l=len(a)

for i in range(l,0,-1):

print(a[i])

class Node:

def init(self,data):

self.left=None

self.right=None

self.data=data

class bst: #To create a tree

def init(self):

self.root=None

def insert(self,data,root):

if root is None:

return Node(data)

if data<root.data:

root.left=self.insert(data,root.left)

elif data>root.data:

root.right=self.insert(data,root.right)

return root

def inorder\_traversal(self,root):

if root:

self.inorder\_traversal(root.left)

print(root.data,end=" ")

self.inorder\_traversal(root.right)

def preorder\_traversal(self,root):

if root:

print(root.data,end=" ")

self.preorder\_traversal(root.left)

self.preorder\_traversal(root.right)

def postorder\_traversal(self,root):

if root:

self.postorder\_traversal(root.left)

self.postorder\_traversal(root.right)

print(root.data,end=" ")

bst\_tree=bst()

root=None

root=bst\_tree.insert(20,root)

root=bst\_tree.insert(10,root)

root=bst\_tree.insert(900,root)

bst\_tree.inorder\_traversal(root)

bst\_tree.preorder\_traversal(root)

bst\_tree.postorder\_traversal(root)

class Node:

def init(self,data):

self.left=None

self.right=None

self.data=data

class bst: #To create a tree

def init(self):

self.root=None

def insert(self,data,root):

if root is None:

return Node(data)

if data<root.data:

root.left=self.insert(data,root.left)

elif data>root.data:

root.right=self.insert(data,root.right)

return root

def inorder\_traversal(self,root):

if root:

self.inorder\_traversal(root.left)

print(root.data,end=" ")

self.inorder\_traversal(root.right)

def preorder\_traversal(self,root):

if root:

print(root.data,end=" ")

self.preorder\_traversal(root.left)

self.preorder\_traversal(root.right)

def postorder\_traversal(self,root):

if root:

self.postorder\_traversal(root.left)

self.postorder\_traversal(root.right)

print(root.data,end=" ")

def search(self,root,key):

if root.data==key:

return True

if root is None:

return False

def search(self,root,key):

if(key<root.data):

return self.search(root.left,key)

elif(key>root.data):

return self.search(root.right,key)

else:

return True

bst\_tree=bst()

root=None

root=bst\_tree.insert(20,root)

root=bst\_tree.insert(10,root)

root=bst\_tree.insert(900,root)

bst\_tree.inorder\_traversal(root)

print("\n inorder\_traversal")

bst\_tree.preorder\_traversal(root)

print("\n preorder\_traversal")

bst\_tree.postorder\_traversal(root)

print("\n postorder\_traversal")

print("\n Search\_in\_bst")

print(bst\_tree.search(root,10))

class Node:

def init(self,data):

self.left=None

self.right=None

self.data=data

class bst: #To create a tree

def init(self):

self.root=None

def insert(self,data,root):

if root is None:

return Node(data)

if data<root.data:

root.left=self.insert(data,root.left)

elif data>root.data:

root.right=self.insert(data,root.right)

return root

def inorder\_traversal(self,root):

if root:

self.inorder\_traversal(root.left)

print(root.data,end=" ")

self.inorder\_traversal(root.right)

def preorder\_traversal(self,root):

if root:

print(root.data,end=" ")

self.preorder\_traversal(root.left)

self.preorder\_traversal(root.right)

def postorder\_traversal(self,root):

if root:

self.postorder\_traversal(root.left)

self.postorder\_traversal(root.right)

print(root.data,end=" ")

def search(self,root,key):

if root.data==key:

return True

if root is None:

return False

def search(self,root,key):

if(key<root.data):

return self.search(root.left,key)

elif(key>root.data):

return self.search(root.right,key)

else:

return True

def Height(self,root):

if root is None:

return 0

else:

return max(self.Height(root.left),self.Height(root.right))+1

def sum\_of\_roots(self,root):

if root is None:

return 0

return root.data + self.sum\_of\_roots(root.left)+self. sum\_of\_roots(root.right)

def count\_root(self,root):

if root is None:

return 0

return 1+self.count\_root(root.left)+self.count\_root(root.right)

bst\_tree=bst()

root=None

root=bst\_tree.insert(20,root)

root=bst\_tree.insert(10,root)

root=bst\_tree.insert(900,root)

bst\_tree.inorder\_traversal(root)

print("\n inorder\_traversal")

bst\_tree.preorder\_traversal(root)

print("\n preorder\_traversal")

bst\_tree.postorder\_traversal(root)

print("\n postorder\_traversal")

print("\n Search\_in\_bst")

print(bst\_tree.search(root,10))

print("\n Height of BST")

print(bst\_tree.Height(root))

print("\n sum of root")

print(bst\_tree.sum\_of\_roots(root))

print("\n count\_root")

print(bst\_tree.count\_root(root))

DAY-9

def binary\_search(arr,key):

start=0

end=len(arr)-1

mid=(start+end)//2

while(start<=end):

if (arr[mid]==key):

return mid

elif(key>arr[mid]):

start= mid+1

else:

end=mid-1

mid=(start+end)//2

return -1

arr=[4,5,6,7,8,9,10,11,12]

key=7

result=binary\_search(arr,key)

print(result)

class Solution:

def binarysearch(self, arr, k):

start, end = 0, len(arr) - 1

result = -1

while start <= end:

mid = (start + end) // 2

if arr[mid] == k:

result = mid

end = mid - 1

elif arr[mid] < k :

start = mid + 1

else:

end = mid - 1

return result

def quick\_sort(arr):

if len(arr)<=1:

return arr

left=[]

right=[]

equal=[]

pvt=arr[-1]

for i in arr:

if(i<pvt):

left.append(i)

elif(i>pvt):

right.append(i)

else:

equal.append(i)

print("pvt",pvt)

print("left sub array",left)

print("right sub array",right)

print("equal array",equal)

return quick\_sort(left)+equal+quick\_sort(right)

arr=[23,63,44,57,12,45,36]

sorted\_arr=quick\_sort(arr)

print(sorted\_arr)

def insertion\_sort(arr):

for i in range(1,len(arr)):

key=arr[i]

j=i-1

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

arr[j+1]=arr[j]

j-=1

arr[j+1]=key

return arr

arr=[8,3,14,9]

sorted\_arr=insertion\_sort(arr)

print(arr)

def bubble\_sort(arr):

n=len(arr)

for i in range(n):

for j in range(0,n-i-1):

if arr[j]>arr[j+1]:

arr[j],arr[j+1]=arr[j+1],arr[j]

return arr

arr=[8,3,45,9]

sorted\_sort=bubble\_sort(arr)

print(arr)

DAY-10

[27-05-2025 10:02]

arr=list(map(int,input().split()))

arr.sort()

a1=arr[-1]

a2=arr[-2]

sum=0

avg=(a1+a2)/2

for i in range(len(arr)):

if arr[i]<avg:

arr[i]=0

for i in range(len(arr)):

sum+=arr[i]

print(sum)

[27-05-2025 10:40]

r=7

unit=2

arr=[2, 8, 3, 5, 7, 4, 1, 2]

food\_req=r\*unit

for i in range(len(arr)):

food\_req=food\_req-arr[i]

if (food\_req<0):

break

print(abs(food\_req))

[27-05-2025 11:47] Mouni💖: # Enter your code here

s=input()

dc=s[5:9]

s1=s[0:5]

if(len(s)==10 and s[-1].isalpha() and dc.isdigit() and s1.isalpha()):

print("valid")

else:

print("invalid")

[27-05-2025 12:04]

n = int(input())

m = int(input())

count\_divisible = 0

count\_not\_divisible = 0

for i in range (1,m+1):

if(i%n!=0):

count\_not\_divisible+=1

else:

count\_divisible+=1

print(abs( count\_not\_divisible-count\_divisible))

[27-05-2025 13:56]

for i in range(4):

for j in range(4):

if(i==j):

print("$",end=" ")

else:

print("\*",end=" ")

print()

[27-05-2025 14:31]

for i in range(1,5):

for j in range(1,5):

if i == 1 or i == 4 or j == 1 or j == 4:

print("\*", end=" ")

else:

print(" ", end=" ")

print()

[27-05-2025 14:34]

for i in range(1,5):

for j in range(1,5):

if i == 1 or i == 4 or j == 1 or j == 4:

print(j+ (i-1),end=" ")

else:

print(" ", end=" ")

print()

[27-05-2025 14:42]

for i in range(1,5):

for j in range(1,5):

if (i<j):

print(" ", end=" ")

else:

print("\*",end=' ')

print()

[27-05-2025 15:23]

for i in range(1,5):

for j in range(1,5):

if (i>j):

print(" ", end=" ")

else:

print("\*",end=' ')

print()

for i in range(1,5):

for j in range(1,5):

if (i<j):

print(" ", end=" ")

else:

print("\*",end=' ')

print()

for i in range(1,5):

for j in range(1,5):

if (i+j>=5):

print("\*", end=" ")

else:

print(" ",end=' ')

print()

for i in range(1,5):

for j in range(1,5):

if (i+j<=5):

print("\*", end=" ")

else:

print(" ",end=' ')

print()

for i in range(4,0,-1):

print("\*"\*i)

for i in range(1,5):

for j in range(1,5):

if (i<j):

print(" ", end=" ")

else:

print(j,end=' ')

print()