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 FYBSc-(CS) **Roll No.** 1823 **Year** 2019-20

This is to certify that the work entered in this journal is the work of Mst. / Ms. Patel Aashutosh lalit

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

Teacher In-Charge

Head of Department

Date :_____

Examiner

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
	SEMESTER – II			
1)	Implement linear search to find an item in the list	35	25/11/19	2/12/19
2)	Implement Binary search to find an searched no. in the list	39	2/12/19	
3)	Implemention of Bubble sort program on given list	41	9/12/19	9/12/19
4)	Quick sort	43	16/12/19	
5)	Implementation of stack	45	6/12/19	06/01/20
6)	Implementation of Queue	42	13/1/20	
7)	Evaluation of Postfix	49	20/1/20	17/02
8)	Implementation of linkedlist	51	27/1/20	
9)	Implementation of merge sort	61	17/2/20	

☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
10)	Implementation of sets using python.	55	3/2/20	17/02/20
11)	Binary Search Based program	57	10/2/20	10/02/2020

Practical no. 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 targeted item is sequentially match with each item in the list

It is worst searching algorithm with worst case item 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 scanning the middle term.

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

1) Unsorted :

Algorithm

1) Create an empty list and assign it to a variable.

2) Accept the total no. of elements to be inserted into the list from the user say 'n'.

3) Use for loop for adding the element into the list.

4) Print the new list

5) Accept an element from the user that to be searched in the list.

6) Use for loop in a range from '0' to the total no. of elements to search the elements from the list

7) Use if loop that the elements in the list is equal to the element accepted from user.

8) If the element is found then print the statement that the element is found along with the elements position.

9) Use another if loop to print that the element is not found if the element which is accepted from user is not in the list.

10) Draw the output of given algorithm

OUTPUT :-

```
input(" linear search ")
a=[]
n = int(input("enter the range"))
for s in range(0, n):
    s = int(input("enter a number: "))
    a.append(s)
    print(a)
c = int(input("Enter a number to search"))
for i in range(0, n):
    if (a[i] == c):
        print("Number found at this position", i)
        break
else:
    print("Number not found")
```

OUTPUT:-

>>> Enter the range : 3
>>> Enter a number : 1
[1]
>>> Enter a number : 2
[1, 2]
>>> Enter a number : 3
[1, 2, 3]
>>> Enter a number to search: 2
>>> Number found at this position : 1

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

OUTPUT:

```
>>> Enter the range: 3
>>> Enter a number: 1
    [1]
>>> Enter a number: 3
    [1, 3]
>>> Enter a number: 2
    [1, 2, 3]
>>> Enter a number to search: 4
>>> Number not found.
```


2) Sorted linear search

sorting means to arrange the element in increasing or decreasing order

Algorithm:

1) Create Empty list and assign it to a variable.
2) Accept total no. of elements to be inserted into the list from user, say 'n'.
3) Use for loop for using append() method to add the element in the list
4) Use sort() method to sort the accepted elements and assign in increasing order the list, then print the list
5) Use If statement to give the range in which element is found in given range then display "Element not found".
6) Then use else statement, if element is not found in range then satisfy the given condition.
7) Use for loop in range from 0 to the total no. of elements to be searched before doing this accept and search number from the user using input statement.

8) Use if loop that the elements in the list is equal to the elements accepted from the user.

9) If the element is founed then print the statement that the element is found along with the element position

10) Use another if loop to print that the elements is not found if the x element which is accepter from user is not this in the list

11) Attach the input and output of above algorithm.

mr
2/12/19

SOURCE CODE:-

```
a[]
n=int(input("Enter a range : "))
For b in range(0,n):
    b=int(input("Enter a number : "))
    a.append(b)
    a.sort()
    print(a)

s=int(input("Enter a number to be 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)
    if(s==a[m]):
      Print("Element found at=",m)
      break
    else:
      if(s<a[m]):
        l=m-1
      else:
        f=m+1
```


Practical No. 2

AIM:- Implement Binary search to find an searched no in the list.

THEORY:-

Binary Search

Binary search is also known as half-interval search, logarithmic search or binary chop 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 will need to search entire list in linear search, which is time consuming, 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, add elements in list using append() method
4) Use sort() method to sort the accepted element and assign it in increasing ordered list print the list after sorting.

5) Use If loop to give the range in which element is found in given range then display message "Element not found".

6) Then use else statements, if statement is in range then satisfy the below condition

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

8) Initialize first to 0 and last to last element list. as array is starting from 0 hence initialized 1 less than the total count.

9) Use for loop & assign the given range

10) If statement in list and still the element searched is not found then print the element(n)

11) Else if the item to be searched is still the middle down then
Initialize last(h) = mid(m) - 1
else
Initialize first(l) = mid(m) - 1

12) Repeat till you found the element stick input & output of above algorithm.

OUTPUT:-

```
Enter a range : 5
Enter a number : 2
[2]
Enter a number: 8
[2, 8]
Enter a number: 6
[2, 6, 8]
Enter a number: 3
[2, 3, 6, 8]
Enter a number: 5
[2, 3, 5, 6, 8]
Enter a number to be search: 3
element found at position = 3
```

SOURCE CODE:-

```
print ("Bubble sort Algo: ")
a[]
b = int (input ("Enter a number of block "))
For i in range (0, b):
    s = int (input ("Enter a number: "))
    a.append (s)
    print (a)
n = len(a)
For i in range (0, n):
    For j in range (n-1):
        if a[i] < a[j]:
            temp = a[j]
            a[j] = a[i]
            a[i] = temp
print ("Element found at:- ", a)
```

2/12/19

PRACTICAL NO. 03

AIM:- Implementation of Bubble sort program on given list

Theory:- Bubblesort is based on the idea of repeatdly comparing pairs of adjacent element and then swapping their position if they exist in the wrong order. In this, we sort the given element in ascending or descending order by comparing two adjacent element at a time.

Algorithm:-

1) Bubble sort algorithm start by comparing the first two elements 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 first is smallest than second element then we do not swap the elements.

4) Again second and third elements are compared ~~from the~~ ~~process~~ and swapped if it is necessary and this process goes on until the last and second last element is compared and swapped

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

6) Display the output of the above algorithm of bubble sort stepwise.

OUTPUT:-

Bubble Sort Algo:
Enter a number of blocks 5
enter a number: 2
[2]
enter a number: 4
[2, 4]
enter a number: 6
[2, 4, 6]
enter a number: 8
[2, 4, 6, 8]
enter a number: 1
[2, 4, 6, 8, 1]

Element found at: [8, 2, 4, 6, 1]
Element found at: [2, 8, 4, 6, 1]
Element found at: [2, 4, 8, 6, 1]
Element found at: [2, 4, 6, 8, 1]
Element found at: [1, 2, 4, 6, 8]

9/10/19

```
def quick(alist):
    help(alist, 0, len(alist)):
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 + 2
    r = last
    done = False
while not done:
    while l <= r and alist[l] = pivot:
        l = l + 1
    while alist[r] = pivot and r >= l:
        r = r - 1
    if r < l:
        done = True
    else:
        t = alist[l]
        alist[l] = alist[r]
        alist[r] = t
```


AIM :- Implement quick sort to sort the given list

Theory :- The given sort is a recursion algorithm based on the divide and the conquer technique.

Algorithm:-

1) Quick sort first select a value which is called pivot value. First element serve as our first value since we know that first will eventually end up as last in that list

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.

3) Partitioning begins by locating two position markers lets call them leftmark & rightmark at the begin and end of remaining items in the list. The goal of the partition process is to move item that are on wrong side with respect to pivot while also converging.

4) We begin by incrementing leftmark until we locate a value that is greater than the pi

5) At the end point where rightmark becomes less than leftmark, we stop. The position of rightmark is how the split point.

6) The pivot value can be exchanged with content of split point and PV is now in place.

7) In addition, all items to left of split point are less than PV & all the item to the right of splitpoint are greater the PV. The line can now be divided at split point & quick can be involved on 2 values.

8) The quicksort function involves a recursive funct^n. quicksort helper.

9) Quicksort helper begins with same basi as the merge sort.

10) If the length of the list is 0 or equal to 1 it is already sorted.

11) If it is greater then it can be partioned & recursively sorted.

12) The partition function, implements the process earlier.


```
t = alist[first]
alist[first] = alist[t]
alist = [0] = t
return r

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

```
print ("Aashutosh Patel")
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 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:
            p = self.l[self.tos]
            print("Top element =", p)
stack()
```


AIM :- Implementation of stacks using python list

Theory :- A stack is an linear data structures that can be represented in the real world in the form of a physical stak or a pile. The element in the stack are added or removed only from one position i.e. the top most position. Thus the stack works on the LIFO (last in first out) principle as the element that was inserted last will be removed first. The operation of adding and removing is called as push and pop.

Algorithm :

1) Create a class stack with instance variable into.

2) Define the ~~init~~ method with self argument and initialize the initial value and then initialize an empty list.

3) Define method push and pop under the class stack.

4) Use if statement to give the condition that if length of given list is greater than the range of list.

5) Or else print the statement as insert the element into the stack and the value.

6) Push method used the insert the element but pop method used to delete the element.

7) If in pop method, value is less than return the stack is empty or else delete the element.

8) Assign the element value in push method to for and print the given value in pop.

9) Attach the input and output by above algorithm.

Output :-

Ashutosh Patel

s.push(10)
s.push(20)
s.l
[10, 20, 0, 0, 0]
s.pop()
data = 20
s.l
[10, 0, 0, 0, 0]
s.peek()
Top element = 20.

06/01/22

```
#code
class queue:
    global r
    global f
    def __init__(self):
        self.r = 0
        self.f = 0
        self.l = [0, 0, 0]
    def enqueue(self, data):
        n = len(self.l)
        if self.r < n:
            self.l[self.r] = data
            self.r = self.r + 1
            print("Element inserted ...", data)
        else:
            print("Queue is full")
    def dequeue(self):
        n = len(self.l)
        if self.f < n:
            print(self.l[self.f])
            self.l[self.f] = 0
            print("Element Deleted...")
            self.f = self.f + 1
        else:
            print("Queue is empty")
queue()
```


Aim: Implementing a Queue Python list

Theory: Queue is a linear data structure which has 2 reference front and rear. Implementation a queue using python list is the simplest as the python list provides inbuilts functions to perform the specified operation of queue. It based on the principal that a new element is inserted after rear element of queue is deleted, which is at front. In a data structure based on first in first out principle.

Queue(): Creates a new empty queue

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

Dequeue(): Returns the elements which was at the front. The front is moved to the successive element of Dequeue operation cannot removed elements of the queue is empty.

Algorithm:-

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

Step2:- Define a empty list and define enqueue() with 2 augurment.

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

Step4:- Define enqueue() with self augurment 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.

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

OUTPUT:-

>>> q.add(10)
element insert 10

>>> q.add(20)
element insert = 20

>>> q.add(30)
element insert = 30

>>> q.add(40)
element
queue is full

>>> q.remove()
10 element Deleted.

PRACTICAL No. 7

• AIM:- Program of Evaluation given string by stack in python environment i.e postfix

• Theory:- The postfix expression is free of any parenthesis further we take care of the priorities of the operators in the program.

• Algorithm:-

Step 1:- Define evaluate as function then create a empty stack in python.

Step 2:- Convert the string to a list by using the string() 'split'.

Step 3:- Calculate the length of string and print

Step 4:- Use for loop to assign the range of string then give condition using if statement.

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 p.

Step 6:- If the token is at operator +, -, /, * it will need two operands. Pop the p twice. The first pop is second operands and the second pop is the first operand.

Step 7:- Perform the Arithmatic operation. Push the result back on the 'm'

Step 8:- Print the result of string the evaluation of postfix

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

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))
        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 evaluate value is", r)
print("AP")
```

OUTPUT:-

The evaluated value is 62
AP

CODE:-

```
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 add1(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 add2(self, item):
        newnode = node(item)
        if self.s == None:
            self.s = newnode
        else:
            newnode.next = self.s
            self.s = newnode
    def display(self):
```


AIM:- Implementation of single linked list by adding the node from last position

Theory:- A linked list is a linear data structure which store the elements in a node is a linear fashion but not necessarily continuous. The individual element of the linked list called as node. Node comprises of 2 parts ① Data ② Next Data stores all the elements for example mobile no., name, address, etc. whereas next refers to next node. In case of larger list, if we add/remove any element of the list have to adjust themselves every time a new node is added.

Algorithm:-

step 1 :- Traversing of a linked list means using all the node in the linked list in order to perform some operation on them

step 2 :- The entire linked list can be accessed with the first node of the linked list. The first node of the linked list in turn is referred by the pointer of the linked list.

Step3:- The entire linked list can be traversed using the node which is referred by the head pointer of the linked list.

Step4:- Now, we can traverse the entire linked list using the head pointer that is used to refere the first node of the list.

Step5:- The head pointer shouldn't be used to traverse the list because the head pointer is our only reference to the first node in the linked list, modifying the reference of the head pointer can lead to changes which we can't afford

Step6:- Use a temporary node to traverse the the entire list thus avoiding reference the first node and the entire day.

Step7:- The temporary variable will be a copy of the node currently being traversed which will itself be a node in the list.


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

def delete(self):
    if self.s == None:
        print("list is empty")
    else:
        head = self.s
        while True:
            if head.next != None:
                d = head
                head = head.next
            else:
                d.next = None
                break
```

```
s = LinkedList()
s.add2L(50)
s.add2L(60)
s.add2L(70)
s.add2L(80)
s.add2L(40)
s.add(30)
s.add(20)
s.display()
```

52

OUTPUT:-

10
30
40
50
60
70
80

53

step 8:- Current will refer to the first node if the second node is to be accessed, we refer the next node of that list. as:

n = n.next

step 9:- Similarly we can traverse rest of nodes on the linked list using while loop.

step 10:- Our concern is to find ~~the~~ terminating condition for while loop.

step 11:- The last node (tail) of linked list has no next nodes, thus the value in its next field is NULL. As a result, we refer the last node as:

self.s = NULL.

step 12:- Attach the coding acc. input/output of the attached algorithm.

CODE :-

```
print ("Aashutosh Patel 1823")
set1 = set()
set2 = set()
for i in range (8, 15):
    set1.add(i)
for i in range (1, 12):
    set2.add(i)
print ("set1:", set1)
print ("set2:", set2)
print("\n")
set3 = set1 | set2
print ("Union of set1 and set2: set3", set3)
set4 = set1 & set2
print ("Intersection of set1 and set2: set4", set4)
print("\n")
if set3 > set4:
    print ("set3 is support of set4")
elif set3 < set4:
    print ("set3 is same as set4")
else:
    print ("set4 is subset of set3")
    print ("\n")
set5 = set3 - set4
print ("elements in set3 and not in set4: set5", set5)
print ("\n")
```


Aim :- Implementation of sets using python.

Algorithm :-

Step 1 :- Define two empty set as set 1 and set 2 now. Use for statement providing the range of above 2 sets.

Step 2 :- Now. add() method used for addition the element according to given range then print the set

Step 3 :- Find the union and intersection of above 2 set by using & (and), | (or) (). print the set

Step 4 :- Use if statement to find out the subset and superset of set 3 and set 4.

Step 5 :- Display that element in set 3 is not in set 4 using mathematical equation.

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

step 7:- Use clear() to remove all the sets and print the set after clearing the element present in the set.

```
if set4.isdisjoint(set4):
    print("set4 and set5 are mutually exclusive \n")
set5.clear()
print("after applying clear set5 is empty set:")
print("set 5":set5)
```

OUTPUT :-

>>>
Aashutosh Patel 1823
set1: {8, 9, 10, 11, 12, 13, 14}
set2: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}
union of set1 and set2: set3 {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
Intersection set1 and set2: set4 {8, 9, 10, 11}
set 3 is superset of set 4
set 4 is subset of set 3.
element in set3 and not in set4: set5 {1, 2, 3, 4, 6, 7, 12, 13, 14}
set4 and set5 are mutually exclusive after applying class, set5 is empty set
set5. set()

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

PRACTICAL NO. 11

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

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

Inorder :- 1) Traverse the left subtree. The left subtree in turn might have left and right subtree.
2) Visit the root node.
3) Traverse the right subtree and repeat it.

Preorder :- 1) Visit the root node
2) Traverse the left subtree. The left subtree in turn might have left and right subtree.

Postorder :- 1) Traverse the left subtree. The left subtree in turn might have left and right subtree.
2) Traverse the right subtree.
3) Visit the root node.

Algorithm :-

step1 :- Define class node and define init() method with 2 argument. Initialize 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.

step3 :- Define add() method for adding the node. Define a variable p that p = node(value)

step4 :- Use If statement for checking the condn that root is none then use else statement for if node is less than the main node then put on arrange that in lefside.

step5 :- Use If statement within that else statement for checking that node is greater than main root then put it into right side.

step6 :- After this, leftsubtree and right subtree, repeat this method the arrange the node according the Binary search Tree

```
        else:
            h.r = newnode
            print(newnode.data, "added on right of",
                  h.data)
            break
def preorder(self, start):
    if start != None:
        print(start.data)
        self.preorder(start.l)
        self.preorder(start.r)
def inorder(self, start):
    if start != None:
        self.inorder(start.l)
        print(start.data)
        self.inorder(start.r)
def postorder(self, start):
    if start != None:
        self.inorder(start.l)
        self.inorder(start.r)
        print(start.data)
```

```
T = Tree()
T.add(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("preorder")
T.preorder(T.root)
print("inorder")
T.inorder(T.root)
print("postorder")
T.postorder(T.root)
print("Aashutosh Patel").
```

OUTPUT :-

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

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

Aashutosh Patel

Step 8:- Inorder(), Preorder() & Postorder() with exact arrangement and use If statement that exact is same and return that in all.

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

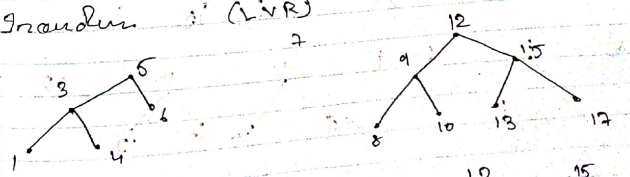
Step 10:- From Preorder, we have to give condition in else that first root.

Step 11:- From Postorder, In else part, assign left then right and then go for root node

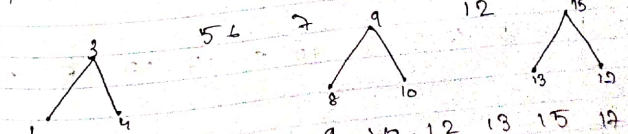
Step 12:- Display the output and input of above Algorithm.

1) Inorder :- (LVR)

Step 1)



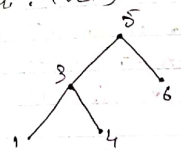
2)

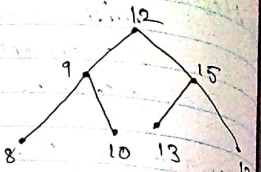


3) 1 3 4 5 6 7 8 9 10 12 13 15 17

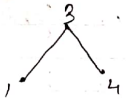
• Preorder : (VLR)

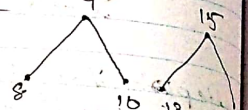
Step1) 7





2) 7 5 3 6 12

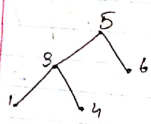


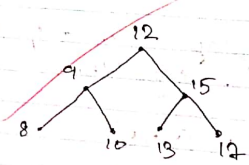


3) 7 5 3 1 4 6 12 9 8 10 15 13 17

• Postorder : (LRV)

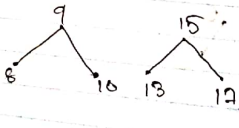
Step1:





2)

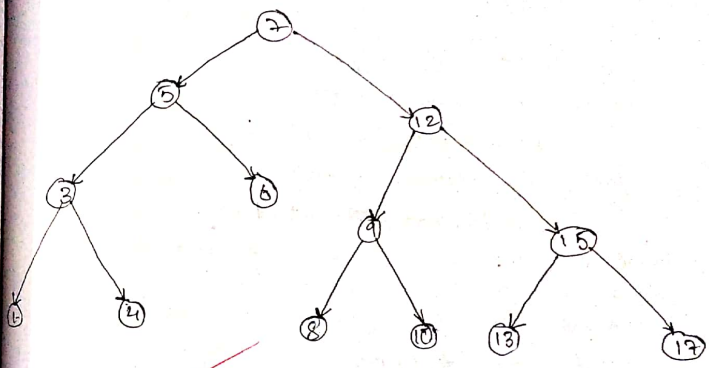
6 5



12 7

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

Binary Search Tree :-



10/02/22

```
def sort [arr, l, m, r]
    n1 = m - l + 1
    n2 = r - m
    l = [0] * [n1]
    for i in range (0, n1):
        l[i] = arr [l + i]
    for i in range (0, n2):
        R[i] = arr [m + 1 + i]
l = 0
j = 0
k = l
while i < n1 & i < n2:
    if L[i] <= R[j]:
        arr [k] = L[i]
        i + = 1
    else:
        arr [k] = R[j]
        j + = 1
        k + = 1
    while l < n1:
        arr [k] = L[i]
        i + = 1
        k + = 1
    while j < n2:
        arr [k] = R[j]
        j + = 1
        k + = 1
```


PRACTICAL :- 9

AIM :- Implementation of merge sort

THEORY :- like quicksort, merge sort is an divide & conquer algorithm. It divides input array into two halves call itself for the two halves & then merges the two sorted halves. The merge() fun. is used for merging two halves. The merge (arr, l, m, r) is key process that assumes that arr[l, m] and arr[m + 1, r] are sorted & merges. The two sorted sub. arrays into one.

ALGORITHM :-

Step 1) Define the sort (arr, l, m, r)

Step 2) stores the sorting position of both parts in temporary variables.

Step 3) Checks iff first part comes to an end or not.

Step 4) checks if second part comes to an end or not

Step 5) checks which part has smaller element.

step 6) Now the real array has element in sorted manner including both parts.

step 7) Defines the correct array in 2 parts

step 8) Sort the 1st part of array

step 9) Sort the 2nd part of array

merge both parts by comparing elements of both the parts.


```
def mergesort (arr, l, r):
    if l<r:
        m= int ((l +(r-1))/2)
        mergesort (arr, l, m)
        mergesort (arr, m+1, r)
        sort (arr , l, m, r)

    print (arr)
    n= len (arr)
    mergesort (arr, 0, n-1)
    print (arr)
```

OUTPUT:-

[12 11 13 5 6 7]

[5 6 7 11 12 13]