780

Practical No:- 1

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

Theory:-

Linear Search

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

It is worst searching algorithm with worst case time complexity. It is a force approach. On the other hand 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 carry element with the key element to be ~~formed~~ found, if both of them matches, the algorithm returns that element found and its algorithm is also formed.

1]. Unsorted:-

Algorithm

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

Step2:- Accept the total no. of elements to be inserted into the list from the user ~~so~~ Say 'n'.

Input :-

```
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
```


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

Step4 :- Print the new list.

Step5 :- Accept an element from the user that to be searched in the list.

Step6 :- Use for loop in a range from "0" to the total no. of elements to search the elements from the list.

Step7 :- Use if loop that the elements in the list is equal to the element accepted from user.

Step8 :- If the element is found then print the statement that the element is found along with the elements position.

Step9 :- Use another if loop to print that the element is not found if the element which is accepter from user is not their in the list.

Step10 :- Draw the output of given algorithm.

2]. Sorted Linear search
Sorting means to arrange the element in increasing or decreasing order.

Algorithm:-

Step1:- Create Empty list and assign it to a variable.

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

Step3:- Use for loop for using append() method to add the element in the list.

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

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

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

Step7:- Use for loop in range from 0 to the total no. of elements to be searched before doing this accept and search number from the user using Input statement.

Input :-

```
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)
    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
```

Step8:- Use if loop that the elements in the list is equal to the element accepted from the user.

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

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

Step11:- Attach the input and ouput of above algorithim.

2/11/15

Practical No:-2

Aim:- Implement Binary search to find the 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 need to search entire list in linear search, which is time consuming. This can be avoided by using Binary fashion search.

Algorithm:-

Step1:- Create Empty list and assign it to a variable

Step2:- Using Input method, accept the range of given list.

Step3:- Use for loop, add elements in list using append() method.

Step4:- Use sort() method to sort the accepted elements and assign it in increasing ordered list print the list after sorting.

Input:-


```
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 search number:"))
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
```

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

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

Step7:- Accept an arguement & key of the element that element has to be ~~sorted~~ searched.

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

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

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

Step11:- 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

Step12:- Repeat till you found the element stick the input & output of above algorithm.

Output:-

Enter a range = 4
Enter a number = 2
[2]
Enter a number = 1
[1,2]
Enter a number = 4
[1,2,4]
Enter a number = 8
[1,2,4,8]
Enter a search number = 2
Element found at = 1.

Practical No: 3

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

Theory:-

Bubble sort

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

Algorithm:-

Step1:- Bubble sort algorithm starts by comparing the first two elements of an array and swapping if necessary.

Step2:- 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.

Input:-

```
print ("Bubble sort algorithm")
a = []
b = int (input ("Enter number of elements="))
for s in range (0,b):
    s = int (input ("Enter the element:"))
    a.append(s)
    print (a)
n = len (a)
for i in range (0,n):
    for j in range (n-1):
        if a[j]<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 element = 7
[7]
Enter the element = 10
[7,10]
Enter the element = 5
[7,10,5]
Enter the element = 4
[7,10,5,4]
Enter the element = 8
[7,10,5,4,8]
Elements after sorting = [4,5,7,8,10]
```

9/1/19

Step3:- If the first element is smaller than second then we do not swap the element.

Step4:- Again second and third elements are compared and swapped if it is necessary and this process go on until last and second last element is compared and swapped.

Step5:- If there are n elements to be sorted then the process mentioned above should be repeated n-1 times to get the required result.

Step6:- Stick the input & output of above algorithm of bubble sort stepwise.

Practical No:- 4

Aim:- Implement Quick sort to sort the given list

Theory:-

Quick Sort

The quick sort is a reverse algorithm based on the divide and conquer technique.

Algorithm:-

Step 1:- Quick sort first selects a value which is called pivot value, first element serve as our first pivot value, Since we know that first will eventually end up as last in that list.

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

Step 3:- Partitioning begins by locating two position markers let call them leftmark and rightmark. At the beginning and end of the remaining items in the list. The goal of the partition process is to move items that are on wrong side with respect to pivot value

Coding:-


```
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 and r>=l:
            r=r-1
        if r<l:
            done=True
        else:
            temp=alist[l]
            alist[l]=alist[r]
            alist[r]=temp
```

```
        temp = alist[first]
        alist[first] = alist[r]
        alist[r] = temp
        return r
x = int(input("Enter range for the list:"))
alist = []
for b in range(0,x):
    b = int(input("Enter element:"))
    alist.append(b)
    n = len(alist)
quick(alist)
print(alist)
```

Output:-

```
>>> Enter range for the list: 4
Enter element:55
Enter element:47
Enter element:23
Enter element:650
[23, 47, 55, 650]
```

Step 4:- We begin by incrementing leftmark until we locate a value that is greater than the pivot value. Then we decrement rightmark until we find value that is less than the pivot value. At this point, we have discovered two items that are out of place with respect to eventual split point.

Step 5:- At the point where rightmark becomes less than the leftmark, we stop. The position of rightmark is now the split point.

Step 6:- The pivot value can be exchanged with the content of split point and pivot value is now in place.

Step 7:- In addition, all the items to left of split point are less than pivot value and all the items to the right of split point are greater than pivot value. The list can now be divided at split point and quick sort can be invoked recursively on the two halves.

Step 8:- The quick sort function invokes a recursive function, quicksorthelper.

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

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

Step 11:- If it is greater, than it can be partitioned and recursively sorted.

Step 12:- The partition function implements the process that was described earlier.

Step 13:- Display and stick the coding and output of the above algorithm.

Coding :-

```
class stack:
    def __init__(self):
        global tos
        self.l=[0,0,0,0,0,0,0,0]
        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.tos=self.tos-1
    def peek(self):
        if self.tos<=0:
            print("stack is empty")
        else:
            a=self.l[self.tos]
            print("data=",a)
s=stack()
```


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 a physical stack or a pile. The elements in the stack are added or removed only from one position i.e, the topmost position. Thus, the stack works on the LIFO (Last in First Out) principle as the element that was inserted last will be removed first. A stack can be implemented using array as well as link list. Stack has three basic operation :- push, pop, peek. The operations of adding and removing the elements is known as push & pop.

Algorithm :-

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

Step 2 :- Define the init method with self argument and initialize the initial value and then initialize to an empty list.

Step 3 :- Define methods push and pop under the class stack.

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

Step 5:- Or Else print statement as P insert the element into the stack and initialize the va

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

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

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

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

Step 10:- Attach the input and output of above algorithm.

Output :-


```
s.push(10)
s.push(20)
s.1
[10,20,0,0,0]
s.pop()
data = 20
[10,0,0,0,0]
s.pop()
data = 10
s.pop()
stack is empty
s.peek()
data = 20
```

```
# 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")
q.queue()
```


Practical No:- 6

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 and element of queue is deleted which is at front. In simple terms, a queue can be described as a data structure based on first in first out FIFO principle.

* Queue():- 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. A dequeue operation cannot remove if the queue is empty.

Algorithm:-

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

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

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

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

Step 5:- Now call the Queue() function and give the element that has to be added in the empty list by using enQueue() and print the list after adding and same for deleting and display the list after deleting the elements from the list.

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.deleremove()
    10 element deleted
```

```
# 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))
    return stack.pop()
s = "869 * +"
r = evaluate(s)
print("The evaluated value is", r)
print("UJ")
```

Practical No:- 7

Aim:- Program on Evaluation of given string by using stack in Python Environment ie Postfix.

Theory:- The postfix expression is free of any parenthese. Further we took care of the priorities of the operators in the program. A given postfix expression can easily be evaluated using stacks. Reading the expression is always from left to right in postfix.

Algorithm:-

Step 1:- Define evaluate as function then create an empty stack in Python.

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

Step 3:- Calculate the length of string and print it.

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

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

Step 8:- When the input expression has been completely processed, the result is on the stack. Pop the 'p' and return the value.

Step 9:- Print the result of string after the evaluation of Postfix.

Step 10:- Attach output and input of above algorithm.

Output

The evaluated value is 62

Uj

```
## 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 addL(self,item)
        new node: node(item)
        if self.s = NONE
            self.s = newnode
        else:
            head = self.s
            while head.next! = NONE
                head = head.next
                head.text = new node
    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)
        delete(self):
        if self.s = NONE;
            print("List is Empty")
```


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 stores the elements in a node in a linear fashion but no necessarily contiguous. The individual element of the linked list called a Node.

Algorithm:-

Step 1:- Transversing of a linked list means visiting all the nodes in the linked list inorder to perform same operation on them.

Step 2:- The entire linked list means can be accessed us the first node of the linked list the first node of the linked list int term of referred by the head pointer of the linked list.

Step 3:- Thus the entire linked list can be transversed using the node which is referred by head pointer.

Step 4:- Now that we know that we can transverse the entire linked list using the head pointer we should only we it to refer the first node of list only.

Step 5:- We should not use head pointer to transverse the entire linked list because the head pointer is our only reference to the 1st node in the linked list, modifying the reference of the head pointer ~~can~~ will lead to ~~chages~~ changes which we cannot revert back.

Step 6:- We may lose the reference to the 1st node in our linked list and ~~hence~~ hence most of one list so in order to avoid making some unwanted charges to the 1st node we will use a temporary node to transverse.

Step 7:- We will use this temporary node as a copy of the node we are currently transversing. Since we are ~~missing~~ making temporary node a copy of current the datatype of the temporary node should also be node.

Step 8:- But the 1st node is referred by current so we can transverse to 2nd ~~node~~ node as h=h.next.

Step 9:- Similarly we can transverse rest of nodes in the linked list using some method by while loop.

```
        else:
            head=self.s
        while True:
            if head.next!=NONE:
                d=head
                head=head.next
            else:
                d.next=NONE
                break
```

056

```
s=linkedlist()
s.addL(50)
s.addL(80)
s.addL(70)
s.addL(60)
s.addL(40)
s.add (30)
s.add(20)
s.display()
```

Output:-

20
30
40
50
60
70
80

Step 10:- Our concern now is to find terminating condition for while loop.

Step 11:- The last node in the linked list is referred to the tail of linked list. : the last node of linked list does not have any next node. The value in the next field of the last node.

Step 12:- We can refer to the last of linked list self.s none.

Step 13:- We have to now see how to start transversing the linked list how to identify whether we have reached the last node of linked list or not.

Step 14:- Attach the coding or input and O/P of above algorithm.

Practical No: 9

Aim:- Implementation of Merge Sort.

Theory:- Like Quick Sort merge sort is a Divide & conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves. The merge (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 starting position of both parts in temporary variables.

Step 3:- Check 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 parts has smaller elements.

```
#Code
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 j in range (0, n2):
        R[j] = arr [m+1+j]
    i = 0
    j = 0
    k = l
    while i<n1 & j<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
def mergesort(arr, l, r):
    if l<r:
        m = int((l+(r-1))/2)
        merge sort (arr, l, m)
        merge sort (arr, m+1, r)
```

```
def 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]

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 the both ~~path~~ parts by comparing elements of both the parts.

Practical No:-10

Algorithm :-

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

Step 2 :- Now add() method used for adding the element according to given range, then print the sets after adding.

Step 3 :- Find the union and intersection of above 2 sets by using (coma), : (or) method print the sets of union and info as set 3 and set 4.

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

Step 5 :- Display that element in set 3 and set 4 using mathematical operation.

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

```
# Code
print ("Tushar Dhule 1839")
set 1 = {set()
set 2 = set()
for i in range (8,15):
    set 1.add(i)
for i in range (1,12):
    set 2.add(i)
print ("set 1:")
print ("set 2:", set 1)
print ("\n", set 2)
set 3 = ~~set 1 | set 2~~ set 1 | set 2
print ("Union of set 1 and set 2: set 3", set 3)
Set 4 = set 1 & set 2
print ("Intersection of set 1 and set 2: set 4", set 4)
print ("\n")
if set 3 > set 4:
    print ("set 3 is superset ~~subset~~ of set 4")
elif set 3 < set 4:
    print ("set 3 is subset of set 4")
    print ("\n")
set 5 = set 3 - set 4
print ("Elements in set 3 and not in set 4: set 5"
print ("\n")                                    set 5)
if set 4 is disjoint (set 5):
    print ("set 4 and set 5 are Mutually
            Exclusive \n")
set 5.clear()
print ("After applying clear, set 5 is empty
                                    set:")
print ("set 5:", set 5)
```

060

Output :-

>>> Tushar Dhule

set 1 : { 8, 9, 10, 11, 12, 13, 14 }

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

Union of set 1 and set 2 : set 3 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 }

Intersection set 1 and set 2 = set 4 { 8, 9, 10, 11 }

set 3 is superset of set 4

set 4 is subset of set 3

element in set 3 and 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

set 5. set ()

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

Practical No: 11

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

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

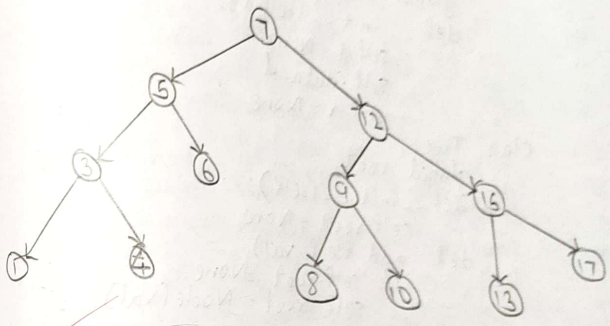
→ Inorder :- i) Transverse the left subtree, the left subtree intum might have left and right subtrees
ii) Visit the root node
iii) Transversion the right subtree and repeat it.

→ Preorder :- i) Visit the root node
ii) Traverse the leftsubtree intum might have left and right subtree.
iii) Traverse the right subtree repeat it.

→ Post order :- i) Traverse the left subtree. The left subtree intum might have left and right subtrees
ii) Traverse the right subtrees
iii) Visit the root node.

062

Binary Search Tree



```
#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(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
                    else:
                        h.r = newnode
                        print (newnode.data "added
                    on right of" h.data)
```


Algorithm :-

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

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

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

Step 4 :- Use If statement for checking the condition that root is none then use else statement for if node is less than the main node not then then put or arrange that in leftside.

Step 5 :- 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.

Step 6 :- Use If statement within that else statement for checking that node is greater than main root then put in into rightside.

Step 7:- After this leftsubtree and right subtree repeat this method to arrange the node accordingly to Binary search tree.

Step 8:- Define Inorder(), Preorder() and postorder() with root argument and we If statement that root is none and return that in all.

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

Step 10:- For preorder, we have to give condition in else that first root, left and then right node.

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

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

```
def break
    preorder(self,start):
    if start!=None:
        print (start.data)
        self.preorder(start.l)
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(100)
T.add(80)
T.add (70)
T.add (85)
T.add (10)
T.add (78)
T.add (60)
T.add (88)
T.add(15)
T.add(12)
print ("preorder")
T.preorder (T.order)
print ("inorder")
T.inorder (T.root)
```

print ("Postorder")
T. postorder (T.root)
print ("Tushar Dhule")

Output :-

80 added on left of 100
70 added on left of 80
85 added on right of 80
10 added on left of 70
78 added on right of 70
60 added on right of 10
88 added on right of 85
10 added on left of 60
12 added on left of 15

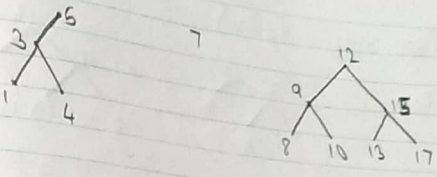
Preorder
100
80
70
10
60
15
12
78
85
88

Inorder
10
12
15
60
70
78
80
85
88
100

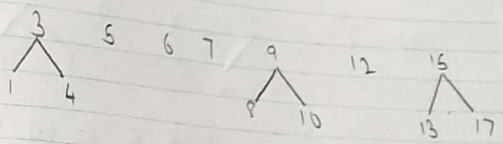
Postorder
10
12
15
60
70
78
80
85
88
100
Tushar Dhule

Inorder (LVR)

Step 1 :



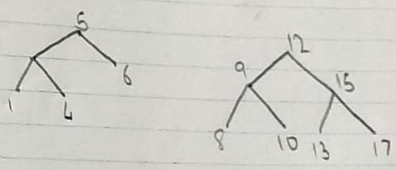
Step 2 :



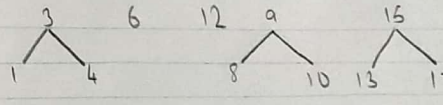
Step 3 : 1 3 4 5 6 7 8 9 10 12 13 15 17

Preorder (VLR)

Step 1 : 7



Step 2 : 7 5 3 6 12 9 15

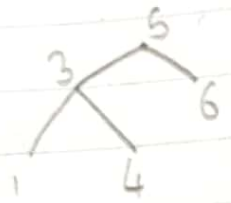


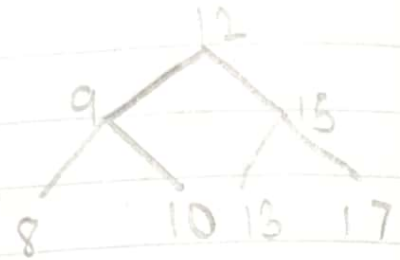
Step 3 : 7 5 3 1 4 6 12 9 8 10 15 13

280

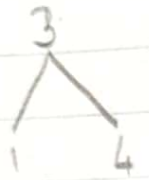
Post order : (LRV)

Step 1 :

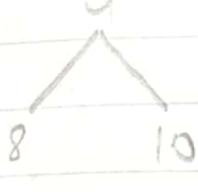


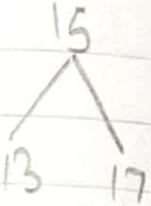


Step 2 :



6 5





12 7

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