PERFORMANCE

Term	Remarks	Staff Member's Signature
I	DBMS	MBhanjal 04/10/19
II	Completed	MB 14/04/20

Exam Seat No. _____

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 F.Y.B.Sc Roll No. 1803 Year 2019-2020

This is to certify that the work entered in this journal is the work of Mst. / Ms. Hemant Saw

who has worked for the year 2019-2020 in the Computer Laboratory.

04/10/19

Teacher In-Charge

15/10/19

Examiner

Head of Department

Date : _____

Exam Seat No. _____

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 SY.B.Sc Roll No. 1803 Year 2019-20

This is to certify that the work entered in this journal is the work of Mst. / ~~Ms.~~ Hemant Sood

who has worked for the year 2019-20 in the Computer Laboratory.

14/1/20

Teacher In-Charge

Head of Department

Date : _____

Examiner

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
1.	Draw ER diagram and Convert entities & relationship to table Consider the following set of requirement for a bank database.	1 - 4	18/7/19	MB
2.	Write relation Algebra Queries on the tables Created in practical - I.	5-8	25/7/19	
3.	Creating database & tables Query's in SQL to perform.	9-13	1/8/19	MB 04/19
4.	Altering, Drop & Backup of Database.	14-17	8/8/19	
5.	Aggregate function grouped by C.	18-21	22/8/19	
6.	Date function, math function & String function.	22-23	29/8/19	
7.	i) Inner join ii) Outer join.	24-25	19/9/19	
8.	In Row Inclause & Outer clause	26-27	19/9/19	
9.	View, Insert records & drop the view.	28-29	26/9/19	
10.	DCL Statements.	30-31	26/9/19	

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
1)	Implement linear search to find an item in the list.	33-35	27/11/19	20/12/19
2)	Implement Binary search to find an searched no: in list.	37-38	6/12/19	
3)	Implementation of Bubble sort program on a list.	39-40	20/12/19	23/12/19
4)	Implement quick sort to sort the given list.	41-42	20/12/19	
5)	Implementation of stack using python list	44-45	3/1/2020	03/01/2020
6)	Implementation of Queue using python list.	46-47	8/1/2020	17/01/2020
7)	Program on Evaluation of given by using stack in python environment i.e Postfix.	48-49	13/1/20	29/01/2020
8)	Linked List.	50-52	24/1/20	07/02/20

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
9.	Binary Search.	52-55	7/2/20	
10.	To demonstrate use of circular queue.	56-57	14/2/20	
11.	To sort a list using merge sort.	58-59	14/2/20	

PRACTICAL-1

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

Theory:-

LINEAR SEARCH

Linear Search is one of the simplest searching algorithm in which targetted searching algorithm in which 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 examing the middle term.

Linear Search is a technique to compare each & every element with the key element to be found, if both of them matches, the algorithm return that element found & 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 elements to be inserted into the list from the user say 'n'.
Step 3: Use for loop adding the element in to 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 elements to search 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 with the elements position.
Step 9: Use another if loop to print that the element is not found if the element which is not found if the element their in the list.
Step 10: Draw the output of given algorithm.

Unsorted


```
s = int (input ("Enter the required 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 number not found "
```

Output

```
Enter the required number 4
required number found in position 2
```

Search

```
s = list (map (int, input ("Enter the elements").split()))
s.sort()
print s
a = int (input ("Enter the number to be searched"))
for i in range (len (s)):
    if (a == s[i]):
        print "Number found! In the position", i
        break
else:
    print "Number not found!"
```

Output

Enter the elements: 9,6,7,12,5,3

[5, 6, 7, 9, 12]

Enter the number to be searched: 5

Number found! In position 3.

2) Sorted:

Algorithm:

Step 1: Create empty list & assign it to a variable accept total no of elements to be inserted into the list from user say 'n'.

Step 2: Use for loop for using append() method to add the elements in the list. Use sort() method to sort the accepted element & assign in increasing order the list then print the list.

Step 3: Use if statement to give the range in which element is found in given range then display "Element not found".

Step 4: Then use else statement, if element is not found in range then satisfy the given cond".

Step 5: Use for loop in range from 0 to the total no of elements of to be searched before doing this accept on search no. form user using input statement.

Step 6: Use if loop that the elements in the list is equal to the element accepted from user. If the element is found then print the statement that the element is found algorithm the element position.

Step 7: Use another if loop to print that if the element which is not their in the list.

Step 8: Hlath the input and output of above algorithm.

```
a=[]
n=int (input ("Enter the range "))
for b in range (0,n):
  b=int (input ("Enter the number"))
  a. append (b)
   a.sort ()
  print (a)
s=int (input ("Enter the number to search"))
if (s<a[0] or s>a[n-1]):
      print ("Element not found")
else :
   f=0
   l=n-1
   for i in range (0,n):
      m= int ((f+l)/2)
   print (m)
   if (s==a[m]):
     print ("Element found at:", m)
     break
   else:
      if (s<a[m]):
          l=m-1
      else:
          f=m+1
```

PRACTICAL - 2

Aim:- Implement Binary Search to find an Searched number in the list.

Theory:-

Binary Search is also known as Half Inolved Search, Logarithmic Search or binary is a search algorithm that finds the Position of a target value within a sorted array. If you are looking for the number which is at the end of the list then you need to search entire list in linear then you this can be avoided by using Binary fashion search.

Algorithm:-

1) Create Empty List & assign it to a variable
2) Using input method, Accept the range of given list.
3) Use for Loop, add elements in list using append() method.
4) Use Sort() method to sort the accepted element & assign it in creating ordered list. print the list after sorting.

5) Use if loop to give the range in with element is found in given range then display a "Element not found".

6) Then use else statement of statement is not found in range then satisfy the below Condn.

7) Accept an argument & key of the element that element has to be searched.

8) Initialize first to 0 & last element of the list as array is starting from 0 hence, it is initialized less than the total count.

9) Use for loop & assign the given arrange.

10) If statement in list & still the element to be searched is not found then finds the middle element (m).

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

12) Repeat till you found the element Strike the input & o/p of above algorithm.

Output:

```
>>> Enter the range : 3
>>> Enter the number: 2
    [2]
>>> Enter the number: 1
    [1,2]
>>> Enter the number to search: 3
>>> Element not found.
```

```
a = []
b = int(input("Enter number of element : "))
for i in range (0,b):
    a = int (input (" Enter the element : "))
    a.append(a)
    print(a)
for i in range (0, len(a)):
    for j in range(len(a)-1):
        if a[i] < a[j]:
            temp = a[j]
            a[j] = a[i]
            a[i] = temp
print ("Elements after sorting are:", a)
```

Output:

```
>>> Bubble sort algorithm
    Enter number of elements : 3
    Enter the elements : 8
    [8]
    Enter the elements : 9
    [8, 9]
```

8. Print ["Bubble sort algorithm"]

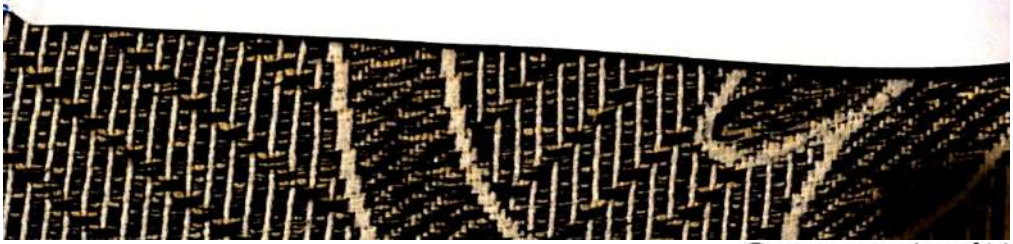
a=[]

b=

Enter the Element: 1

[8,9,1]

Elements after sorting are: [1,8,9]



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 elements & 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:-

1) Bubble sort algorithm stand by comparing the first two element of an array and swapping if necessary.

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

3) If the element is greater than second then the do not swap the element.

4) Again second and third elements are compared & swapped if necessary & this process go on until last & second last element is compared & swapped.

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

6) Write the output & input of above algorithm of bubble sort stepwise.

Q4.

Code:-

```
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 not done:
        while l <= r and alist[l] <= pivot:
            l = l+1
        while alist[r] >= pivot & r >= 1:
            r = r-1
        if r < 1:
            done = True
        else:
            t = alist[l]
            alist[l] = alist[r]
            alist[r] = t
    t = alist[first]
    alist[first] = alist[r]
```

PRACTICAL-4

QUICK SORT

Aim:- Implement Quick sort is a recursive algorithm based on the divide

Theory:- The quick sort is a Recursive algorithm based on the divide & Conquer technique

Algorithm:-

1) Quick sort first select a value which is called pivot value, first element same as our first pivot value since we known that our last.

2) The position partition process will happen next. It will find the split point & at the same time move other items to the co-operate side of the unit either less than or greater than pivot value.

3) Partitioning begins by locating two position markers called them left mark and right mark at the same time & end of

remaining items in the list. The goal of partition process is to move items that on the wrong sides with respect to pivot value while also centering on the split value.

4) We begin by increasing leftmark, until we locate a value that is greater than is greater than the Pivot the then decrement right mark until we find value that

5) At the point where rightmark became less than leftmark we became less stop. The position of right mark is now the split point.

6) The pilot value can be exchanged with the contents of staff count & pivot value is not in place.

7) The quick sort function involved a recursive function, quick sort helper.

8) The quick sort helper begins with same base as the merge sort.

```
alist [n]=t
action n
x=int (input ("Enter range for list: "))
alist=[]
for b in range (0,x):
    b=int (input ("Enter the 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: 5
Enter element: 8
Enter element: 1
[1, 3, 4, 5, 8]
```

23/12/19

9) If length of the list is the less than equal on it is already sorted.

10) If it is greater, then it can be partition on recursively sorted.

11) The partition function implement the process described earlier.

12) Display & stick the coding & output 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 physical stack or a pile. The element in the stack are added or removed from one position i.e the topmost position. Thus the stack works on LIFO principal as the element that was inserted last will be removed first. A stack can be implemented by array as well as linked list. stack has three basic operation Push, pop, Peek. The operation of adding & removing the elements is known as push & pop.

Algorithm:-

Step 1:- Create a class with instance variable items.
Step 2:- Define the init method with self arguments & the initialize the initial value & then initial to the empty list
Step 3:- Define methods push & pop under the class stack.

Source Code:-

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

03/01/22

ii

```
>>> S.push(10)
>>> S.push(20)
>>> S.push(30)
>>> S.push(40)
>>> S.push(50)
>>> S.i
[10, 20, 30, 40, 50]

Data = 50
>>> S.pop()
Data = 40
>>> S.pop()
Data = 30

>>> S.pop
Data = 20
>>> S.pop()
Data = 10
>>> S.i
[0, 0, 0, 0, 0]
>>> S.peek()
Stack is empty.
```

03/01/2020

Step 4:- Use if condition to give the condition that if length of the given list is greater than the range list than print "stack is full".

Step 5:- OR else print statement as insert the element but pop method used to delete element from stack.

Step 6:- Push method used to element but pop method used to delete element from stack.

Step 7:- If in pop method, value is less than 1, than return the stack is empty. or else delete the element from the top most position of stack.

Step 8:- First condition checks whether the no. of element are zero while the second case checks whether top is assigned any values, then we can be sure that stack is empty.

Step 9:- Assign the element values in push method to add the print given value is popped or not.

Step 10:- Attach the code & output of above.

PRACTICAL-6

IMPLEMENTING A QUEUE USING PYTHON LIST

Aim:- Implementing a queue using python list.

Theory:- Queue is a linear data structure which has 2 references front and rear. Implementing a queue using python list is the simplest as the python list provides inbuilt functions to perform the specified operations of the queue. It is based on the principal that a new element is inserted after rear & element of queue is deleted which is at front. In simple term queue is deleted which is at front. In simple term, a queue can be described as a data structure in First in first out

→ Queue (): An new empty queue.

→ Enqueue (): Insert an element at the rear of the queue. & similar to that of insertion of linked using tail

→ Dequeue (): Returns the element which was to the front the front is moved to the successive element. A dequeue operation cannot remove element if the

Code:-

```
class Queue:
    global r
    global f
    global a
    def __init__(self):
        self.f = 0
        self.r = 0
        self.a = [2,4,5,6,7]
    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 = 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", value)
            self.f = self.f + 1

b = Queue()
```

11

OUTPUT :-

```
>>> b.a
[2,4,5,6,7]
>>> b.enqueue(a)
('Inserted :', a)
>>> b.a
[a,4,5,6,7]
>>> b.dequeue()
('queue element deleted', a)
```

Algorithm:-

Step 1:- Define a class queue & assign global variable then define init() method with self argument in init(), assign or initialize the initial value with the help of self argument.

Step 2:- Define a empty list & define enqueue() method with 2 argument assign the length empty list.

Step 3:- Use if statement that length is equal to zero then queue is full or else insert the element in empty list or display that queue element added successfully & increment by 1.

Step 4:- Define dequeue() with self argument enter this use if statement that list is equal to length of list then display queue is empty or else give that front is at zero & using that delete the element from front side & increment it by 1.

Step 5:- Now, Call the Queue() function & give the element that has to be added in the empty list by using enqueue() & print the list after adding & same for deleting & display the list after deleting the element from list.

PRACTICAL - 7

Evaluation 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 parantheses. further we took car of the priorities of the operators in the program. A given postfix expression can easily be evaluated using Stack Reading the expression is always from left to right in postfix.

Algorithm:-

S1:- define evaluate as function then create a empty stack in python.

S2:- Convert the string to the list by using the string method 'Split'.

S3:- Calculate the length of string & print it

S4:- Use for loop to assign the range of string then give condition using if statement.

S5:- Scan the token list from left to right If token is operand, convert it from a

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(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))
        elif k[i] == '/':
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) / int(a))
    return stack.pop()
s = "8 6 9 * +"
x = evaluate(s)
print("The value is :", x)
```

output

The value is: 62

string to an integer & push the value onto the 'p'.

S6:- If the token is an operator *, /, +, -, it will need two operands. Pop the 'p' twice. The first pop is second operand & the second pop is the first operand.

S7:- Perform the arithmetic operation. Push the result back on the 'p'.

S8:- When the input expression has been completely processed the result is on the stack. Pop the 'p' & return the value.

S9:- Print the result of string after the evaluation of postfix.

S10:- Attach output & input of above algorithm.

24/01/2020

PRACTICAL - 8

LINKED LIST

- Aim:- Implementation of Single linked list by adding the nodes from last node 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 called a node. Node comprises of 2 parts 1) Data 2) Next Data stores all the info. wrt 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 problems.

- Algorithm :-

S1:- Traversing of a linked list means visiting all the nodes in the linked list in order to perform some operation on it.

Source Code :-

```
class node:
    global data, next
    def __init__(self, item):
        self.data = item
        self.next = None
class LinkedList:
    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
            head.next = newnode
    def addB(self, item):
        newnode = node(item)
        if self.s == None
            self.s = newnode
        else:
            newnode.next = self.s
            self.next = newnode
    def display(self):
        head = self.s
        while head.next != None
            print(head.data)
            head = head.next
        print(head.data)
```

Output.

```
>>> ho.addL(30)
>>> ho.addL(40)
>>> ho.addB(20)
>>> ho.addB(10)
>>> ho.display
10
20
30
40
```

52:- The entire linked list means be accessed use of first node of the linked list, the first node of linked list is reflected by Head pointer of Linked list.

53:- Thus the entire linked list can be transversed using the node which is referred by the head assignment of linked list.

54:- 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.

55:- We should not use the head pointer to transverse the entire linked list because of head pointer is the only reference to the first node in linked list, modifying the reference of the head pointer can lead to change while cannot back.

56:- We may lose the reference to the first node of our linked list, & hence most of our linked list making some unwanted changes to the list node, we will use the temporary node to transverse the entire linked list.

S7:- We will use this temporary node as a copy of the node we are currently traversing. Since we are making a temporary node, a copy of current node, the datatype of the temporary node should also be node.

S8:- 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 1st node.

S9:- But the 1st node is referred by current so we can traverse to 2nd node as n = n.next

S10:- Similarly we can traverse rest of nodes in the linked list using same method by using

S11:- Our concern now is to find terminating condition for using while loop.

S12:- So we can refer to it as None.

S13:- We have to now see how to start traversing the linked list & identify if we have reached the last node of linked list or not.

S14:- Write the input & output of above algorithm.

PRACTICAL - 9

BINARY SEARCH

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. Can have either 0 or 1 or 2 children that it is ordered property of binary tree that it is ordered such that one child is identified as left child & other as right child.

Inorder:- i) Traverse the left subtree. The left sub return might have left & right subtrees.
ii) Visit the root node.
iii) Traverse the right subtree & repeat it.

Preorder: i) Visit the root node.
ii) Traverse the left subtree the left subtree return.
iii) Traverse the right repeat it.

Postorder:- i) Traverse the left subtree. The left subtree return might have left & right subtrees.
ii) Traverse the right subtrees.
iii) Visit the root node.

Source Code:-

```
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 == 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 left side successfully at", h.val)
                        break
                    else:
                        h = h.left
                else:
                    if h.right == None:
                        h.right = p
                        print(p.val, "Node is added to right side successfully at", h.val)
                        break
                    else:
                        h = h.right
```

* Algorithm:-

S1:- Define class node & define init() method with 2 argument. Initialize the value in this method

S2:- Again, Define a class BST that is Binary search Tree with init() method with self argument & assign the root is None.

S3:- Define add() method for checking the condition adding the node. Define a Variable p that point (value)

S4:- 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 in leftside.

S5:- Use while loop for checking the node is less than or greater than the main node & break the loop if it is not satisfying.

S6:- Use if statement within that else statement for checking that node is greater than main root then put it into rightside.

S7:- After this, left subtree & right subtree, repeat this method to arrange the node according to Binary search Tree.

12

Q:- Define Inorder(), Preorder() & Postorder() with root as current & use if statement that root is none & return that in all.

Q:- In inorder else statement used for going that condition first left, root & then right node.

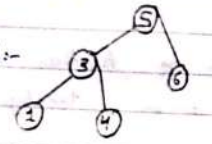
S10:- For preorder, we have to give condition in else & then go that first root, left & then right node.

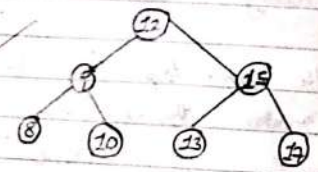
S11:- For postorder in else root, left then right & then go for root node.

S12:- Display the output & input of above Algorithm.

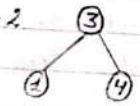
→ Inorder :- (LVR)

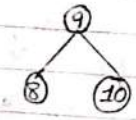
S1:-

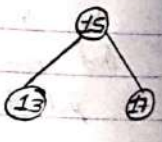




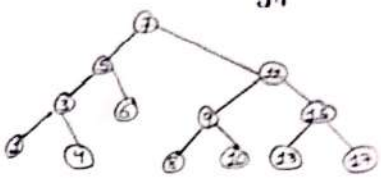
S2:- 2 3 5 6 7 9 12 15







S3:- 1 3 4 5 6 7 8 9 10 12 13 15 17



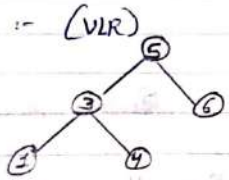
```
def Inorder(root):
    if root == None
        return
    else:
        Inorder(root.left)
        print(root.val)
        Inorder(root.right)
def Preorder(root):
    if root == None
        return
    else:
        print(root.val)
        Preorder(root.left)
        Preorder(root.right)
def Postorder(root):
    if root == None
        return
    else:
        Postorder(root.left)
        Postorder(root.right)
        print(root.val)
```

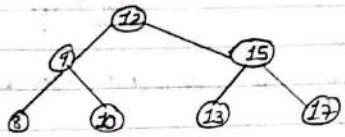
```
t = BST()
t.add(7)
t.add(5)
t.add(12)
t.add(3)
t.add(6)
t.add(9)
t.add(15)
t.add(1)
t.add(4)
t.add(8)
t.add(10)
t.add(13)
t.add(17)
```

```
print("\n Inorder form of Tree")
Inorder(t.root)
print("\n Preorder form of Tree")
Preorder(t.root)
print("\n Postorder form of Tree")
Postorder(t.root)
```

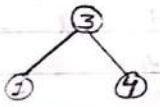
Preorder :- (VLR)

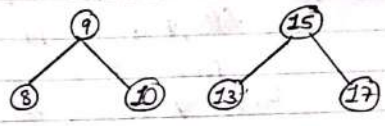
S1:-





S2:-

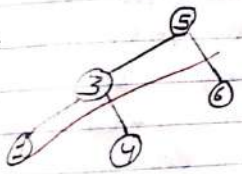


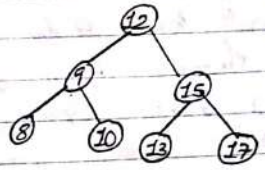


S3:- ~~7~~ 5 3 1 4 6 12 9 8 10 15 13 ~~17~~

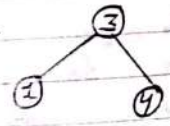
Postorder :- (LRV)

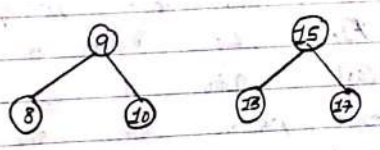
S1:-





S2:-





S3:- 1 4 3 6 5 8 10 9 13 15 12 17

PRACTICAL - 10

Aim: To demonstrate the use of circular queue.

Theory: In a linear queue, once the queue is completely full, it is not possible to insert more elements. Even if we dequeue the queue to remove some of the elements until the queue is empty, no new elements can be inserted. When we dequeue any element to remove it from the queue, we are actually moving the front of the queue forward, thereby reducing the overall size of the queue. And the 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 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, front will always point to the front of the queue & rear pointer will always point to the end of the queue.

```
Queue Code:-
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 < self.r):
            print(self.l[self.f])
            self.f = self.f + 1
        else:
            print("Queue is empty")
            self.f = 0

q = Queue()
```

OUTPUT:-

```
q = Queue()
> q.add(44)
  data added : 44
> q.add(99)
  data added : 99
> q.add(77)
  data added : 77
> q.add(88)
  data added : 88
> q.add(99)
  data added : 99
> q.add(33)
  data added : 33
> Queue is full
  q.remove()
> q.remove(44)
  data removed : 44
```

Initially, the head & the tail position 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, & once the data is always added it is incremented to point tail operation. Below we have some common real-world examples where circular queue are used:

i) Computer Controlled Traffic signal system
User cordless queue.

ii) CPU & scheduling & Memory Management.

Practical - 11

Aim: To write a program using Merge sort.

Theory: Like Quicksort Merge sort is a divide & Conquer algorithm. It divides input array into two halves & 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] & the two sorted sub-arrays into one. The array is recursively divided into two halves till the size becomes 1, the merge process comes into action & sorted array is back till the complete array is merged.

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

Source Code:-

```
def merge sort (arr):
    if len(arr) > 1:
        mid = len (arr ) //2
        left half = arr [:mid ]
        right half = arr (mid:)
        merge sort (left half)
        merge sort (right half).
        i=j =k=0
        while i<len (left half) and
                i < len (right half ):
            if leftHalf [i] <right half [j]
                arr[k] = left half [i]
                i= i+1
        else:
            arr [k] = right half [j]
            j= j+1
        k=k+1
    while i <len (Left half):
        arr [k] =left half [i]
        i= i +1
        k= k+1
```

```
arr = [27, 8, 9, 70, 55, 62, 99, 55, 14, 10]
print ("RANDOM LIST:", arr)
Merge sort (arr)
print ("\n MERGE SORTED LIST:", arr)
```

OUTPUT:-

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

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

Merge sort is more efficient than the quick sort for some types of lists if the data to be sorted can only be efficiently accessed sequentially. It is the this is applies whose data structure are very common.