☆☆ INDEX ☆☆

No.	Title	Page No.	Date	Staff Member's Signature
1.	Implement linear search to find an item in the list	36	25/11/19	2/12/19
2.	Implement binary search to find a searched no. in the list	40	02/12/19	
3.	Implementation of bubble sort program on given list.	42	09/12/19	9/12/19
4.	Sorting Techniques	43	16/12/19	
5.	Stacks	46	6/1/20	17/01/20
6.	Implementation of Queue	49	13/1/20	
7.	Postfix & Prefix conversion	51	21/1/20	
8.	Implementation of Linked list	53	28/1/20	
9.	Merge & Sort	63		03/03/20

No.	Title	Page No.	Date
10.	Implementation of sets in python.	57	3/2/20
11.	Binary Search Tree	59	10/2/20

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 matched with each item in the list. It is the worst searching algorithm with worse case-time complexity. It is a free approach. On the other hand incase of an ordered list instead of searching the list in sequence. A binary search is used which will start by examining the middle term.

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

Algorithm: (unsorted)

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

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

(iii) Use 'for' loop for adding the elements into the list.

(iv) Print the new list.

(v) Accept the element that is to be searched from the user.

(vi) use the 'for' loop in range from '0' to the total no. of elements to search the elements from the list.

(vii) use 'if' loop that the element in the list is equal to the element accepted from the user.

(viii) If the element is found, print a statement that the element is found along with the elements position.

(ix) use another 'if' loop to print that the element is not found, if the element accept from the user is not in the list.

(x) Draw the output of the given algorithm.

• Sorted Linear search :

Program :

```
input ("Linear search")
a = []
n = int (input ("Enter the range: "))
for s in range (0, n):
    s = int (input ("Enter a no. : "))
    a.append (s)
    print (a)
c = int (input ("Enter a no. to search:"))
for i in range (0, n):
    if (a[i] == c):
        print ("NO. found at position :", i)
        break
    else :
        print ("NO. 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 no. to search : 2
>>> No. found at position : 1
```

Algorithm : (Unsorted)

(i) Create empty list and assign it to a variable.

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

(iii) use 'for' loop for using append() to add the elements in the list.

(iv) use sort() to sort the accepted method to add the element in the list in ascending order and print the list.

(v) use 'if' statement to give the range in which the element is found in given range and display 'Element found'.

(vi) use the else statement, if the element is not found in the given range and doesn't satisfy the condn.

(vii) use 'for' loop in range from 0 to the total no. of elements to be searched, before doing this, accept and search no. from the user using input statement.

(viii) Use 'if' loop that the element in the list is equal to the element accepted from the user.

(ix) If the element is found then print the statement that the element is found along with the current element position.

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

(xi) Attach the input and output of the above ~~otog~~ algorithm.

9/12/19

• Unsorted Linear Search :

Algorithm:

```
input ("Linear search"
a = [ ]
n = int(input("Enter the range :"))
for s in range (0, n):
    s = int (input("Enter a no.:"))
    a.append (s)
    a.sort( )
    print (a)
c = int (input ("Enter a no. to search :"))
for i in range(0, n):
    if i in range (0, n):
        if (a[i] == c ):
            print ("NO. found at
                position : ", i)
            break
    else :
        print ("No. not found.")
```

Output :

>>> Enter the range : 3
>>> Enter a no. : 1
[1]
>>> Enter a no. : 3
[1, 3]
>>> Enter a no. : 2
[1, 2, 3]
>>> Enter a no. to search : 4
>>> No. not found.

PRACTICAL NO. 2

Aim: Implement Binary search to find a number in the given 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 need to search entire list in linear search, which is time consuming.

This can be avoided by using Binary Fashion search.

Algorithm:

(i) Create Empty list and assign it to a variable.

(ii) Using input method, accept the range of given list.

(iii) Use for loop, odd elements in list using append() method.

(iv) Use sort() to sort the accepted element and assign it in increasing ordered list print the list after sorting.

(v) Use 'if' loop to give the range in which element is found in given range

then display a message "Element ~~not~~ found".

(vi) Then use 'else' statement if statement is not found in range the satify the below condition.

(vii) Accept an arguement and key of the element that has to be searched.

(viii) Initialize first ~~two~~ elements to 0 and last element of the list ie. Minus 1 less than the total count.

(ix) Use 'for' loop and assign the given range.

(x) If statement in list and still the element to be searched is not found then find the middle element (m).

(xi) 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

(xii) Repeat till you ~~found~~ find the element stick the input and output of above algorithm.

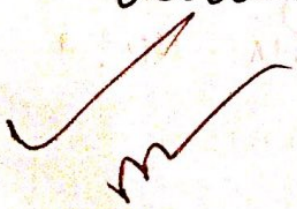
RR
2/12/19

Program:

```
a = []
n = int(input("Enter a range: "))
for b in range(0, n):
    b = int(input("Enter a no. :"))
    a.append(b)
    a.sort()
    print(a)
s = int(input("Enter a no. to be searched:"))
if (s < a[0] or s > a[n-1]):
    print("Element not found.")
else:
    f = 0
    l = n-1
    for i in range(0, n):
        m = int((f+l)/2)
        if (s == a[m]):
            print("Element found at:", m)
            break
        else:
            if (s < a[m]):
                l = m+1
            else:
                f = m+1
```

Output:

```
>>> Enter a range: 5
Enter a number: 3
[3]
Enter a number: 6
[3, 6]
Enter a number: 2
[2, 3, 6]
Enter a number: 7
[2, 3, 6, 7]
Enter a number: 4
[2, 3, 4, 6, 7]
Enter a no. to be searched: 6
Element found at: 2
```



PRACTICAL NO. 03

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

Theory : Bubble sort :

Bubble sort is based on the idea of repeatedly comparing pairs of adjacent elements and the swapping their positions if they exists in the wrong order.

This is the simplest form of sorting available. In this, we sort the given element in ascending or descending order by comparing two adjacent elements at a time.

Algorithm :

(i) Bubble sort algorithm starts by comparing the first two elements of any array, ~~one~~ and swapping if necessary.

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

(iii) If the first element is smaller than second element then we do not swap the element.

(iv) Again second and third elements are compared and swapped if it is necessary and this process goes on until the last

and second last element is compared and swapped if it is necessary and this process goes on until the last and second last element is compared and swapped.

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

(vi) Display the output of the above algorithm of bubble sort stepwise.

9/12/19

Program:

```
print ("Bubble sort")
a = []
b = int (input("Enter no. of elements : "))
for s in range (0, b):
        s = int (input("Enter the number: ")
        a. append (s)
        print (a)
n = len(a)
for i in range (0, b):
    for j in range (n-1):
        if a[i] < a[j]:
                temp = a[j]
                a[j] = a[i]
                a[i] = temp
print ("Elements after sorting: ", a)
```

Output :

>>> Bubble sort algorithm

Enter number of elements : 5

Enter the number : 7

[7]

Enter the number : 2

[7,2]

Enter the number : 9

[7,2,9]

Enter the number : 4

[7, 2, 9, 4]

Enter the number : 3

[7, 2, 9, 4, 3]

Elements after sorting : [9, 2, 7, 4, 3]

Elements after sorting : [2, 9, 7, 4, 3]

Elements after sorting : [2, 7, 9, 4, 3]

Elements after sorting : [2, 4, 7, 9, 3]

Elements after sorting : [2, 3, 4, 7, 9]

PRACTICAL NO. 4

Aim: Implement Quick sort to sort Given list

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

Algorithm: (i) Quick sort first selects a value which is called pivot value first element serve as our first value since we know that first value will eventually end up last in that list.

(ii) 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.

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

(iv) We begin by incrementing leftmark until we locate a value that is greater than left value.

(v) At the point where rightmark becomes less than leftmark, we stop. The position of rightmark is now the split point.

(vi) The pivot value can be exchanged with content of split point and pivot value

is now in place.

(vii) In addition, all items to left of split point are less than pivot value and all the item to the ~~left~~ right of split point are greater than pivot value. The line can now be divided at split point and quick can be involved on two values.

(viii) The quick sort function involves a recursive function, quick sort helper.

(ix) Quicksort helper begins with same base as the merge sort. If the length of the list is 0 or equal to 1 it is already sorted.

(x) If it is greater then it can be partioned and recursive sorted. The partition function, implement the process earlier.

```
def quick (alist)
    help (alist , 0, len(alist-1))
def help (alist , first, last)
    if first < last
        split = port (alist , first, last)
        help ( alist, first, split-1)
        help( alist , split +1, last)
def past (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 [1]
        alist [l] = alist [r]
        alist [r] < 1
    t = aust [first]
    alist [first] = alist [4]
    alist = [r] : t
    return r
```

```
x= int(input("Enter range : "))
alist=[ ]
for b in range(0, x)
    b = int(input("Enter list elements : "))
    list= append
    u = len(alist)
quick(alist)
print(alist)
```

PRACTICAL NO. 5

Aim: Implementation of stacks using python lis

Theory: A stack is a linear data structure that can be represented in the real world in the form of a physical stack or a pile. The element in the stack are added or removed only from one position it the top most position. Thus the stack works on the LIFO 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: (i) Create a class stack with instance variable

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

(iii) Define method push and pop under the class stack.

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

(v) Else print the statement as 'insert the element' and the accept the values.

(vi) Push method used to insert the element whereas the pop method is

used to delete the element.

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

(viii) Assign the element value in push method to in and print the given value in pop.

(ix) Attack the input and output by above algorithm.

```
print ("Kaushikraj Shetty")
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)
s = stack()
```

Output:

81.

KaushikRaj Shetty

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

PRACTICAL NO. 06

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 in the simplest, as python list provides inbuilt function to perform specified operation of queue. It is based on the principle that a new element is inserted after rear element of queue is deleted. It is 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 limit 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 given queue is empty.

Algorithm:

(i) Define a class queue and assign global var then define init() with self arguement and assign the initial value.

(ii) Define an empty list and define enqueue() with 2 arguements.

(iii) Use if statement that len equal to rear then queue is full. Insert the element in empty list or display that element added successfully and increment by 1.

(iv) Define enqueue() with self arguement under this. Use if statement that front is equal to length then display queue is empty or else give the front as zero.

(v) Call the queue() and give the element that has to be added in the empty list by using enque() and print the list after adding and deleting and display the list after deletion.

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 enqueue (self):
        n = len (self.l)
        if self.f < n :
            print (self.l[self.F])
            self.l[self.f] = 0
            print ("Element deleted . ")
            self.f = self.f +1
        else :
            print ("Queue is empty")

q = queue()
```

Output :

>> Q.add (10)

element insert = 10

>> Q.add (20)

element insert = 20

>> Q.add (30)

element insert = 30

>> Q.add (40)

queue is full

>> Q.remove()

10 element deleted.

PRACTICAL NO. 7

Aim: Program of evaluation given string by stack in python environment ie. postfix.

Theory: The postfix expression is free of any parantheis further. Use to take care of the peiorities of the operators in the program.

Algorithm:

(i) Define evaluate as function then create an empty stack in python.

(ii) Convert the string to the list by using the string (), 'split'.

(iii) Calculate the length of string and print.

(iv) use 'for' loop to assign the range of string then give condition using 'if' statement

(v) 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.

(vi) If the token is at operator +, −, /, * it will need two operands. Pop the p

twice. The first pop is the second operand and the second pop is the first operand.

(vii) Perform the arithematic operation. push the result back on to 'm'.

(viii) Print the result of stuing the evaluation of postfix.

(ix) Attach the input and output of the above algorithm.

10/02/2020

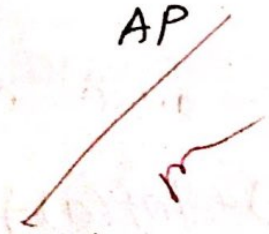
code


```
def evaluate():
    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
```



PRACTICAL NO. 8

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

Theory: A linked list is a linear data structure which ~~does~~ stores the elements in a node is a linear fashion but not necessarily continuous. The individual element of the linked list is called a node.

Node comprises of two parts:
① Data & ② Next.
Data stores all the information w.r.t the element. Next refers to the next node. In case of larger list, all elements of list have to adjust themselves every time a new node is added. To solve this tedious task, linked list is used.

Algorithm:

(i) Traversing of a linked list means using all the nodes in the linked list in order to perform some operation on them.

(ii) 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 refered by the pointer of the linked list,

(iii) Thus the entire linked list can be traversed using the node which is refered by the head pointer.

(iv) Now, we can traverse the entire linked list using the head pointer, that is used to refer the first node of the list.

(v) 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 revert.

(vi) use a temporary node to traverse to the entire list thus avoiding reference to first node and the entire day.

(vii) The temporary variable will be a copy of the node currently being traversed which will itself be a node in the list.

(viii) 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.

(ix) Similarly the rest of the list of nodes can be traversed using while loop.

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

self.s = NULL

(xi) Attach the coding on input/output of the attached algorithm.

code :


```
Class node :
    global data
    global 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.s = newnode
```

```
def display (self):
    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.addL(50)
s.addL(60)
s.addL(70)
s.addL(80)
s.addL(40)
s.add (30)
s.add (20)
s.display()
```

Output :

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

10/02/2020

PRACTICAL NO. 10

Aim : Implementation of sets using python.

Algorithm :

(i) Define two empty set as set 1 and set 2 now. Use for statement providing the range of above 2 sets.

(ii) Now use add() for addition of the element according to given range then print the set.

(iii) Find the union and intersection of above 2 set by using &(and), !(or). Print the set.

(iv) Use if statement to find out the subset and superset of set 3 and set 4.

(v) Display the elements in set 3 is not in set 4 using mathematical operation.

(vi) Use disjoint() to check if anything is common or, whether element is present or not. If not then display that it is mutually exclusive event.

(vii) Use clear() to remove or delete the sets and print the set after leaving the element present in the set.

(viii) Print corresponding output to the user with the above attached code.

Code :

```
print ("KaushikRaj")
set1 = set()
set2 = set()
for i in range(8,15):
    set1.add(i)
for i in range(1,12):
    set2.add(i)
print("set 1 :", set1)
print("set 2 :", set2)
print("\n")
set3 = set1 / set2
print("Union : ",set3)
set4 = set1 & set2
print("Intersection : ", set4)
print("\n")
if (set3 > set4):
    print("set 3 is subset of set")
    print("\n")
elif set3 < set4:
    print("set 3 is same as set4.")
else:
    print("Set 4 is subset of set 3")
    print("\n")
set5 = set3 - set4
print("Element in set 3 & not in set 4: set5",set5)
print("\n")
```

```
if set4.isdisjoint(set4):
    print("set 4 & set 5 are mutually exclusive")
set5.clear()
print("After applying clear, set 5 is empty")
print("set 5:", set5)
```


Output:

KartikRaj

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

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

union of set1 & set 2 : set3 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 }

Intersection set1 & set 2 : set 4 { 8, 9, 10, 11 }

set 3 is support of set 4

set 4 is subset of set 3.

Element in set 3 & not in set 4 : set 5

{ 1, 2, 3, 4, 6, 7, 12, 13, 14 }

set 4 and set 5 are mutually exclusive

after applying class, set 5 is empty set

set5.set()

PRACTICAL NO. 11

Aim: Program based on Binary search Tree by implementing inorder, preorder & postorder.

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.

- **Inorder:** (i) Traverse the left subtree. The left subtree inturn might have left and right subtrees.
  (ii) Visit the root node.
  (iii) Traverse the right subtree and repeat it.

- **Preorder:** (i) Visit the root node.
  (ii) Traverse the leftsubtree. The left subtree inturn might have left and right subtree.
  (iii) Traverse the right subtree. Repeat it.

- **Postorder:** (i) Traverse the left subtree. The left subtree inturn might have left and right subtree.
  (ii) Traverse the right subtree.
  (iii) Visit the root node.

Algorithm:

(i) Define class node and define init() method with 2 arguement. Initialize the value in this method.

(ii) Define a class BTree that is Binary search Tree, with init() with self arguement and assign the root is None.

(iii) Define add() for adding the node. Define a variable p that holds node valu

(iv) use 'if' statement for checking the condition that root is None(NULL) then we use else statement for if node is less than the main node the put or arrage it leftside.

(v) Use while loop for checking the node is less than or greater than the main node and break the loop if it is not satisfying.

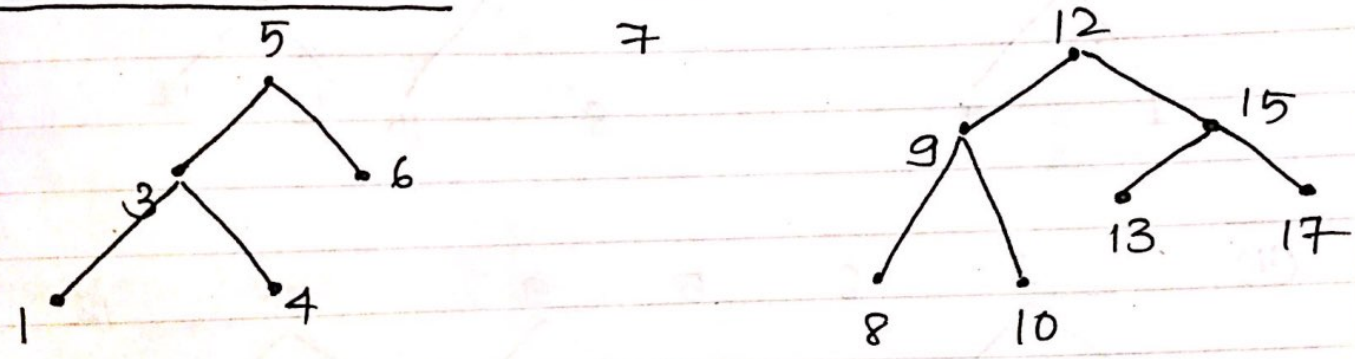
(vi) Use if statement within that else statement for checking which node is greater and assign it to the right.

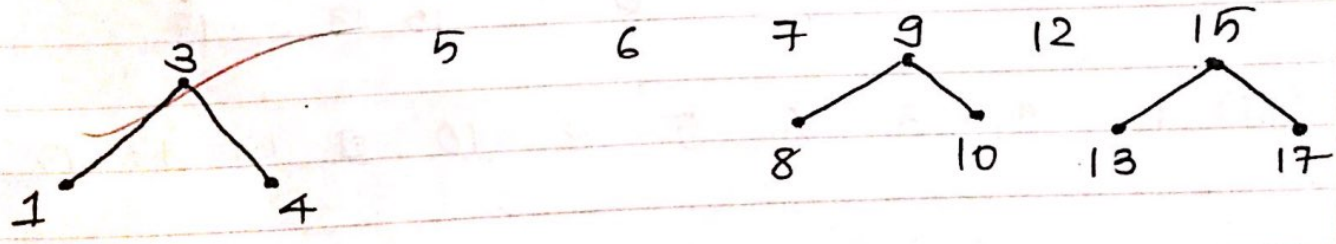
(vii) Repeat left and right subtree to arrange the node according to binary tree.

(viii) Define Inorder(), Preorder() and Postorder() with root arguement and use 'if' statement that root is none and return it.

(ix) Display the output in systematic order of: Inorder (Left - Root Right)
Postorder (Left - Right - Root)
Preorder (Root - Left - Right)

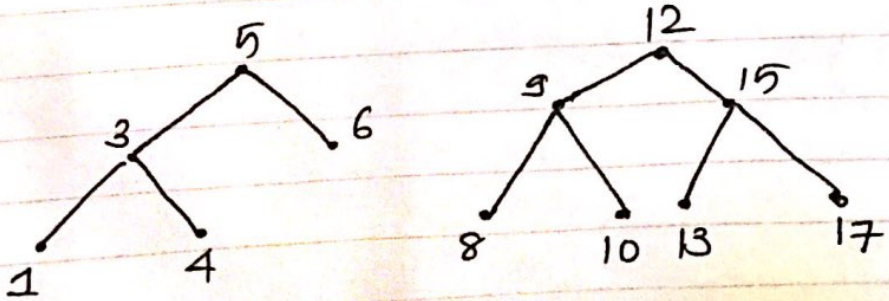
Manual Calculation: (Inorder)

i. 

ii 

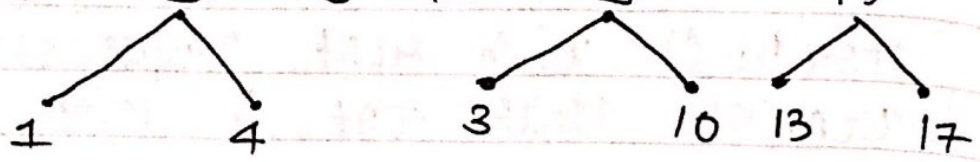
iii 1 3 4 5 6 7 8 9 10 12 13 15 17

(Preorder)

(i) 7 

19.

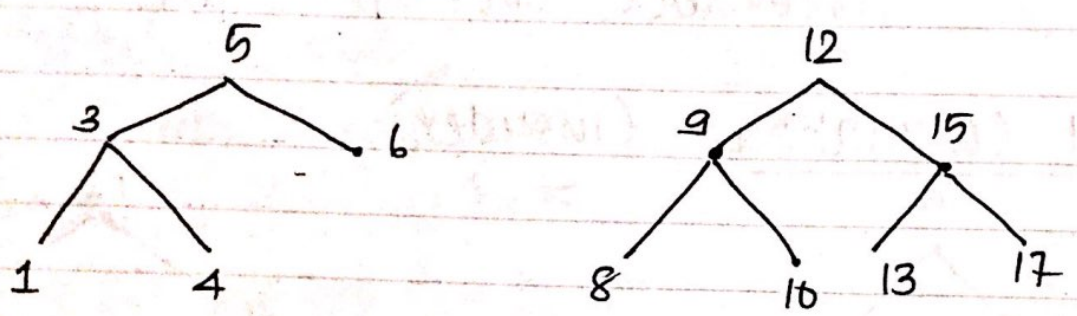
(ii) 7 5 3 6 12 9 15



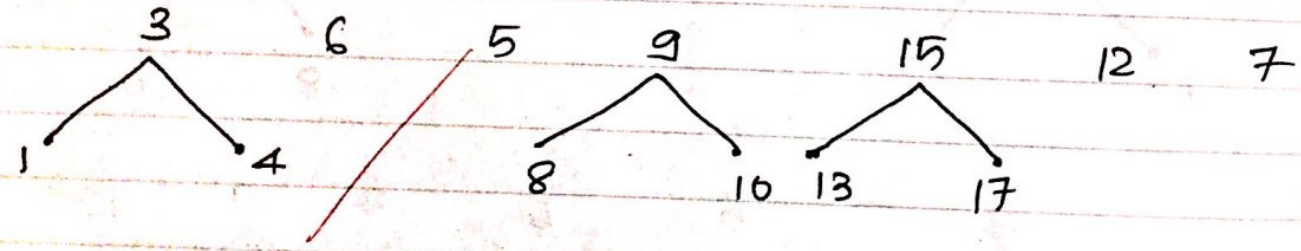
(iii) 7 5 3 1 4 6 12 9 8 10 15 13 17

(Postorder)

(i)



(ii)



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

#Code :

```
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(value)
        else:
            newnode = Node(value)
            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 to left :", h.data)
                        break
                else:
                    if h.r != None:
                        h = h.r
                    else:
                        h.r = newnode
```

```
                print(newnode.data, "Added to
                                right: ", 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:
            ~~print~~
            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)
```

Q2.

```
T.add(4)
T.add(8)
T.add(10)
T.add(13)
T.add(17)
print("Preouder")
T.preouder(T.root)
print("Inouder")
T.preouder(T.root)
print("Postorder")
T.postorder(T.root)
```

Output:

5 Added on Left of: 7
12 Added in Right of: 7
3 Added on Left of: 5
6 Added in Right of: 5
9 Added on Left of: 12
15 Added in Right of: 12
1 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 in Right of: 15

Preorder:

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

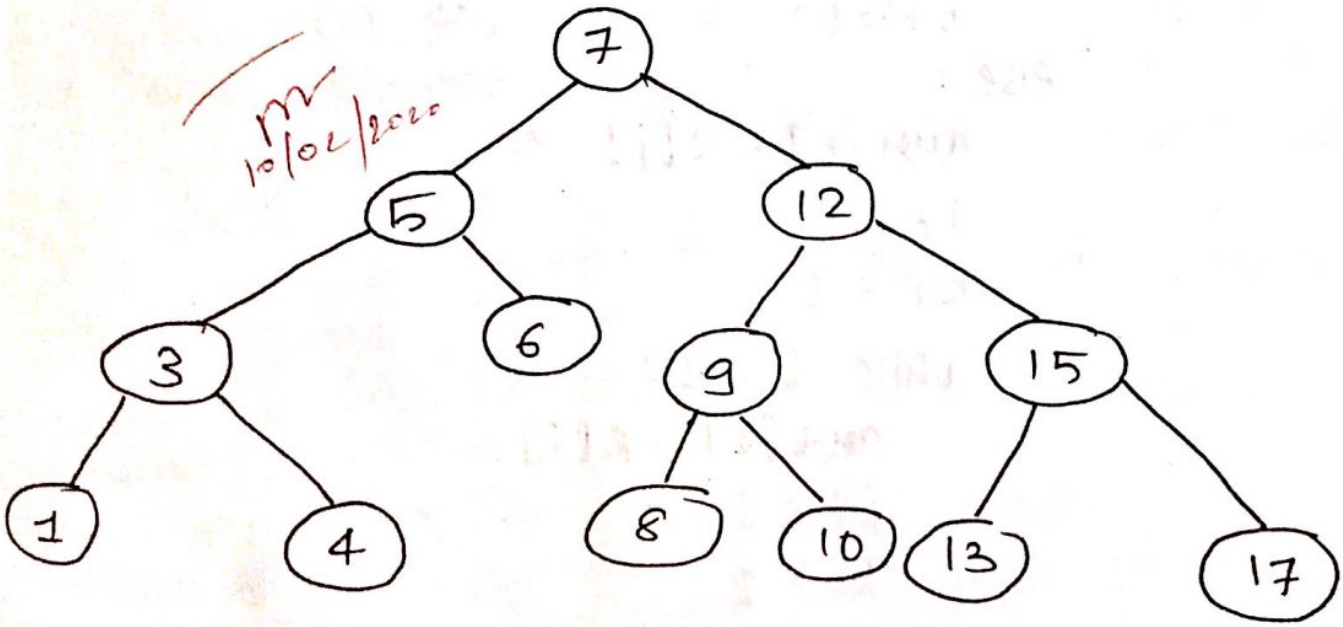
Inorder

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

Postorder:

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

Binary Tree search:



10/02/2020

PRACTICAL NO. 9

Aim: Implementation of merge sort.

Theory:

Like Quick sort, merge sort is a divide and conquer algorithm it divides input array in two halves and then merge the two halves. The merge(arr, l, m, r) is key process that assumes that arr(l, m) and arr(m+l-r) are sorted and merges the two sorted sub-arrays into one.

Algorithm:

(i) Define the sort (arr, l, m, r)
(ii) Stores the starting position of both parties in temporary variables.
(iii) Checks if first part comes to an end or not.
(iv) Checks if second part comes to an end or not.
(v) Checks which parts has smaller elements.
(vi) Now the real array has elements in sorted manner including both parts.

(vii) Define the correct array in two parts.

(viii) Sort the first part of array.

(ix) Sort the second part of array. Merge both parts by comparing elements of both parts.

```
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+1]
    for i in range (0, n2):
        R[i] = arr [m+1+j]
    L = 0
    j = 0
    K = l
    while i<n1 & i<n2:
        if l[i] <= R[j]
            arr [w] = l [i]
            i+ = l
        else:
            arr [t] = R [j]
            j+ = 1
            K+ = 1
            while l<n1:
                arr [k] = R[j]
                j+ = 1
                k+ = 1
```

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

03/03/2020.