**Linear Search**

list = [10, 20, 30, 40, 50]

x = 40

for i in range(len(list)):

if list[i] == x:

print("Element found at position :", i)

break

else:

print("Element not found")

**Bubble Sort**

list = [10,15, 4, 23, 0]

print("unsorted list is:", list)

for j in range(len(list)-1):

for i in range(len(list)-1):

if list[i]>list[i+1]:

temp = list[i]

list[i] = list[i+1]

list[i+1] = temp

print("sorted list is:", list)

**Selection Sort**

list = [10,15, 4, 23, 0]

print("unsorted list is:", list)

for i in range(len(list)):

min\_idx = i

for j in range(i+1, len(list)):

if list[j] < list[min\_idx]:

min\_idx = j

temp = list[i]

list[i] = list[min\_idx]

list[min\_idx] = temp

print("sorted list is:", list)

**Insertion Sort**

def insertionSort(list):   
 for i in range(1, len(list)):

key = list[i]

j = i– 1

while j >= 0 and key < list[j]:

list [j + 1] = list [j]

j = j – 1

list [j + 1] = key

list = [9, 5, 1, 4, 3]

insertionSort(list)

print('Sorted Array in Ascending Order:')

print(list)

**Merge sort**

def mergesort(list):

if len(list) > 1:

mid = len(list)//2

left\_list = list[ 0 : mid ]

right\_list = list[mid:]

mergesort(left\_list)

mergesort(right\_list)

i=0

j=0

k = 0

while i < len(left\_list) and j < len(right\_list) :

if left\_list [ i ] < right\_list [ j ]:

list[k] = left\_list [ i ]

i = i + 1

k = k + 1

else :

list[k] = right\_list[j]

j = j + 1

k = k + 1

while i < len(left\_list):

list[k] = left\_list [ i ]

i = i + 1

k = k + 1

while j < len(right\_list):

list[k] = right\_list[j]

j = j + 1

k = k + 1

list = [ 10 , 2, 12, 7, 3 ]

print(“ The unsorted list is :”, list)

mergesort(list) # call to mergesort

print(“ The sorted list is :”, list)

**Quick sort**

def partition(array, low, high):

pivot = array[high]

i = low-1

for j in range(low, high):

if array[j] <= pivot:

i = i + 1

(array[i],array[j]) = (array[j],array[i])

(array[i + 1],array[high]) = (array[high],array[i + 1])

return i + 1

def quick\_sort(array, low, high):

if low < high:

pi = partition(array, low, high)

quick\_sort(array, low, pi - 1)

quick\_sort(array, pi + 1, high)

array = [10, 7, 8, 9, 1, 5]

quick\_sort(array, 0, len(array) - 1)

print("Sorted arrar is:", array)

**Binary Search**

def binarySearch(array, x, low, high):

if high >= low:

mid = (low+high)//2

if array[mid] == x:

return mid

elif array[mid] > x:

return binarySearch(array, x, low, mid-1)

else:

return binarySearch(array, x, mid + 1, high)

else:

return -1

array = [3, 4, 5, 6, 7, 8, 9]

x = 4

result = binarySearch(array, x, 0, len(array))

if result != -1:

print("Element is present at index " + str(result))

else:

print("Not found")

**singly linked list.**

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

class LinkedList:

     def \_\_init\_\_(self):

         self.head = None

def printList(self):

temp = self.head

while (temp):

print (temp.data,end=” ”)

temp = temp.next

node1 = Node('Mon')

node2 = Node('Tues')

node3 = Node('Wed')

llist = LinkedList()

llist.head = node1

llist.head.next = node2;

node2.next =node3;

print(“The linked list is:”)

print(“\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*”)

llist. printList()

**insert and delete new nodes at the beginning of linked list.**

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

class LinkedList:

     def \_\_init\_\_(self):

         self.head = None

def insert(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

self.head = new\_node

def delete(self):

self.head = self.head.next

def printList(self):

temp = self.head

while (temp):

print (temp.data)

temp = temp.next

node1 = Node('Mon')

node2 = Node('Tues')

node3 = Node('Wed')

llist = LinkedList()

llist.head = node1

llist.head.next = node2;

node2.next =node3;

print(“The Original linked list is:”)

print(“\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*”)

llist. printList()

print(“After Insertion:”)

print(“\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*”)

llist.insert(‘Sun’)

llist. printList()

print(“After Deletion:”)

print(“\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*”)

llist.delete()

llist. printList()

**Fibonacci**

**def** fib(n):  
 **if** (n==0):  
 **return** 0  
 **if** (n==1 **or** n==2):  
 **return** 1  
 **else**:  
 **return** fib(n-1)+fib(n-2)  
n=int(input(**'Enter an Integer:'**))  
print(**'The Fibonacci sequence of given integer is:'**,end=**""**)  
**for** i **in** range(0,n):  
 print(fib(i),end=**","**)

**doubly**

**class** node:  
 **def** \_\_init\_\_(self,data):  
 self.prev=**None** self.next=**None** self.data=data  
**class** dlist:  
 **def** \_\_init\_\_(self):  
 self.head=**None  
 def** printlist(self):  
 print(**"\n Trversal in forward direction"**)  
 node=self.head  
 **while**(node **is not None**):  
 print(node.data)  
 last=node  
 node=node.next  
 print(**"\n Traversal is reverse direction"**)  
 **while** last:  
 print(last.data)  
 last=last.prev  
node1=node(**'Mon'**)  
node2=node(**'Tues'**)  
node3=node(**'Wed'**)  
dlist=dlist()  
dlist.head=node1  
dlist.head.prev=**None**dlist.head.next=node2  
node2.prev=dlist.head  
node2.next=node3  
node3.prev=node2  
print(**'The doubly linked list is:'**)  
print(**'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'**)  
dlist.printlist()

**fact**

**def** factorial(n):  
 **if**(n==0 **or** n==1):  
 **return** 1  
 **else**:  
 **return** n\*factorial(n-1)  
n=int(input(**'Enter an Integer:'**))  
result=factorial(n)  
print(**'The factorial of given integer is:'**,result)

**queue**

queue=[]  
**def** enqueue():  
 item=input(**"Enter the item:"**)  
 queue.append(item)  
 print(item,**'is inserted to queue'**)  
**def** dequeue():  
 **if not** queue:  
 print(**'Underflow (empty Queue)'**)  
 **else**:  
 item=queue.pop(0)  
 print(**'Deleted item='**,item)  
**def** display():  
 print(queue)  
**while True**:  
 print(**"Select the operation 1.Insert 2.Delete 3. Display 4.Quit \n"**)  
 choice=int(input())  
 **if** choice==1:  
 enqueue()  
 **elif** choice==2:  
 dequeue()  
 **elif** choice==3:  
 display()  
 **elif** choice==4:  
 **break  
 else**:  
 print(**'Enter Correct operation'**)

**stack**

stack=[]  
**def** push():  
 **if** len(stack)==n:  
 print(**"Overview (stack is full)"**)  
 **else**:  
 item=int(input(**"Enter item to push:"**))  
 stack.append(item)  
 print(stack)  
**def** pop():  
 **if not** stack:  
 print(**"Underflow (empty stack)"**)  
 **else**:  
 item=stack.pop()  
 print(**"Popped element="**,item)  
 print(stack)  
n=int(input(**"Enter stack limit:"**))  
**while True**:  
 print(**'Select the operation 1.Push 2.Pop 3.Quit'**)  
 choice=int(input())  
 **if** choice==1:  
 push()  
 **elif** choice==2:  
 pop()  
 **elif** choice==3:  
 **break  
 else**:  
 print(**'Enter Correct Operation'**)

**circularlinkedlist**

**class** node:  
 **def** \_\_init\_\_(self,data):  
 self.data=data  
 self.next=**None  
class** circularlinkedlist:  
 **def** \_\_init\_\_(self):  
 self.head=**None  
 def** printlist(self):  
 temp=self.head  
 **if** self.head **is not None**:  
 **while**(**True**):  
 print(temp.data,end=**"->"**)  
 temp=temp.next  
 **if**(temp==self.head):  
 **break**clist=circularlinkedlist()  
node1=node(**'Mon'**)  
node2=node(**'Tues'**)  
node3=node(**'Wed'**)  
node4=node(**'Thur'**)  
clist.head=node1  
clist.head.next=node2  
node2.next=node3  
node3.next=node4  
node4.next=clist.head  
print(**'The circular linked list is:'**)  
print(**'\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*'**)  
clist.printlist()

**Tree**

**class** Node:  
 **def** \_\_init\_\_(self,data):  
 self.data=data  
 self.left=**None** self.right=**None  
class** BinarySearchTree:  
 **def** \_\_init\_\_(self):  
 self.root=**None  
 def** insert(self,value):  
 newNode=Node(value)  
 **if** self.root **is None**:  
 self.root=newNode  
 **else**:  
 curNode=self.root  
 **while** curNode **is not None**:  
 **if** value<curNode.data:  
 **if** curNode.left **is None**:  
 curNode.left=newNode  
 **break  
 else**:  
 curNode=curNode.left  
 **else**:  
 **if** curNode.right **is None**:  
 curNode.right=newNode  
 **break  
 else**:  
 curNode=curNode.right  
 **def** preorder(self,rt):  
 print(rt.data,end=**" "**)  
 **if** rt.left **is not None**:  
 self.preorder(rt.left)  
 **if** rt.right **is not None**:  
 self.preorder(rt.right)  
 **def** postorder(self,rt):  
 **if** rt.left **is not None**:  
 self.postorder(rt.left)  
 **if** rt.right **is not None**:  
 self.postorder(rt.right)  
 print(rt.data,end=**" "**)  
 **def** inorder(self,rt):  
 **if** rt.left **is not None**:  
 self.inorder(rt.left)  
 print(rt.data,end=**" "**)  
 **if** rt.right **is not None**:  
 self.inorder(rt.right)  
bst=BinarySearchTree()  
ls=[25,10,35,20,65,45,24]  
**for** i **in** ls:  
 bst.insert(i)  
print(**"\nPre-order"**)  
bst.preorder(bst.root)  
print(**"\nPost-order"**)  
bst.postorder(bst.root)  
print(**"\nIn-order"**)  
bst.inorder(bst.root)