print("queue element inserted",value)
    def dequeue(self):
        if(self.f==len(self.a)):
            print("queue is empty").
        else:
            value=self.a[self.f]
            self.a[self.f]=0
            print("queue element is deleted",value)
            self.f+=1
b=Queue()
```

Practical - 06

* Aim:-> Implementing a Queue using python list.

Theory:-> Queue is a linear data structure which has 2 reference front and rear implementing a queue using python list is the simplest as the python list provide inbuilt functions to perform the specified operations of the queue it is based on the ~~prin~~ principle that a new element is inserted after rear and element of queue is deleted which is at front. In simple terms ; a queue can be described as a data structure based on ~~Fi~~ First in First out FIFO principle.

- Queue ( ): a new empty queue.
- Enqueue ( ): Insert an element at the rear of the queue and similar to that of insertion of linked with tail.
- Dequeue ( ): Returns the element which was at the fro the front is moved to the successive eleme
  A. dequeue operation cannot remove eleme if the queue is empty

(*) Algorithm: →

Step 1: → Define a class Queue and assign variables then define init() method with argument in init(), assign or initialize the value with the help of self argument.

Step 2: → Define a empty list and define enq() method with 2 argument, assign the len() of empty list.

Step 3: → use IF statement that length is equ to rear then Queue is full or else insert the element in empty list or display Queue. element added successfully and increment by 1.

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

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

Output:

>>> b.enqueue(2)
insert = 2

>>> b.enQueue(4)
insert = 4

>>> b.enQueue(7)
insert = 7

>>> b.q
[2, 4, 7]

>>> b.deQueue()
Queue element deleted = 2

>>> b.dequeue()
Queue element deleted = 4

>>> b.dequeue()
Queue element delet = 7

>>> b.dequeue()
Queue is empty

>>> b.q
[0, 0, 0]

>>> b.enqueue(8)
Queue is full.

10/01/2021

Practical 07

Aim:→ Evaluation of Postfix Expression
Program on Evaluation of given string by using Stack in python environment i.e., Postfix

Theory:→ The Postfix Expression is free of any Parenthesis. Further we took care of the Priorities of the operators in the program. A given Postfix Expression can easily be evaluated using Stacks. Reading the expression is always from left to Right in Postfix.

Algorithm:→

Step1:→ Define evaluate as function then create a empty stack in python

Step2:→ Convert the string to a list by using the String method 'split'

Step3:→ Calculate the length of string and Print it

Step4:→ Use for loop to assign the Range of String then give condition using if Statement.

#=code →

```
def evaluate(s):
    k = s.split()
    n = len(k)
    stack = []
for i in range(n):
    if k[i].isdigit():
        stack.append(int(k[i]))
    elif k[i] == '+':
        a = stack.pop()
        b = stack.pop()
        stack.append(int(b) + int(b))
    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 = '8 6 9 * +'
r = evaluate(s)
print("the evaluated value is", r)
print("Fiesto")
```

Output →
('the evaluated is ', 62/5)
Ties10.

Scanned with CamScanner

step 5:→ Scan the token list from left to right if token is an operand, convert it from a string to an integer and push the value onto the 'S'

step 6:→ If the token is an Operator *, /, +, -, it will need two operands. Pop the 'S' twice. The first pop is second operand and the second pop is the first operand.

step 7:→ Perform the Arithmetic Operations. Push the result back on the 'S'.

step 8:→ When the input expression has been completely processed the result is on the stack. Pop the 'S' and return the value.

Step 9:→ Print the result of stacking after the evaluation of Postfix

Step 10:→ Attach output and input 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 in a linear fashion but no necessarily contiguous. The individual element of the linked list is called Node. Node comprises of 2 parts.
1.] Data 2.] Next. Data store all the information about the element for ex- roll no, name, address, etc. whereas next refers to the next node. In case of larger list, we add/remove any element from the 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 problems.

Algorithm:->

Step1:-> Transversing of a linked list means all the nodes in the linked list in order to perform same operation on them.

Step2:-> The entire linked list means can be accessed with the first node of the linked list

Step3:-> Thus the entire linked list can be transversed using the node which is referred by the head pointer of the linked list

```
class node:
    global data
    global next
    def __init__(self,item):
        self.data = item
        self.next = None

class linked list
    global s
    def __init__(self)
        self.s = None
    def addL(self, item):
        newnode = node(item)
        if self.s = None
            self.s = newnode
        else:
            head = self.s
        while head.next != None
            head = head.next
    def addB(self, item):
        newnode = node(item):-
        if self.s = = None:
            self.s = Newnode
        else:
            newnode.next = self.s
            self.s = newnode
    def display(self):
        head = self.s
        while head.nent != None:
            print(head.data)
```

Output ->

```
>>> start.addL(50)
>>> start.addL(60)
>>> start.addL(70)
>>> start.addL(80)
>>> start.addB(40)
>>> start.addB(30)
>>> start.addB(20)
>>> start.display()

>>> 20
    30
    40
    50
    60
    70
    80

>>> print("Ayush Mishra")
Ayush Mishra
```

step4:- Now that we know that we can transverse the entire linked list using the head pointer, we should only use it to refer the first node of the list only.

step5:- we should not use the head pointer to transverse 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 changes which we cannot revert back.

step6:- we may lose the reference of the 1st node in our linked list and hence most of our linked list so in order to avoid making same unwanted changes to the 1st node, we will use temporary node to transverse the entire linked list.

step7:- we will use temporary node as a copy of the node we are currently transversing. Since we are making temparary node a copy of current node the datatype of temparary node should also be node.

step8:- Now that current is referring to the first node if we want to access 2nd node of list we can refer it as a the next node of 1st node.

Step9:-> But 1st node is preserved by current we can transverse the 2nd node as h = h.next

Step10 -> ~~But 1st node~~ Similarly we can transverse rest of node in the linked list using same logic by while loop.

Step11 -> Our concern is to find terminating condition by the while loop.

Step12 -> The last node in the linked list is referred by the tail of linked list. Since the last node of linked does not have any next node.

Step13 -> So we can refer to the last node of linked list self.s = None.

Step14 -> We have to see now how to start transverse the linked list & how to identify the last node of linked list.

Practical - 9

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 as one child is identified as left child and other as Right child.

Inorder : (i) Transverse the left subtree. The left subtreesum might have left and Right subtree.
(ii) Visit Root node.
(iii) Transverse the right subtree and repeat it.

Preorder: (i) Visit the Root node.
(ii) Transverse the left subtree. The left sub tree in turn might have ~~naigh~~ left & Right subtrees.
(iii) Transverse the Right subtree

Algorithm:→

Step1:→ Define class node and define init() method with 2 Argument initalize the value in this Method

Step2:→ Again, Define a class BST that is Binary Search tree. with init() method with self argument and assign the root is None

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

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

    def add(self, value)
        p = node(value)
        if self.root == None:
            self.root = p
            print("Root is added Successfully
                                        p.val
        else:
            h = self.root
            while True:
                if p.val < h.val
                    h = left == None:
                        h = left = p
                        print(p.val, "Node is added
                            to leftside succesfully
                            at", p.val)
                        break
                    else:
                        h = h.left

            else:
                if h.right == None:
                    h.right = p
                    print(p.val, "Node is added
                        to
```

```
        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)
```

Steps:- define add() method for adding the define a variable p that p= node(val

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

Step5:- Use while loop for checking the node is less than or greater than the main node or break the loop if it is not satisfying

Step6:- use if statement with that else statem for checking that node is greater than main root then put it into right side

Step7:- After this, left subtree and right subtree, Ho this method to average arrange the node according to Binary search tree.

Step8:- Define Inorder(), Preorder() and post() with Root argument and use if statement th root is none and return that i now.

Step9:- In Inorder, else statement used for giving that condition first left, root and then Right node.

Step10:- For preorder we have to gave

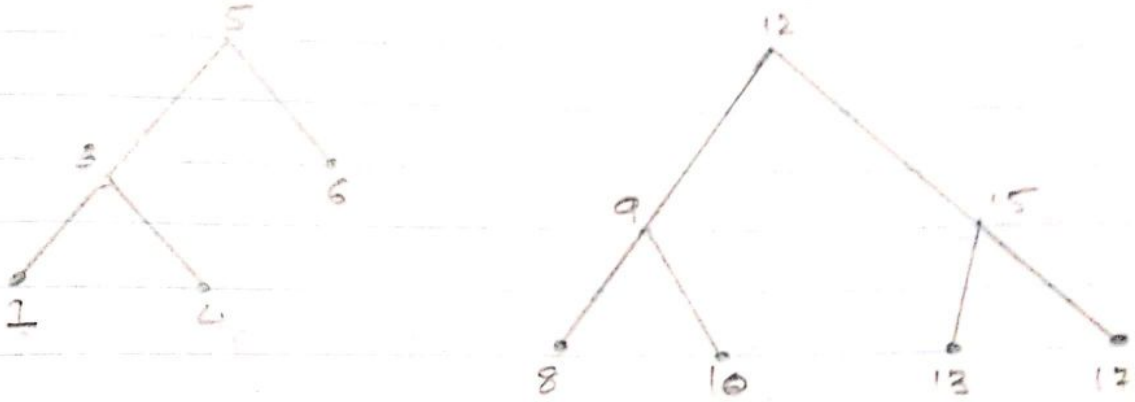
condition in else that first root, left and then right node.

Step 11 :-> For postorder, In else post assign left then right and then go for root node.

Step 12 :-> Display the output and input of above algorithm.

(*) In order : (LVR)

Step 1 :->

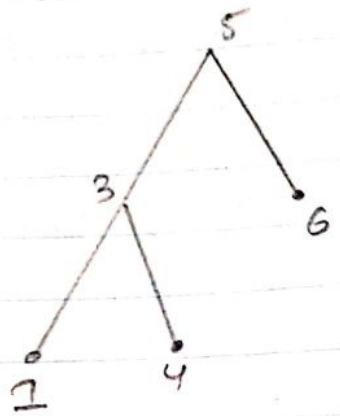


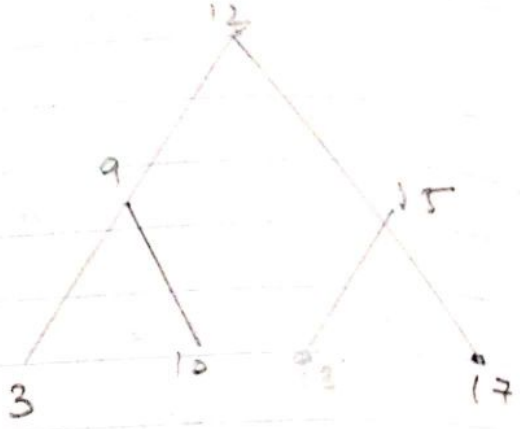
Step 2 :->

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

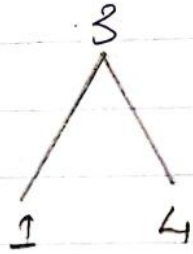
Preorder :-> (VLR)

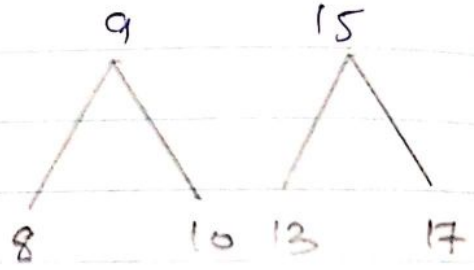
Step1 :->





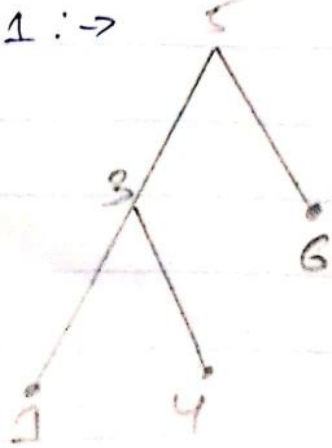
Step2 :->

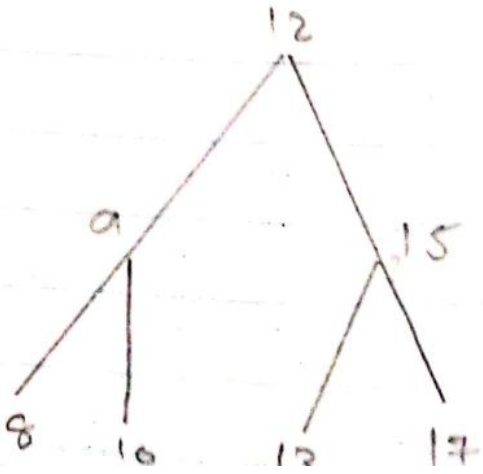
7 5 

6 12 

Step3 :-> 7 5 3 1 4 6 12 9 8 10 15
13 17.

Postorder :-> (LRV)

Step 1 :-> 



Step 2:-

Step 3: ->

1 4 3 6 5 8 10 9 13 14 15 12 7

Output →

```
>>> t = BST()
>>> t.add(7)
('Root is added successfully', 7)
>>> t.add(5)
('Root is added successfully', 5)
>>> t.add(12)
('Root is added successfully', 12)
>>> t.add(3)
('Root is added successfully', 3)
>>> t.add(6)
('Root is added successfully', 6)
>>> t.add(9)
('Root is added successfully', 9)
>>> t.add(15)
('Root is added successfully', 15)
>>> t.add(1)
('Root is added successfully', 1)
>>> t.add(4)
('Root is added successfully', 4)
>>> t.add(8)
('Root is added successfully', 8)
>>> t.add(10)
('Root is added successfully', 10)
>>> t.add(13)
('Root is added successfully', 13)
>>> t.add(17)
('Root is added successfully', 17)
```

=> Inorder (Left, Root, Right)

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

=> Preorder (Root, Left, Right)

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

=> Postorder (Left, Right, Root)

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

Practical 10

Aim:- To sort a list using merge sort

Theory:- like Quicksort, Merge sort is a divide and 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() function is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. The array is recursively divided in two halves till size becomes 1. Once the size becomes 1, the merge process comes into action and starts merging arrays back till the complete array is merged.

Applications:

- Merge sort is useful for sorting linked lists in O(nlogn) time. Merge sort accesses data sequentially and the need of random access is low.
2.) Inversion Count Problem.
3.) Used in External Sorting.

Merge sort is more efficient than quick sort for some types of lists if the data to be sorted can only be efficiently accessed sequentially and is thus is popular in languages like Lisp, where sequentially accessed data structure are very common.

```
def mergesort(arr):
    if len(arr)>1:
        mid = len(arr)//2
        lefthalf = arr[:mid]
        righthalf = arr[mid:]
        mergesort(lefthalf)
        mergesort(righthalf)
        i = j = k = 0
        while i<len(lefthalf) and j<len(righthalf):
            if lefthalf[i]<righthalf[j]:
                arr[k] = lefthalf[i]
                i = i+1
            else:
                arr[k] = righthalf[j]
                j = j+1
            k = k+1
        while i<len(lefthalf):
            arr[k] = lefthalf[i]
            i = i+1
            k = k+1
arr = [27, 89, 70, 55, 62, 99, 45, 14, 10]
print("RANDOM LIST:", arr)
mergesort(arr)
print("\n MERGESORTED LIST:", arr)
```

OUTPUT:->

>>>

RANDOM LIST : [27, 99, 70, 55, 62, 09, 47, 14,

MERGESORTED LIST: [10, 14, 27, 47, 55, 62,

>>>

RANDOM LIST: [27 62 0;

Practical 11

Class Queue

Aim:- To demonstrate the use of circular queue.

Theory:- In a linear queue, once the queue is completely full, it's not possible to insert more elements. Even if we dequeue the whole queue to remove some of the elements in order to be able to insert new elements. When we dequeue any element to remove it from the queue, we are actually moving the ~~rear~~ front of the queue, forward, thereby reducing the overall size of the queue. And we cannot insert new elements, because the rear end is still at the end of the queue. The only way is to reset the linear queue for a fresh start.

Circular Queue is also a linear data structure which follows the principle of FIFO, but instead of ending the queue at the last position, it again starts from the first position after the last. Hence making the queue behave like a circular data structure. In case of a circular queue head pointer will always point to the front of the queue and tail pointer will always point to the end of the queue. Initially, the head and tail pointer will be pointing to the same location this would mean that the queue is empty. New data is always added to the location pointed by the tail pointer, and once

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

    def remove(self):
        n=len(self.l)
        if(self.f<=n-1):
            print("Data Removed:",self.l[self.f])
            self.l[self.f]=0
            self.f=self.f+1

        else:
            s=self.f
            self.f=0
            if(self.f<self.r):
                print(self.l[self.f])
```

```
        self.r = self.r + 1
    else:
        print ("Queue is empty :")
        self.f = 5

q = Queue()
```

Output:->

```
>>> q = Queue()
>>> q.add(44)
('Data added: ', 44)
>>> q.add(57)
('Data added: ', 57)

>>> q.remove()
('data removed: ', 44)
```

the data is added rail pointer is incremented to point to the next available location

Applications: Below are have some common real-world examples where circular queue is used

1.] Computer controlled traffic signal system uses circular queue

2.] CPU scheduling Memory Management

12/02/2020