Sorted array :

Code :

```
def linear (arr, x) :
    for i in range (len (arr)) :
        if arr[i] == x :
            return i

inp = (input (" Enter elements in array : ")). split()
array = [ ]
for ind in inp :
    array. append (int (ind))
print (" elements in array are : ", array)
array. sort()
x1 = int (input (" enter element to be searched :"))
x2 = linear (array, x1)
if x2 == x1 :
    print (" Element found at position ", x2)
else :
    print (" Element not found ")
```

>>> Enter elements in array 1 2 3 4 5
>>> Elements in array are : [1, 2, 3, 4, 5]
>>> Enter element to be searched : 2
The element is found at position 1.

OS - PRACTICAL 1.

Aim : Linear Search

Q1. Sorted Array :

Algorith :

Step 1 : Define a Function with two parameters use for conditional statement with range.

step 2 : Now use if conditional statement to check whether the given number by user is equal to elements in the array.

Step 3 : If condition in step 2 satifies, return the index no. of the given array. If the condition dosen't satisfy then get out of loop.

Step 4 : Now initialize a variable to enter elements in the array From user. Now use split() method to split the values.

Step 5 : Now initialize a vanable as empty array.

Step 6 : Use for conditional statement to append the elements given as input by user in the empty array.

Step 7 : Now again initialize another variable to ask user to find the element in the array.

Step 8: Again initialize a variable to call the defined function.

Step 9: Use if conditional statement to check if variable in step 8 matches with the element you want to find then print the index corresponding to the element. If condition dosen't satisfy then print that element is not found.

Q2. Unsorted Array:

Algorith:

Step 1: Define a function with 2 parameters use for conditional statement with range that is length of array to find index.

Step 2: Now use if conditional statement to check whether the given statement is equal to the elements in array.

Step 3: If the condition in step 2 satisfies, return the index no. of the given array. If condition dosen't satisfy then get out of loop.

Step 4: Now initialize a variable to enter element in the array form user. Now use split() method and to split the values.

Unsorted array.

```
def linear(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
inp = input("Enter elements in array:").split()
array = []
for ind in inp:
    array.append(int(ind))
print("Elements in array are:", array)
x1 = int(input("Enter the elements to be searched:"))
x2 = linear(array, x1)
if x2 == x1:
    print("Element found at location", x2)
else: print("element not found")
```

```
>>> Enter elements in array 3 2 4 5 1
>>> Elements in array are: [3 2 4 5 1]
>>> Element to be searched: 4
Element found at location 2
```

Step 5: Now initialize a variable as array i.e empty.

Step 6: Now use for conditional statement to append the elements given as input by user in the empty array

Step 7: Now again initialize another variable to ask user to find the element in the array.

Step 8: Again initialize a variable to call the defined function.

Step 9: Use if conditional statement to check if variable in step 8 matches with the element you want to find then print the index corresponding to element. If the condition dosen't satisfy then print that element is not found.

19/12/19

```
a = [ ]
print ( " enter 'n' to quit")
print ( " enter the element of the array:")
while ( True):
    x = input( )
    if (x. casefold () == 'n'):
        Break
    else:
        a. append (int(x))

a =sorted (a)
print (" The seabed that is", a)
y = int ( input ("enter the element to be searched"))
lb = 0
ub = len(a)
m = (ub + lb) /2
while (True)
    if (y == a [int(m)]
        print (" the number "o" was found at position
        {1}. format. int (m) +1))

    Break
    elif(y > = a[int(m)]
        lb =m+1
        m = ( ub+lb) / 2
    elif y not in a:
        print (" The no. was not found"))
        Break
```

8

output :-

Enter 'n' to quit

Enter the element of the array

21
23
53
n
26
76
n

The sorted list is [12, 21, 23, 26, 53, 76]

Enter the element to be searched : 21

The no. '21' was found at position 1

```
a = [ ]
print("enter integer value for the list and n to
      stop entering :")

while(True):
    x = input()
    if(x.casefold(x) == 'n'):
      Break
else:
    a.append(int(n))
print("The unsorted list is", a)
For i in range (len(a) -1(1):
    for j in range (len(a) -1-i);
    if(a[i] > a[j] + 1):
        a[i], a[j+1] = a[i+1], a[i]
    print("the sorted list is", a
```

output - enter the integer value for list type

5
12
32
123
100
99
69
n

The unsorted[5, 12, 32, 123, 100, 99, 69]

The sorted [5, 12, 32, 69, 99, 100, 123]

Bubble Sort

Partical No 3

Aim: To sort a list in average order using bubble sort algorith.

Step - 1 - Define an empty list use the while conditional statement to input as many value as the list.

Step - 2 Print the unsorted list, using the for conditional statement using intable 'i' in range len (list) or iterable that from 0 to len(list) - 1

step - 3 - Use the if conditional to check greater than element Swap the value if true.

step - 4 - Terminate the program

Practical No. 4

Aim: Implementation of stacks using python list.

Theory: A stack is a linear data structure that can be represented in real world in form of a physical stack, or a pile. The element in the stack are added or removed only from one position. Thus the stack works in LIFO (Last In First Out) principle as that element that was inserted last will be removed first. A stack can be implemented using array as well as linked list. Stack has 3 basic operation. (Push, Pop, Peek) The operations of adding and removing the elements is known as push and 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 method 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.

Code:

```
print("Ayaan Hyder")
class stack:
    global tos
    def __init__(self):
        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 empty")
        else:
            k=self.L[self.tos]
            print("data=",k)
            self.tos=self.tos-1
s=stack()
```

A. Ayaan Hyder

```
>>> s.push(10)
>>> s.push(20)
>>> s.push(30)
>>> s.push(40)
>>> s.push(50)
>>> s.push(60)
>>> s.push(70)
>>> s.push(80)
  Stack is full
>>>
>>> s.pop()
  data = 70
>>> s.pop()
  data = 60
>>> s.pop()
  data = 50
>>> s.pop()
  data = 40
>>> s.ppp()
  data = 30
>>> s.pop()
  data = 20
>>> s.pop()
  data = 10
>>> s.pop()
  Stack empty.
```


Step 5: Or else print statement as insert the element into the stack and initialize variable.

Step 6: Push method used to insert the element but pop method used to delete the element from stack ~~at topmost~~ position.

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 9: Assign the element value in push method & print and print the given value is poped

Step 10: Attach the input and output of above algorith.

Step 8: First condition checks whether the no. of elements are zero while the second case checks whether tos is assigned any value. If tos is not assigned any value then they can be stored as stack is empty.

m
02/11/2020

Practical No 5

Aim - Implement quick sort to sort the given list.

Theory - The quick sort is a recursive algorithm based on the divide and conquer rule.

Algorith -

1. Quick sort firsts selects a value which is called pivot value first value element. Serve as our first pivot 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 same time move other items to the appropriate side of the list either list then on greater than pivot value.

3. Partitions begin by locating two position makes lets call them left mark & right mark at the partition process is to move items then are on to the wrong side with respect to first value which use converging on the split point.

4. We begin by increasing leftmark until use locate is value that is greater than p.v value we then decrease value until we find value that is less.

code

```
def quicksort(a list):
    quick sort Helper(a list, 0, len(a list)-1)
def quicksort Helper(a list, first, last):
    if first < last:
        splitpoint = partition(a list, first, last)
        quicksort Helper(a list, splitpoint+1, last)
def partition(a list, first, last):
    pivot value = a list[first]
    leftmark = first+1
    rightmark = last
    done = false
    while not done:
        while leftmark <= rightmark and a list[leftmark]
    <= pivotvalue:
        leftmark = leftmark+1
    while a list[right mark] >= pivotvalue and
    rightmark >= leftmark:
    rightmark = rightmark-1
    if rightmark < leftmark:
        done = True
    else:
    temp = alist[leftmark]
    alist[leftmark] = alist[rightmark]
    alist[rightmark] = temp
    temp = alist[first]
    alist[first] = alist[rightmark]
    alist[rightmark] = temp
    temp = alist[first]
```

```
alist[first] = alist[rightmark]
    return rightmark
alist = [ 92, 54, 67, 89, 55, 11, 80, 100]
  quicksort(ist)
  print(list)
```

```
s.push(10)
s.push(20)
s.push(30)
s.push(40)
s.push(50)
s.push(60)
s.push(70)
s.push(80)
s.pop()
s.pop()
s.pop()
s.pop()
6.pop()
5.pop()
5.pop()
```

output:

Stack is full

data = 70

data = 60

data = 50

data = 40

data = 30

data = 20

data = 10

Stack is empty.

5. Or else print statement as insert the element into stack and initialize the values.

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

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

8. 1st condition checks whether the no. of elements is 0 which the second case whether to S is assigned any value, then we can be sure that stack is empty.

9. Assign the element values in push method and add and print the value is poped most.

10. Attack the input and output of above algorith.

Title: Implementing a queque using python list.

Theory: Queque is a linear data structure which has 2 references front and rear. Implementing a queque using python list is the simplest as the python list provides inbuilt functions to perform the specified operations of the queque.

Queque(): Creates a new empty queue

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

Dequeque(): Returns the element which was at the front, the front is moved to the successive elements.

Output:

```
>>> Q.add(30)
>>> Q.add(40)
>>> Q.add(60)
>>> Q.add(70)
>>> Q.add(80)
>>> Q.add(90)
  queque is ful
>>> 0.
```

Code:


```
class Queue:
    global r
    global f
    def __init__(self):
        self r = 0
        Self f = 0
        Self e = [0,0,0,0,0,0]
    def add(self. data):
        n = len(self.l)
        if Self.r < n-1:
        Self.l[self.r] = data
        Self.r = Self.r+1
        else:
            print("Queue is ful")
    def remove(self):
        n = len(self.l)
        if self.f < n-1:
            Print(self.l[self.f])
            self.f = self.f+1
        else:
            print("Queue is empty")
    Q = Queque().
```

Algorith :

1. Define a class Queque and assign global variable then (define __init__) method with self assignment in init() assign or initialize the initial value with the help of self argument.

2. Define a empty list and define enqueue() method with 2 arguments assign the length of empty line.

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.

Define Queue with self argument. Use if statement that front is equal to length of list then display Queue is empty or else give that front is at 0 and using that delete element from front side and increment by 1.

Now call the Queue() function and give the element that has to be added in the empty list by using enqueue() and print list after adding and same for deleting.

Practical 7

Aim: Program on Evaluation of given string by using in python environment i.e Postfix.

Algorithm :-

1. Define evaluate as function then create a empty stack in pyth
2. Convert the string to a list by using string method 'split'.
3. Calculate the length of string and print it.
4. Use the loop to assign the range of string then give condition using if statement.
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'.
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
7. Perform the arithmetic operation. Push the result back on the 'm'.

* 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.appened(int(b) + int(a))
        elif k[i] == '-':
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) - int(a))
        elif k[i] == '*':
            a = stack.ppp()
            b = stack.pop()
            stack.appent(int(b) * int(a))
        else:
            a = stack.pop()
            b = stack.pop()
            stack.append(int(b) / int(a))
    return stack.pop()
    s = '8194+'
    r = evaluate(s)
    print("The evaluated value is", r)
```

When the input expression has been completely processed the result is on the value.

Print the result of string after the evaluation of postfix.

Attach output and input of above algo.

Practical : 8

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

Algorithm :

1. Traversing of a linked list means positioning all the nodes in the linked list in order to perform some operations on them.

2. The entire linked list can be accessed as the first node of the linked list.

3. Thus the entire linked list can be traversed using the node which is refered by the head pointer of the linked list.

4. Now that we know that we can traverse the entire linked list using the head pointer, we should only use it to refer the first node of the list only.

5. We should not use the head pointer to traverse the entire linked list because head pointer is our only reference to first node.

6. You may lose reference to first node in our linear list and hence most of our linked list. To avoid this we will make use of temporary node to terminate entire linked list.

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)

start = linkedlist()

# output:-
>>> start.add_L(30)
>>> start.add_L(70)
>>> start.add_L(60)
>>> start.add_L(50)
>>> start.add_L(40)
>>> start.add_B(30)
>>> start.add_B(20)
>>> start.display()
20
30
40
50
60
70
80
```


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

8. Now that current is refering to the first node. If we want to access second node of list we need to refer it as next node of 1st node.

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

10. Similarly, we can traverse rest of the nodes in the linked list.

11. Our concern is now to find terminating condition for the while loop.

12. The last node in the linked list is refered by the tail of linked list.

13. So we can refer to last node of linked list.

14. We have to now see how to start traversing the linked list and how to identify whether we have reached the last node.

Practical : 9

Aim: Implementation of mergesort by using python.

Theory:-

Algorithm:

Step 1: The list is divided into left and right in each recursive call until two adjacent element are obtain.

Step 2: Now begin the sorting process then use if iteration traverse the two halfes value in each then k iteration transverse the whole list and value change alongside.

Step 3: Do the value of is smaller than the value of j(i) is sorting to the arr [i+1] and if increasing if first R[j] is sorting.

Step 4: This way the value being assigned through h(i+1) are all sorted.

Step 5: At the end of the loop. One of the values may not have become traversed completely remain slot at the list.

Step 6: Thus merging sort has been completed.

Code:

```
def sort ( arr: L.M.R )
    n1. m1. L+1
    n2. r-m
    L = [0] * (n1)
    R = [0] * (n2)
for i in range (0, n1):
        L[i] = arr [i+1]
for j in range (0,42):
        R[i] = arr[m+1+j]

i = 0
j = 0
k = L
while i<n1 and j<n2 :
        if L[i] < = R[i]:
        arr[k] = L[i]

    else :
        arr [k] . R[i]
        j = j+1
        k = k+1
        i = k+1.
```

```
        arr[k] = L[i]
        i = i+1
        k = k+1
    while i<n2
        arr[k] = R[i]
        j+=1
        k=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 = [12, 23, 34, 56, 78, 45, 86, 98, 42]
print(arr)
n = len(arr)
mergesort(arr, 0, n-1)
print(arr)
```

output:-

[12, 23, 86, 56, 48, 45, 86, 98, 42]

[12, 23, 36, 56, 42, 45, 78, 96, 98]

Practical 10.

Aim :- Implementation of set by using Python.

Algorith :-

Step 1 : Define two empty set as at 1 and 2. Now user state provide their range of 2 sets.

Step 2 : Now add() method is used for addition two element according to give range then print the sets for addition.

Step 3 : Find the union and intersection of above 2 st by using 1 method print the set by union and intersection of 3.

Step 4 : Use if statement to find out the subset and superset of at 3 and 4 Display Set

Step 5 : Display that element in at 3 is using mathemat operation.

Step 6 : Use clear() to remove an delete the sets and print the set after clearing the element present in the set.

Code :

```
Set 1 = set()
Set 2 = set()
for i in range (8,15)
        Set 1 add (i)
print( " Set 1 : " Set 1)
print (Set 2 :4 " Set 2)
print ( "\n")
Set 3 = (set 1 / Set 2)
print(" Union of set 1 and Set 2 : Set 3", Set 3)
print( " \n")
if Set 3 > Set 4 :
    print( " Set is superset of set 4")
elif set 3 < Set 4 :
        print( ' Set 3 is subset of set4")
else :
    print( " Set 3 is same as Set 4")
if Set 4 < set 3 :
        print ( " Set 4 is subset of set 3")
        print ( " \n")
Set 1. Set 3. Set 4
```

```
print("element in set 5 and list in set 4
      set 4, set)
print("\n")
if set 4 is disjont (set 5).
. print (" set 4 and set 5 are mutually e
set clear()
print(" After applying exclusive")
print(" set 5 4, set 5)
```

output :-

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

set 2 :- { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

union of set 1 and set 2 : str 3

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

intersection of set 1 and set 2

{ 8, 9, 10, 11} set 3 is superset

after applying clear set 5 = set().

Practical 11 :

Aim : Implementation of Binary Search tree using Python and Inorder, preorder and postorder, traversal.

Theory : Binary tree is a tree which supports maximum of children for any node within the tree. Thus any particular node can have 0, 1, 2 children. This is another denity of binary tree that: It is ordered such that one child is identified as left child and other as right.

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

ii. Preorder (i) Visit the root node.
(ii) Traverse the left subtree. The left subtree inturn have left & right.
(iii) Traverse the right subtree repeat it.

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

Algorith

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 then put on assange that in leftside.

Step 5: Use whileloop for checking the node is less than or greater than the main node and break the loop if it is not satisfying.

Step 6: After this, left subtree and rightsubtree repeat the method to arrange the node according to BST.

Step 7: Define Inorder(), preorder() and postorder()

Step 8: In inorder, else statement used for giving that condition first left, root and then right root.

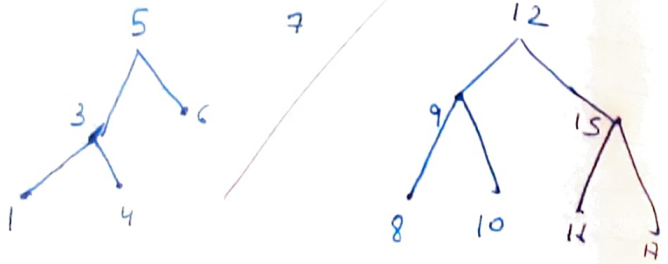
Step 9: for preorder, we have to give condition in else first left node then right node.

Step 10: for postorder. In else part assign left then right and then go for root node.

Step 11: Display the output and input.

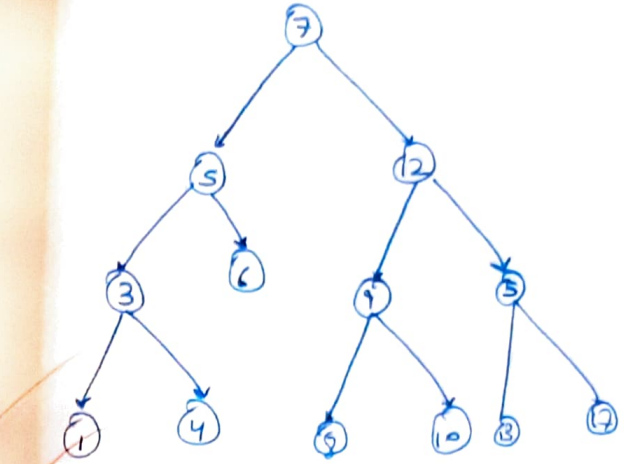
Inorder: (LVR)

Step 1



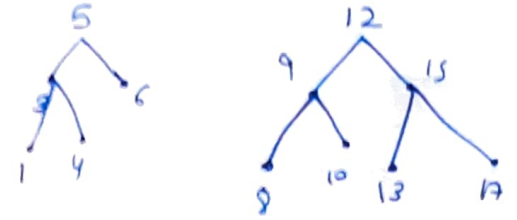
Step 2

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



Preorder (VLR)

Step 1 : 7

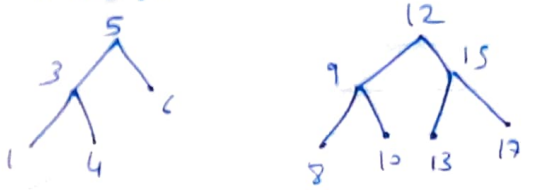


Step 2 : 7 5 3 6 12 9 15

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

Post Order : (LRV)

Step 1



Step 2 : 3 6 5 9 15 12 7

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