THAKUR COLLEGE OF SCIENCE & COMMERCE

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

THAKUR TRUSTS

ISO 9001 : 2015 Certified

Degree College

Computer Journal

CERTIFICATE

SEMESTER I UID No. _____

Class FY CS Roll No. 1813 Year 2019 -20.

This is to certify that the work entered in this journal is the work of Mst. / Ms. Ashish. D. Thakur who has worked for the year 2019-20. in the Computer Laboratory.

Teacher In-Charge

Date : _____

Head of Department

Examiner

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
	DS			
1.	Implement linear search to find an element item in the list	32	29/11/19	
2	Implement binary search to find on searched no. in the list	36	6-12-19	
3.	Implemention of Bubble sort program on given list	39	20-12-19	
4	Implement Quick short to Sort the given list	41	20-12-19	
5.	Implementation of stack using python list.	44	03/01/20	
6	Implement a queue using python list.		10/01/20	
7.	Evaluation of Postfix Expn	48	17/01/20	

Practical no. 1.

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

Theory:-

Linear Search.

Linear search is one of the simpliest searching algorithmd in which integreted 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 examing the middle term.

Linear search is a technique to compare each and every element with the key elements to be found, if both of them matches, the algorithm return that element found and its position also found.

Unsorted

```
a=[4,3,5,8,9,2]
s=int(input("Enter a numbers:"))
for i in range(len(a)):
    if (s==a[i]):
        print("Element found at:",i)
        break
    if (s!=a[i]):
        print("No Element found")
```

Output:

Enter any list of number 18, 22, 11, 17, 63

Enter a number to be searched: 17

Number searched is found at position: 3

(2) Unsorted:-

Algorithum:-

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 user that to be searched on the list.

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

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

Step 10:- Draw the output of given applicat

2] Sorted Linear Search:-

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

Algorithm:

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

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

Step 3:- Use for loop for method to ~~avoid~~ the list. using

:8:

Sorted

```
s=list(input("Enter the number:"))
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 founded in Position", i)
        break
else:
    print("no. not found").
```

Output:-

Enter the number 22,11,36,63,54

a=[11,22,36,54,63]

Enter a number to be searched:= 36

Number is founded at position 2

Step4:- Use sort() method to sort the accepted element and assign in for increasing order the list then print the list.

Step5:- Use if statement to give the range in which element is found in given range then display "element not found".

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

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

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

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

MD 29/11/19

Practical - 2.

Aim:- Implement binary search to find on searched no in the list

Theory:-

Binary search.

Binary search is also known as Half-intuval search algorithm that find the position of the target value within a position of a taaget value within a sorted array. If you are hooking for the number which is at the end of the list than you need to search entire list in linear search. which is ~~true~~ time comum ing. This can be avoided by using Binary fashion search.

Algorithm:-

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

2:- Using input method accept the range of given list.

3:-Use for loop, odcl elements in list using append ( ) methods.

Step4:- Use sort() methods to sort the elements and assign in increasing order list point the list after sorting.

Step5:- Use if loop to give the range in which element is found in given range then display a message element not found.

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

Step7:- Accept an argument and pay of the element that element has to be searched.

Step8:- Initialize first to 0 and last element of the list as array is starting from 0. hence it is initialized 1 less, then total count.

Step9:- Use for loop and assign the given range.

Step10:- If statement in list and still the element to be searched is not found than find the middle element (m).

Step11:- Else input the item to the searched is still less than the middle less then.

```
a=list(input("Enter list"))
a= sort()
c=len(a)
s=int(input("Enter search no:"))
if(s>a[c-1] or c<a[0]):
        print("not a list")
else:
  first,last=0,c-1
  for i in range(0,c):
      m=int(first+last)/2)
      print("found at", m)
      if s==a[m]:
          print("number found")
          break
      else:
          if s<a[m]:
              last=m-a
          else:
              first=m+1
```

Output:-

>>> Enter a range=4
Enter a number=2
[2]
Enter a number=1
[1,2]
Enter a number=4
[1,2,4].

Initialize last(h): mid(m) - 1
Else:
Initialize first(l): mid(m) - 1

ll:- Repeat still you found the element stick
the Input & output of above algorithm

Bubble sort.

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

Theory:- Bubble sort is based on the ideas of repentary comparing of adjacent element of and them swapping their position. If they exist in the increasing order. This is the simplest form of sorting available in this use sort the given element in ascending or decreasing order by comparing two adjacent element at a time.

Algorithm:-

① Bubble sort algorithm, start 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 then second then we need to swap the element.

③ If the element is smaller then second then we also ~~not~~ not swap the element.


```
# Bubble Sort
s= list(input("Enter List of nos: "))
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 list of nos: 1, 2, 5, 34, 80, 97, 1017

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

④ Agein second and third element are compared and swapped if it is necessary and this process go on is compared and swapped.

⑤ There are n ~~ele~~ element to be sorted then the process mentioned above should be repeated n-1 to get the required result.

⑥ Stick the output and input of above alogorithm of bubble sort responsible.

040

Quick Sort

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

Theory:- The quick sort is a R cursive algorithum then based on the divide and conqur technique.

Algorithm:-

Step ① Click sort first elements a value, which is called pivot value, first element sumse as our first pivot value, since we know that first will eventually and up as fast in that list.

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

③ Parts teming begin by locating two position markers lets call them left marks and right marks at the beginning and end of remaining items in the list. The goal of the partition process is to move items that are on correct lists with respects to point value while also

```
def quick(alist):
    help(alpot, 0, len(alist)-1)
def help(alist, friest, last):
    if first < last:
        Split = poat(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 not done:
        while (<=r and alist[2]<
            l = l+1
        while alist[r]>=pivot
            r = r-1
        if r<1:
            done=True
        else:
            t = alist[l]
            alist[r] = alis
            alist[r] =
        t = alist[first]
        alist[first] = a
        alist[r] = t
        return r.
```

```
x= int(input("Enter range fowr list: "))
alist=[]
for b in range(0,x):
    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].
```


converging on the split point.

step 4: We begin by increasing left marks until we loca a value that is greater than the P.V we then decrement right marks until find value that i less than the pivot value. At this point we have discovered two items that are out of place with respect to eventure split point.

step 5: At the point where rightmarks becomes less th leftmarks, we stop. The position of right mark now the split point.

step 6: In addition all the terms to left of ~~the~~ split are less than p.v & all item to the right of split point are greater than P.V. The list can now be divided at split point and quick search can invoked recesively can the two halves.

step 7: The quickest function involves a recurrive function on quicksort per.

step 8: Quicksorthelper begin with same base as merge Sort.

step 9: If length of the list is less then 0 and eq use it already sorted.

Step 10. If it is greater, then it can be partitio and recursively sorted.

Step 11. The position functron implement the pro described earlier.

Step 12 Display and stick the coding with outp of above algorithm.

23/11/22

Practical No - 5

Aim:- Implementation of stacks using Python list

Theory:- A stack is a linear data structure that can be represent in the real world in the form of the 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 added or removed only from one position on the LIFO (last in First out) principle as the elements that was inserted last will be removed first. A stack can be implemented using array as well as linked list, peek. The operation of Adding and removing the elements is known as Push and Pop.

Algorithm:

Step1:- Create a class stack with instance variable items.

Step2:- define the int method with self argument and initialize value and then initialize and empty list.

Step3:- Define method push and pop under the class stacks.

Step 4:- Use if statement to give the condition that 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 value.

Step 6:- Push method use 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 from stack at topmost position.

Step 8:- Assign the element value in push method to and print the given value is poped not.

Step 9:- Attach the input and output of above algorithm

Step 8:- First condition checks whether the no. of elements are zero while the second case whether the assigned any value if top is not assigned any value then can be sure that stack is empty.

Code:-

```
print ("Ashish Thakur")
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:
            h = self.l[self.tos]
            print("Data =", h)
            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()
```

output :-
Ashish Thakur
Data = 50

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

S.peek()

05/01/20

Practical-6.

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 provides inbuilt functions to perform the specified operation of the queue. It is a based on the principle 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 a data structure based on first in first out FIFO principle

Queue() : a new empty queue

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

Dequeue() :- Return the element which was at the f the front is moved to the successive elemer A dequeue operation cannot remove element if queue is empty.

Algorithm:-

Step1:- Define a class Queue and assign global variable then define init() method with self argument in init(), assign or initialize the initial value with the help of self argument.

Step2:- Define a empty list and define enqueue() method with 2 argument assign the length of empty list.

Step3:- Use if statement that length is equal to rear than queue is full or else insert the element in empty list or display the queue element added successfully and increament by 1.

Step4:- Define dequeue() with self argument under 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.

Step5:- Now call the queue() function and give the element that has to be added in the empty list by using enqueue() and print the list after adding and same for deleting and display the list after deleting the element from the list.


```
class Queue:
    global r
    global f
    global a
    def __init__(self):
        self.r = 0
        self.f = 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("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 deleted",va
            self.f += 1
```

```
b = Queue()
```

Output:-

```
>>> b.enqueue(1)
('inserted:', 1)
>>> b.enqueue(5)
('inserted', 5)
>>> b.enqueue(6)
('inserted', 6)
```

```
>>> b.enqueue(7)
('inserted:', 7)
>>> b.enqueue(8)
('inserted', 8
```

```
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(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 = input("Enter the : ")
r = evaluate(s)
print("The evaluated value is : ", r)
```

Practical-7

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 parent 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 righ in postfix.

Algorithm:-

Step1:- Define evaluated as function then create a empt stack in python.

Step2:- Convert the string to a list by using the string method split.

Step3:- Calculate the length of string and point it

Step4:- Use for loop to assign the range of string t give condition using if statement

Steps:- Scan the token list from left to right if token is an operand, convert it from a string to an integers and push the value onto the p:

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

Step7:- Perform the Arithmatic operation. Push the result banks am the 'm'.

Step8:- When the Input expression has been complet processed the result is am the Stauk. Pop the (p) and retoan the value.

Step9:- Print the result of string after the evaluation of Postfix.

Step10:- Attach output and Input of above algorithm.

Output:

- The evaluated value is 19:62.

049

24/01/22

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

Theory:- A linked list is a linear data structure which stored the elements in a node contiguous. The individual elements of the linked list called a Node. Node comprises of 2 parts.
① Data ② Next. Data stores all the information, w.r.t the elements for example roll no, name, address, etc, whereas the next refer to the next node. In case of larger list, if we add/remove any 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 visiting all the node in the linked list in order to perform operation on them.

Step2:- The enter linked list means can be accessed the first node of the linked list. The first node of the linked list is in turn is referred by 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 add(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.next!=None:
            print(head.data
```

Step5:- Thus, the entire linked list can be transversed using the node with is referred by the head pointer of the linked list.

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 list only.

Step3:- 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 be lead to change which we cannot reverse back.

Step6:- We may lose the referance to the 1st node in our linked list, and hence most of our linked list. so in order to avoid making some unwanted changes to the 1st node, we will use a temporary node to transverse the entire linked list.

Step7:- We will use the 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 should also be node.

Output.-

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

>>> 20
    30
    40
    50
    60
    70
    80

>>> print("Ashish Dhakur")
```

Ashish Dhakur

Step8:- Now that current is referring to the first node, if we want to access 2nd nod of list we can refer it as the next node of the 1st node

Step9:- But the 1st node is referred by correct. So we can transverse to 2nd nodes as n= n.next

Step10:- similarly, we can transverse rest of node in the linked list using the same method by while loop.

Step11:- Our concern now is to find terminating condition for the while loop

Step12:- The last node is the linked list is referred by the tail of linked list. Since the last node of linked list does not any next node, the value in the next field by of the last node is None.

Step13:- So, we can refer the last node of ~~List~~ linked list a self.s = None

Step14:- We have to now see how to start transversing the linked list & how to identify whether we reach -ed the last node of linked list or not.

Step15:- Attach the coding or input and output of above algorithm.

Aim:- Program based on Binary search tree by implementing Inorder, Preorder & Postorder Traversal.

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 subtree might have left and right subtree.
ii) Visit the root node
iii) Transverse the right subtree and repeat it.

Preorder: (i) Visit the root node
ii) Transverse the left subtree. The left subtree might have left and right subtree
iii. Transverse the right subtree repeat it

Postorder: i. Transverse the left subtree. The left subtree might have left and right subtree. ii Transverse the right subtree
iii Visit the root node.


```
#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:
                    if h.left == None:
                        h.left = p
                        print(p.val, "Node is added
                              to leftside successfull
                              at", p.val).
                        break.
                    else:
                        h = h.left

                else:
                    if h.right == None:
                        h.right = p.
                        print (p.value, "Node i
                        added torightside su
                        -fully!", h.val).
```

```
        break
    else:
        n=n.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.value)
        Preorder(root.left)
        Preorder(root.right)
def Postorder(root
    if root==None:
        return
    else:-
        Postorder(root.left)
        Postorder(root.right)
        print(root.value)
```


* Algorithm :-

Step 1:- Define class node and define init() method with 2 argument. Initialize the value is this method.

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

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

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

5 Use while loop for checking the node is less t or greater than the main node and break th loop if is not satisfy.

6 Use if statement within that else statement f checking that node is greater than main no then put it onto rightside.

7 After this, leftside subtree and rightsubtree repeat method to arrange the node accordingly Binary search Tree

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

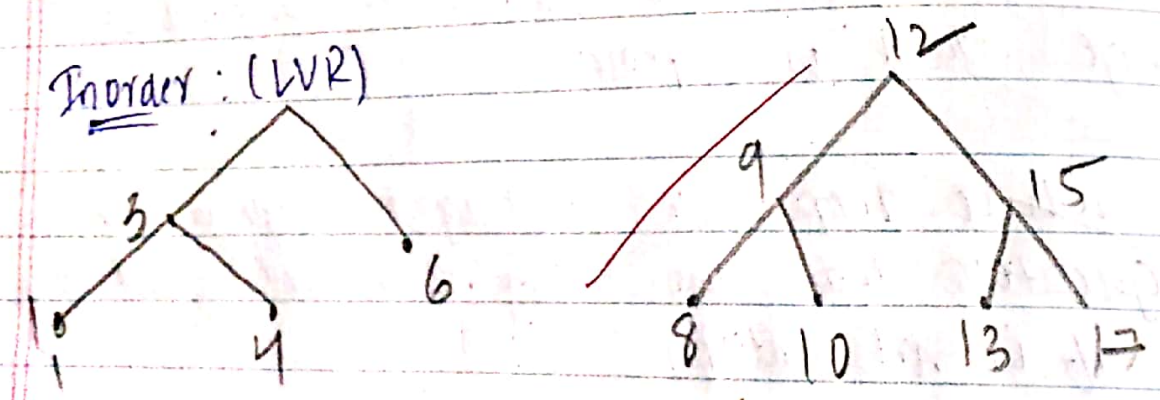
9. In Inorder, else statement used for giving that condition first left, root and then right node.

10. for Preorder, we have to give condition in else that print root, left and then right order.

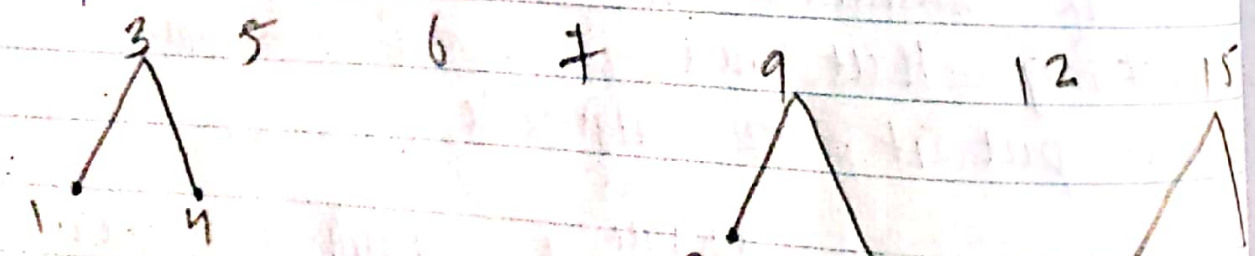
11. for Postorder, In else part, assign left then right and then go for root node.

12. Displays the output and Input of above Algorithm

Inorder : (LVR)



Step 2:-



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

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.added(6)
  ('Root is added successfully', 6)
>>> t.added(9)
  ('Root is added successfully', 9)
>>> t.added(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
```

>>> Inorder (root)

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

>>> Preorder (root)

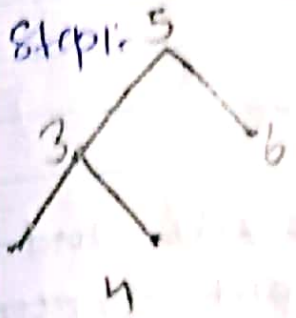
7
3
1
4
6
12
9
8
10
15
13
17

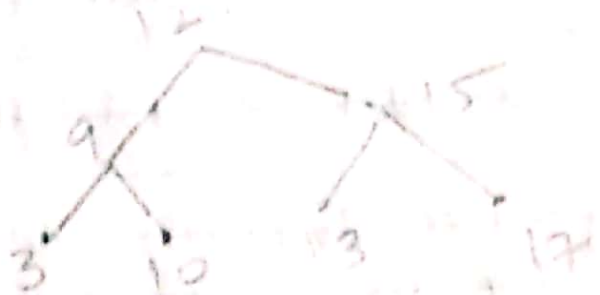
>>> Postorder (root)

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

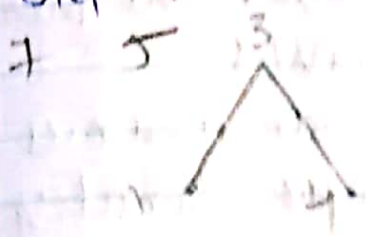
Preorder: (VLR)

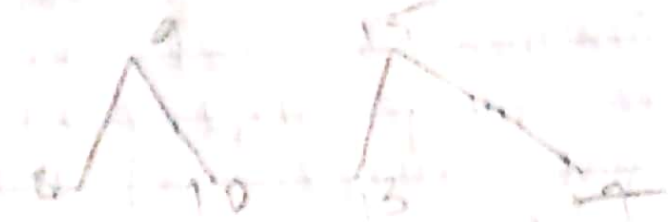
Step 1:-





Step 2:-

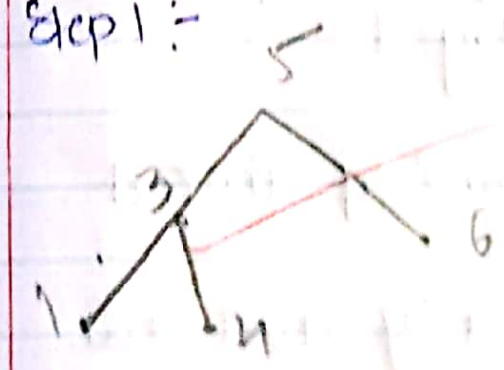


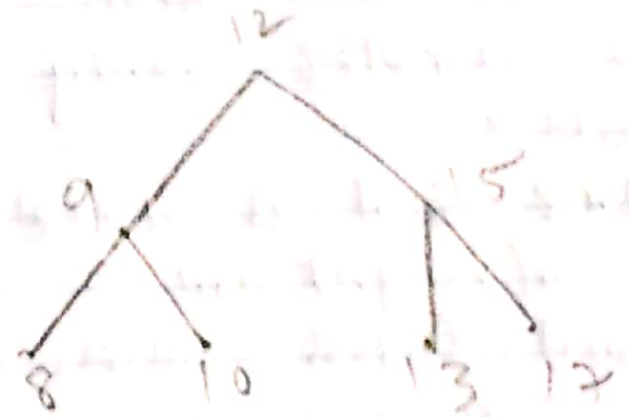


Step 3:- 7 5 3 1 4 6 12 9 8 10 15 13 17.

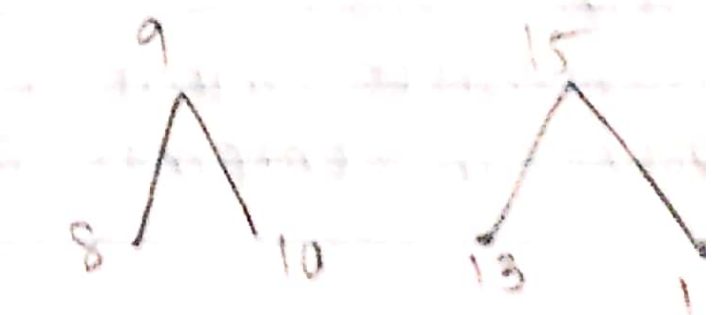
Postorder: (LRV)

Step 1:-





Step 2:-



Step 3:-

1 4 3 6 5 8 10 9 13 17

Prac-10.

Aim:- To sort a list using merge sort

Theory:- Like Quicksort, Mergesort is a Divide and Conquer algorithm. It divides input array in two halves, call itself for the two halves and then merge 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. The array is recursively divided into two halves till the size become 1. Once the size become 1, the merge process come into action and stats merging array back till the complete array is merged.

Application:-

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

Code:-

```
def mergesort(arr):                                    057
    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(righ
                                        -half):
            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("\nMERGESORTED LIST:",arr)
```

Output:-

RANDOM LIST: [27, 89, 70, 55, 62, 99, 45, 14, 10]

MERGESORTED LIST: [10, 14, 27, 45, 55, 62, 70, 89, 10]

Mergesort is more efficient than quicksort for some type of list if the data to be sorted can only be efficiently accessed sequentially, and is thus is popular where sequentially accessed data structure are very common.

Prac - 11.

Aim: To demonstrate the use of circular queue.

Theory: In a linear queue, once the queue is complete full, its not possible to insert more elements. Even if we dequeue the queue to remove some of the elements, until the queue is reset no new element can be inserted. When we dequeue any element to remove it from the queue, we are actualy moving the front of the queue forward, thereby reducing the overall size of the queue. And we cannot insert new elements, because the rear pointer is still at the end of the queue. The only way is to reset the linear queue for fresh start.

Circular Queue is also a linear data structure, which follows the principle of FIFO, but instead of ending the queue at last option, it again start 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 que

```
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

    data remove(self):
        n=len(self.l)
        if(self.f<=n-1):
            print("data is 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[s
                self.f=self.f
```

else:-
print "Que is empty
self.f=-5

q= Queue().

Output:-
>>> q= Queue()
>>> q.add(44)
('Data added: ', 44)
>>> q.add(55)
('Data added: ', 55)
>>> q.remove()
("data is removed: ', 44).

Initially, the head and the 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 the data is added tail pointer is incremented to point to the next available location

Application :- Below we have some common real-world example where circular queue are used.
1. Computer controlled Traffic Signal system use circular queue.
2. CPU scheduling and memory manager

14/02/20