**DAY:1**

**Time Complexity:** How much time it takes to complete a task.

Ex:

If marks<90: [Best Case Scenario]

A++

Elif marks (80-90): [Average Case Scenario]

A

Elif marks(60-70): [Worst Case Scenario]

B

Ex: for(int I=1;i<=n;i++){ } - O(n)

Ex: for(int I=1;i<=3\*n;i++){ } - O(n)

Ex: for(int I=1;i<=n;i++){ } - O(n)

Ex: for(int I=1;i<=n;i)

{

for(j=1;j<=n;j++){ }

} - O(n2)

**Note:** The time complexity doesn’t depend on the n value in the for loop.

**General Time Complexities to remember:**

For loop: O(n)

Nested loop: O(nx), x=based on the number of nested loops

Sorting: O(nlogn)

Recurssion: O(2n)

Binary Search: O(logN)

**Ex: Power of Two**

Time Complexity: O(nlogn)

class Solution:

    def isPowerOfTwo(self, n: int) -> bool:

        if n==0:

            return False

        while n:

            if n==1:

                return True

            if n%2!=0:

                return False

            n=n//2

        return False

Time Complexity: O(n)

class Solution:

    def isPowerOfTwo(self, n: int) -> bool:

        if n==0:

            return False

        if n&n-1==0:

            return True

        else:

            return False

**Difference between Array and List:**

List stores the references(address) in a sequential manner whereas arrays stores the values in the sequential manner.

**Why indexing starts from 0:**

Accessing of an element= Base address + offset \* size of datatype

Offset= how many blocks away is a specific element from the first element

It generates the ending address of the next element using this formula if index starts from 1, that’s why indexing starts from 0.

**Left Shift:** 10<<2

Formula: x<<y=x\*2y

Right Shift: 10>>2

Formula: x>>y=x/2y

**Pseudocode:**

l=[1,2,3,4]

for i in l:

l.remove(i)

print(l)

Output: [2,4]

Explaination: the value of i gets incremented after every iteration

**Pseudocode:**

l=[1,2,3,4,5,6,7]

l.append([1,2,3])

l.extend([1,2])

l.append(1)

l.extend({1,2,3,4,2,2,1,3})

print(l)

print(len(l))

Output: 15

**Pseudocode:**

Time Complexity: O(n)

n=5

x=0

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

x^=i

print(x)

Optimized Code: O(1)

n=5

if n%4==1:

print("1")

if n%4==2:

print(n+1)

if n%4==3:

print("0")

if n%4==0:

print(n)

**Q1. PseudoCode (XOR in a range):**

Time Complexity: O(1)

def xor(n):

if n%4==1:

return 1

if n%4==2:

return n+1

if n%4==3:

return 0

if n%4==0:

return n

l,r=map(int,input().split())

a=xor(r)

b=xor(l-1)

print(a^b)

**Q2. Single Number:**

class Solution:

    def singleNumber(self, nums: List[int]) -> int:

        x=0

        for i in nums:

            x^=i

        return x

**Q3. Decoded XORed Array:**

class Solution:

    def decode(self, encoded: List[int], first: int) -> List[int]:

        a=[first]

        for i in encoded:

            a.append(a[-1]^i)

        return a

**Q4. Count of Prime**

n=int(input())

count=0

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

if n%i==0:

count+=1

print(count)

**Q5. Watermelon Problem**

w=int(input())

if w%2!=0:

print("No")

else:

x=w//2

if x%2==0:

print(x,x)

else:

print(x-1,x+1)

**Q6. An elephant decided to visit his friend. It turned out that the elephant's house is located at point 0 and his friend's house is located at point x(x > 0) of the coordinate line. In one step the elephant can move 1, 2, 3, 4 or 5 positions forward. Determine, what is the minimum number of steps he need to make in order to get to his friend's house.**

x=int(input())

s=0

if x<5:

print("1")

elif x%5==0:

print(x//5)

else:

print((x//5)+1)

**Q7. Input: 10,3 Output: 4 Explaination: (10%3)=1 → not divisible so (10//3)+(10%3)=3+1=4.**

n,m=map(int,input().split())

if n%m==0:

print(n//m)

else:

print((n//m)+(n%m))

**Q8. Bear Limak wants to become the largest of bears, or at least to become larger than his brother Bob. Right now, Limak and Bob weigh a and b respectively. It's guaranteed that Limak's weight is smaller than or equal to his brother's weight. Limak eats a lot and his weight is tripled after every year, while Bob's weight is doubled after every year. After how many full years will Limak become strictly larger (strictly heavier) than Bob?**

a,b=map(int,input().split())

y=0

while a<=b:

a\*=3

b\*=2

y+=1

print(y)

**Q9. Sum of even number in the odd indices of the list**

n,m,a,b=map(int,input().split())

if m\*a<b:

print(a\*n)

else:

print(((n//m)\*b) + min((n%m)\*a,b))

**Q10. Bitwise Operator**

a=5

b=7

print(a^b)

**DAY:2**

**Q1. Sum of even numbers in the odd indices**

l=[2,3,7,5,1,4,6,8,9]

s=0

for i in range(len(l)):

if i%2!=0:

if l[i]%2==0:

s+=l[i]

print(s)

**Q2. Building getting sunlight**

l=[2,3,4,3,2,3,5,4,2]

m=0

s=0

for i in l:

if i>m:

m=i

s+=1

print(s)

**Q3. Police Recruits**

n=int(input())

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

p=0

uc=0

for i in l:

if i==-1:

if p>0:

p-=1

else:

uc+=1

else:

p+=i

print(uc)

**Q4. Votes**

n=int(input())

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

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

c=max(v)\*[0]

for i in range(n):

if a[i]>=18:

c[v[i]-1]+=1

temp=sorted(c,reverse=True)

if temp[0]==temp[1]:

print("-1")

else:

print(c.index(temp[0])+1)

**Q5. Swap two numbers using XOR**

a,b=map(int,input().split())

a=a^b

b=a^b

a=a^b

print(a)

print(b)

**Q6. Odd and even no. using &**

n=int(input())

if n&1:

print("Odd")

else:

print("Even")

**Q7. Odd even no. using XOR**

n=int(input())

if n^1==n+1:

print("Even")

else:

print("Odd")

**Q8. 1^2^3^4^5**

n=5

x=0

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

x^=i

print(x)

**Q9. XOR of all integers from 1 to n**

n=5

if n%4==1:

print("1")

if n%4==2:

print(n+1)

if n%4==3:

print("0")

if n%4==0:

print(n)

**Q10. Decoded XORed Array**

def xor(n):

if n%4==1:

return 1

if n%4==2:

return n+1

if n%4==3:

return 0

if n%4==0:

return n

l,r=map(int,input().split())

a=xor(r)

b=xor(l-1)

print(a^b)

**DAY:3**

**Recurssion**

**Q1. Input: 5 Output: 543212345**

def fun(n):

if n==0:

return

print(n,end=" ")

fun(n-1)

if n!=1:

print(n,end=" ")

n=int(input())

fun(n)

**Q2. Input: 5 5 Output: 3135**

def fun(n):

if n==0:

return

if n%2!=0:

print(n,end=" ")

fun(n-1)

if n!=1:

if n%2!=0:

print(n,end=" ")

n=int(input())

fun(n)

**Q3. Input: 5 Output:123454321**

def fun(n,m=0):

if n==m:

return

print(m+1,end=" ")

fun(n,m+1)

if m!=5 and m>0:

print(m,end=" ")

n=int(input())

fun(n)

**Q4. Even Numbers from 1 to 100:**

def even(n,i=1):

if i!=n:

if i%2==0:

print(i,end=" ")

even(n,i+1)

even(100)

**Q4. Odd Numbers from 1 to 100**

def odd(n,i=1):

if i!=n:

if i%2!=0:

print(i,end=" ")

odd(n,i+1)

odd(100)

**Q5. factorial of a number**

def fact(n):

if n==1:

return n

else:

return n\*fact(n-1)

n=5

print(fact(n))

**Q6. Fibonacci Series- Time Complexity: O(2^n)**

def fib(n):

if n<=1:

return n

else:

return fib(n-1)+fib(n-2)

x=5

print(fib(x))

**Q7. Prime Number**

def pri(n,i=2):

if n<=1:

return False

if i\*i>n:

return True

if n%i==0:

return False

return pri(n,i+1)

n=29

if pri(n):

print("Prime")

else:

print("Non-Prime")

**Q8. Prime Number: Time Complexity- O(n)**

n=int(input())

flag=0

for i in range(2,n//2):

if n%i==0:

flag+=1

break

if flag==0:

print("Prime")

else:

print("Non-Prime")

**Q9. Prime Number :Time Complexity- O(1)**

n=int(input())

flag=0

for i in range(2,int(n\*\*(1/2))+1):

if n%i==0:

flag+=1

break

if flag==0:

print("Prime")

else:

print("Non-Prime")

**Q10. Reverse of Number**

def reverse(x,res=0):

if x<=0:

return res

rem=x%10

res=res\*10+rem

return reverse(x//10,res)

n=int(input())

print(reverse(n))

**Q11. Minimum steps to reduce a number to 1**

def h(n):

if n==1:

return 0

if n%2==0:

return 1+h(n//2)

return 1+min(h(n-1),h(n+1))

n=int(input())

print(h(n))

**Q12. Sum of numbers in a list**

def s(l):

if len(l)==0:

return 0

return l[0]+s(l[1:])

l=[1,2,3,4,5]

print(s(l))

**Q13. Count of Prime Numbers in a list**

def pri(n,i=2):

if n<=1:

return False

if i\*i>n:

return True

if n%i==0:

return False

return pri(n,i+1)

def fun(l,i=0):

if i==len(l):

return 0

if pri(l[i]):

return 1+fun(l,i+1)

else:

return fun(l,i+1)

l=[11,4,3,6,5]

print(fun(l))

**DAY:4**

**Difference between Subset and a Subsequence:**

Subset can be in any order whereas subsequence can only have one sequence.

**Ex:** a=[1,2,3]

**Subsets:** [],[1,2,3],[1,2,],[2,1],[2,3],[3,2],[1,3],[3,1],[1],[2],[3]

**Subsequence:** [],[1,2,3],[1,2],[2,3],[1,3],[1],[2],[3]

**Q1. Sum of n Natural Numbers**

def sum1(n):

if n==0:

return 0

return n+sum1(n-1)

n=5

print(sum1(5))

**Q2. Parameterized Recursion**

def fun(n,sum):

if n<1:

print(sum)

return

fun(n-1,sum+n)

n=5

fun(n,sum=0)

**Q3. Subsets of a list**

def sub(a,i=0,res=[]):

if i==len(a):

print(res)

return

sub(a,i+1,res+[a[i]]) #pick

sub(a,i+1,res) #not pick

l=[1,2,3]

sub(l)

**Q4. Subsets sum**

def sub(a,k,i=0,res=[]):

if i==len(a):

return

if sum(res)==k:

return True

else:

return False

sub(a,k,i+1,res+[a[i]]) #pick

sub(a,k,i+1,res) #not pick

l=[1,2,3,8]

k=7

print(sub(l,k))

**Q5. Subset sum optimized code**

def fun(a,i,k):

if k==0:

return True

if i==0:

return False

if a[i-1]>k:

return fun(a,i-1,k)

return fun(a,i-1,k) or fun(a,i-1,k-a[i-1])

l=[1,2,3,8]

k=7

print(fun(l,len(l),k))

**Q6. Print the subsets having the sum**

def sub(a,k,i=0,res=[]):

if i==len(a):

if sum(res)==k:

print(res)

return

sub(a,k,i+1,res+[a[i]])

sub(a,k,i+1,res)

l=[1,2,3,4,8]

k=8

sub(l,k)

**Q7. Frequency of element in a list**

def fun(a,k,i=0):

if i==len(a):

return 0

if a[i]==k:

return 1+fun(a,k,i+1)

else:

return fun(a,k,i+1)

l=[1,2,3,3,4]

k=3

print(fun(l,k))

**Q8. Dislike of Threes**

l=int(input())

for i in range(l):

a=int(input())

c=0

n=1

while True:

if n%3==0 or n%10==3:

n+=1

continue

c+=1

if c==a:

print(n)

break

n+=1

**DAY:5**

**Backtracking**

**Q1. Sum of elements in a matrix**

m=[[1,2,3],[4,5,6],[7,8,9]]

s=0

for i in range(len(m)):

for j in range(len(m[0])):

s+=m[i][j]

print(s)

**Q2. Count of prime numbers in a matrix**

m=[[1,2,3],[4,5,6],[7,8,9]]

c=0

for i in range(len(m)):

for j in range(len(m[0])):

n=m[i][j]

if n>1:

p=True

for k in range(2,n):

if n%k==0:

p=False

break

if p:

c+=1

print(c)

**Q3. Path to reach the exit**

def path(m,i,j,p,n):

if i==n-1 and j==n-1:

print(p)

return

if i+1<n and m[i+1][j]==1:

path(m,i+1,j,p+"D",n)

if j+1<n and m[i][j+1]==1:

path(m,i,j+1,p+"R",n)

m=[[1,1,1,0],[0,1,1,0],[0,1,1,1],[0,0,0,1]]

path(m,0,0,"",len(m))

**Traversal in a matrix:**

[1, 1, 1]

[1, 1, 1]

[1, 1, 1]

UP=(i-1,j)

DOWN=(i+1,j)

RIGHT=(i,j+1)

LEFT=(i,j-1)

**Q4. Forest Fire (Count of trees that did not catch fire)**

def fire(m,i,j):

c=0

if not m:

return 0

if i<0 or i>=len(m) or j<0 or j>=len(m) or m[i][j]!=1:

return 0

m[i][j]=2

fire(m,i+1,j)

fire(m,i-1,j)

fire(m,i,j+1)

fire(m,i,j-1)

for i in range(len(m)):

for j in range(len(m[0])):

if m[i][j]==1:

c+=1

return c

m=[[1,1,1,0,1],[0,1,1,1,0],[0,0,0,1,0],[1,1,0,0,1]]

print(fire(m,0,0))

**Q5. Number of Islands**

def island(m,i,j):

if not m:

return 0

if i<0 or i>=len(m) or j<0 or j>=len(m) or m[i][j]!=1:

return 0

m[i][j]=2

island(m,i+1,j)

island(m,i-1,j)

island(m,i,j+1)

island(m,i,j-1)

m=[[1,1,1,1,0],[1,1,0,1,0],[1,1,0,0,0],[0,0,0,0,0]]

c=0

for i in range(len(m)):

for j in range(len(m[0])):

if m[i][j]==1:

island(m,i,j)

c+=1

print(c)

**Q6. Possible binary numbers**

def bi(n,s=""):

if n==0:

print(s)

return

bi(n-1,s+"0")

bi(n-1,s+"1")

n=int(input())

bi(n)

**Q7. Count of valid paranthesis**

def par(n,s=0,e=0,p=""):

if s==n and e==n:

print(p)

return

if s<n:

par(n,s+1,e,p+"(")

if e<s:

par(n,s,e+1,p+")")

n=int(input())

par(n)

**Q8. Best time to sell and buys stocks**

def maxProfit(prices):

mp=prices[0]

m=0

for i in prices:

if i<mp:

mp=i

elif i-mp>m:

m=i-mp

return m

def maxProfit(prices):

mp=prices[0]

m=0

for i in prices:

m=max(m,i-mp)

mp=min(mp,i)

return m

p=[7,1,5,3,6,4]

print(maxProfit(p))

**Q9. Factorial Trailing Zeroes (Time Complexity: O(logN)**

def fun(n):

if n<5:

return 0

c=0

while n>=5:

n//=5

c+=n

return c

n=int(input())

print(fun(n))

**DAY:6**

**Sorting**

**Q1. Bubble Sort**

arr=[5,1,4,9,2,6,8]

for i in range(len(arr)):

flag=False

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

if arr[j]<arr[i]:

arr[i],arr[j]=arr[j],arr[i]

flag=True

if not flag:

break

print(arr)

**Q2. Insertion Sort**

arr=[3,6,1,5,9,2]

min\_ind=arr[0]

for i in range(0,len(arr)):

min\_ind=i

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

if arr[j]<arr[min\_ind]:

min\_ind=j

arr[i],arr[min\_ind]=arr[min\_ind],arr[i]

print(arr)

**Q3. Selection sort**

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

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

print(arr)

**Q4.Input:[3,6,1,7,4,2,5] Output:[6,4,2,1,3,5,7]**

a = [3, 6, 1, 7, 4, 2, 5]

a.sort()

b=[]

for i in a:

if i%2!=0:

b.append(i)

else:

b.insert(0,i)

print(b)

**Q5. Largest Number in a list**

a=[7,2,3,6,1] **#Bruteforce approach**

a.sort()

print(a[-1])

a=[7,2,3,6,1] **#Optimized code**

m=a[0]

for i in a:

if i>m:

m=i

print(m)

**Q6. Second Largest element in a list**

a=[7,2,3,6,1]

l=sl=0

for i in a:

if i>l:

sl,l=l,i

elif i>sl and i!=l:

sl=i

print(sl)

**Q7. K-th Largest Number in a list**

l=[7,2,3,6,1]

k=3

for i in range(k):

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

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

l[j],l[j+1]=l[j+1],l[j]

print(l[-k])

**Q8. Input:[2,3,5,1,6,9,8] Output:[2,3,1,5,6,9,8]**

a=[2,3,5,6,7,9,8]

k=2

for i in (k,len(a)):

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

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

a[j],a[j+1]=a[j+1],a[j]

print(a)

**Q9. Input:[[1,2],[5,1],[2,4],[6,3]] Output:[[5,1],[1,2],[6,3],[2,4]]**

a=[[1,2],[5,1],[2,4],[6,3]]

for i in range(len(a)):

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

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

a[j],a[j+1]=a[j+1],a[j]

print(a)

**Q10. Input:[[20,12,11],[10,5,22],[16,7,30]]**

def pri(x):

for i in x:

for j in range(2,int(i\*\*0.5)+1):

if i%j==0:

break

return i

return None

a=[[20,12,11],[10,5,22],[16,7,30]]

b=[]

n=len(a)

for i in a:

b.append(pri(i))

for i in range(n):

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

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

b[j],b[j+1]=b[j+1],b[j]

a[j],a[j+1]=a[j+1],a[j]

print(a)

**Q11. Sorting strings based on length**

a=["abc","hsahjhd","ha","hjajh"]

res=sorted(a,key=len)

print(res)

**Q12. Sorting based on frequency**

a="baabaaccc"

b={}

for i in a:

if i in b:

b[i]+=1

else:

b[i]=1

sc=sorted(b.items(),key=lambda item:item[1])

res=""

for c,count in sc:

res+=c\*count

print(res)

**Q13. Input:[[1,0,0,1],[1,0,0,0],[0,0,1,0],[0,1,1,0]] Output:[9,5,3,13]**

a=[[1,0,0,1],[1,0,0,0],[0,0,1,0],[0,1,1,0]]

b=[5,10,3,4]

for i in a:

s=0

for j in range(len(i)):

if i[j]==1:

s+=b[j]

print(s)

**DAY:7**

**Q1. Merge the lists and sort it- Time Complexity:O(n+m)**

a=[3,5,6,8]

b=[2,4,7]

res=[]

i=0

j=0

while i<len(a) and j<len(b):

if a[i]<b[j]:

res.append(a[i])

i+=1

else:

res.append(b[j])

j+=1

while i<len(a):

res.append(a[i])

i+=1

while j<len(b):

res.append(b[j])

j+=1

print(res)

**Q2. Merge Sort-Time Complexity: O(nlogn)+n [n can be ignored]**

def div(a): **#Time Complexity: O(nlogn)**

if len(a)<=1:

return a

mid=len(a)//2

left=div(a[:mid])

right=div(a[mid:])

return merge(left,right)

def merge(l,r): **# Time Complexity: O(n+m)**

res=[]

i=0

j=0

while i<len(l) and j<len(r):

if l[i]<r[j]:

res.append(l[i])

i+=1

else:

res.append(r[j])

j+=1

while i<len(l):

res.append(l[i])

i+=1

while j<len(r):

res.append(r[j])

j+=1

return res

l=[3,2,1,5,4,6]

print(div(l))

**Q3. K-th largest number using Bucket Sort approach**

l=[2,7,3,1,5,4]

k=3

res=[0]\*(max(l)+1)

for i in l:

res[i]=1

for i in range(len(res)-1,-1,-1):

if res[i]==1:

k-=1

if k==0:

print(i)

break

**Q4. K-th largest element having dupicate elements**

l=[5,3,6,2,2,1,1,8,7,3,1,12]

k=4

res=[0]\*(max(l)+1)

for i in l:

res[i]=1

c=0

for i in range(len(res)):

if res[i]>0:

c+=1

if c==k:

print(i)

**Q5. K-th minimum element in a list having duplicate elements using bucket sort**

a=[5,3,3,5,2,2,2,1,1,4,4,3,1]

dict={}

for i in a:

if i not in dict:

dict[i]=1

else:

dict[i]+=1

m=max(dict.values())

b=[[] for i in range(m+1)]

for i in dict:

b[dict[i]].append(i)

res=[]

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

for j in b[i]:

res.extend([j]\*i)

print(res)

**Q6. Majority Element: The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.**

a=[3,2,2,2,1]

c=-1

v=0

for i in a:

if v==0:

c=i

v=1

elif i==c:

v+=1

else:

v-=1

print(c)

**Q7. Linear Search: Time Complexity-O(n)**

a=[1,4,6,3,5,9]

k=6

for i in range(len(a)):

if a[i]==k:

print(i)

**Q8. Binary Search**

def bi(l,s,e,k):

if s>e:

return -1

mid=(s+e)//2

if l[mid]==k:

return mid

elif k<l[mid]:

return bi(l,s,mid-1,k)

else:

return bi(l,mid+1,e,k)

a=[1, 3, 4, 5, 6, 9, 11]

k=9

s=0

e=len(a)-1

print(bi(a,s,e,k))

**Q9. Search Insert Position: Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order. You must write an algorithm with O(log n) runtime complexity.**

def bi(a,k):

l=0

r=len(a)-1

while l<=r:

mid = l+(r-l)//2

if a[mid]==k:

return mid

elif a[mid]<k:

l=mid+1

else:

r=mid-1

return l

l=[1,3,5,6]

k=4

print(bi(l,k))

**DAY:8**

**Q1. Find First and Last Position of Element in a Sorted Array**

def searchRange(nums,target):

l=0

h=len(nums)-1

while l<=h:

mid=(l+h)//2

if nums[mid]<target:

l=mid+1

else:

h=mid-1

left=l

l=0

h=len(nums)-1

while l<=h:

mid=(l+h)//2

if nums[mid]<=target:

l=mid+1

else:

h=mid-1

right=h

if left<=right and 0<=left<len(nums) and nums[left]==target:

return [left,right]

else:

return [-1, -1]

nums = [5,7,7,8,8,10]

target = 8

print(searchRange(nums,target))

**Q2. Left and Right Rotation in a list**

a=[1,2,3,4,5,6,7]

k=3

res1=a[k:]+a[:k] #Left Rotation

print(res1)

res2=a[-k:]+a[:-k] #Right Rotation

print(res2)

**Q3. Find the index where the rotation starts in a list**

a=[4,5,6,1,2,3]

l=0

h=len(a)-1

while l<=h:

mid=(l+h)//2

if mid<len(a)-1 and a[mid]>a[mid+1]:

res=mid

break

if a[mid]<a[mid-1]:

res=mid-1

break

if a[l]>=a[mid]:

h=mid-1

else:

l=mid+1

print(res)

**Q4. Peak Element in a list**

a=[1,2,3,4,5,3,2]

l=0

r=len(a)-1

while l<r:

mid=(l+r)//2

if a[mid]>a[mid+1]:

r=mid

else:

l=mid+1

print(l)

def findPeakElement(a):

n=len(a)

if len(a)==1:

return 0

if a[0]>a[1]:

return 0

if a[n-1]>a[n-2]:

return n-1

l=1

r=n-2

while l<=r:

mid=(l+r)//2

if a[mid-1]<a[mid] and a[mid]>a[mid+1]:

return mid

if a[mid]>a[mid-1]:

l=mid+1

else:

r=mid-1

return -1

a=[1,2,3,4,5,3,2]

print(findPeakElement(a))

**Q5. Sqrt(x) :**

def mySqrt(x):

l=0

r=x

while l<=r:

mid=(l+r)//2

if mid\*mid==x:

return mid

elif mid\*mid>x:

r=mid-1

else:

l=mid+1

return r

x=4

print(mySqrt(x))

**Q6. Search in Rotated Sorted Array**

def search(nums,target):

l=0

r=len(nums)-1

while l<=r:

mid=(r+l)//2

if nums[mid]==target:

return mid

if nums[l]<=nums[mid]:

if nums[l]<=target<=nums[mid]:

r=mid-1

else:

l=mid+1

else:

if nums[r]>=target>=nums[mid]:

l=mid+1

else:

r=mid-1

return -1

nums=[4,5,6,7,0,1,2]

target=3

print(search(nums,target))

**Q7. Koko Eating Bananas**

from math import ceil

def minEat(piles,h):

l,r=1,max(piles)

def caneat(k):

res=0

for p in piles:

res+=ceil(p/k)

if res<=h:

return True

else:

return False

while l<=r:

mid=(l+r)//2

if caneat(mid):

r=mid-1

else:

l=mid+1

return l

piles=[3,6,7,11]

h=8

print(minEat(piles,h))

**DAY:9**

**Q1. Capacity to ship Packages within D Days**

def shipWithinDays(weights,days):

def canship(k):

cc=0

n=1

for i in weights:

if cc+i>k:

n+=1

cc=0

cc+=i

return n<=days

l=max(weights)

r=sum(weights)

while l<=r:

mid=(l+r)//2

if canship(mid):

r=mid-1

else:

l=mid+1

return l

weights=[1,2,3,4,5,6,7,8,9,10]

days=5

print(shipWithinDays(weights,days))

**Q2. Search a 2D Matrix**

def searchMatrix(matrix,target):

m=len(matrix)

n=len(matrix[0])

l=0

r=m\*n-1

while l<=r:

mid=(l+r)//2

i=mid//n

j=mid%n

mid\_val=matrix[i][j]

if mid\_val==target:

return True

elif mid\_val<target:

l=mid+1

else:

r=mid-1

return False

matrix=[[1,3,5,7],[10,11,16,20],[23,30,34,60]]

target=3

print(searchMatrix(matrix,target))

**Q3. Agressive Cows: You are given an array with unique elements of stalls[], which denote the position of a stall. You are also given an integer k which denotes the number of aggressive cows. Your task is to assign stalls to k cows such that the minimum distance between any two of them is the maximum possible.**

def cankeep(stalls,k,d):

c=1

lp=stalls[0]

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

if stalls[i]-lp>=d:

c+=1

lp=stalls[i]

if c==k:

return True

return False

def aggressiveCows(stalls,k):

l=1

r=stalls[-1]-stalls[0]

bd=0

while(l<=r):

mid=(l+r)//2

if cankeep(stalls,k,mid):

bd=mid

l=mid+1

else:

r=mid-1

return bd

stalls=[1, 2, 4, 8, 9]

k=3

print(aggressiveCows(stalls,k))

**Q4. Nim Game: You are playing the following Nim Game with your friend:Initially, there is a heap of stones on the table.You and your friend will alternate taking turns, and you go first. On each turn, the person whose turn it is will remove 1 to 3 stones from the heap. The one who removes the last stone is the winner. Given n, the number of stones in the heap, return true if you can win the game assuming both you and your friend play optimally, otherwise return false.**

def canWinNim(n):

return n%4!=0

n=4

print(canWinNim(n))

**Q5. Candies and Two Sisters: There are two sisters Alice and Betty. You have n candies. You want to distribute these n candies between two sisters in such a way that: Alice will get a (a>0 ) candies; Betty will get b b>0 ) candies; each sister will get some integer number of candies Alice will get a greater amount of candies than Betty (i.e. a>b ); all the candies will be given to one of two sisters (i.e. a+b=n ). Your task is to calculate the number of ways to distribute exactly n candies between sisters in a way described above. Candies are indistinguishable. Formally, find the number of ways to represent n as the sum of n=a+b where a and b are positive integers and a>b You have to answer t independent test cases.**

t=int(input())

for i in range(t):

n=int(input())

if n<3:

print("0")

else:

print((n-1)//2)

**Q6. Remove Smallset: You are given the array a consisting of n positive (greater than zero) integers. In one move, you can choose two indices i and j (i≠j ) such that the absolute difference between ai and aj is no more than one (|ai−aj|≤1 ) and remove the smallest of these two elements. If two elements are equal, you can remove any of them (but exactly one). Your task is to find if it is possible to obtain the array consisting of only one element using several (possibly, zero) such moves or not. You have to answer t independent test cases.**

t=int(input())

for i in range(t):

n=int(input())

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

arr.sort()

for j in range(n-1):

if abs(arr[j]-arr[j+1])>1:

print("NO")

break

else:

     print("YES")

**DAY:10**

**Q1. Lemonade Change**

def lemonadeChange(bills):

f=0

t=0

tw=0

for i in bills:

if i==5:

f+=1

elif i==10:

if f==0:

return False

f-=1

t+=1

elif i==20:

if t>0 and f>0:

t-=1

f-=1

tw+=1

elif f>=3:

f-=3

tw+=1

else:

return False

else:

if tw>0 and f>0:

tw-=1

f-=1

elif f>=5:

f-=5

else:

return False

return True

bills=[5,5,5,10,20,30]

print(lemonadeChange(bills))

**Q2. Jump Game: You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.**

def canJump(nums):

p=0

for i in nums:

if p<0:

return False

elif i>p:

p=i

p-=1

return True

nums=[3,2,1,0,4]

print(canJump(nums))

**Q3. Shortest Job First**

def solve(bt):

bt.sort()

n=len(bt)

wt=[0]\*n

for i in range(1,n):

wt[i]=wt[i-1]+bt[i-1]

return int(sum(wt)/n)

bt=[4,3,7,1,2]

print(solve(bt))

**Q4. Assign Cookies: Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie. Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.**

def findContentChildren(g,s):

g.sort()

s.sort()

i,j=0,0

c=0

while i<len(g) and j<len(s):

if s[j]>=g[i]:

c+=1

i+=1

j+=1

return c

g=[1,2]

s=[1,2,3]

print(findContentChildren(g,s))

**Q5. N Meeting in one room: You are given timings of n meetings in the form of (start[i], end[i]) where start[i] is the start time of meeting i and end[i] is the finish time of meeting i. Return the maximum number of meetings that can be accommodated in a single meeting room, when only one meeting can be held in the meeting room at a particular time.**

def maximumMeetings(start,end):

m=list(zip(start,end))

m.sort(key=lambda x:x[1])

c=0

l=-1

for s,e in m:

if s>l:

c+=1

l=e

return c

start=[1, 3, 0, 5, 8, 5]

end=[2, 4, 6, 7, 9, 9]

print(maximumMeetings(start,end))

**Q6. Your task is to modify and print the given string in such a way that the adjacent characters in the string don't repeate themselves more than twice.**

s="aabbbc"

res=[]

n=len(s)

for i in range(n):

if i>=2 and s[i]==s[i-1]==s[i-2]:

continue

res.append(s[i])

print("".join(res))

#Your task is it reduce the string to the minimum length where no two adjacent letters can be same.

n="abba"

st=[]

for i in n:

if st and st[-1]==i:

st.pop()

else:

st.append(i)

print("".join(st))

#Length of the longest palindromic subsequence

**Q7. String Compression: Input- "aaaabbbc" Output- "a4b3c1"**

s="aaaabbbc"

res=[]

c=1

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

if s[i]==s[i-1]:

c+=1

else:

res.append(s[i-1]+str(c))

c=1

print("".join(res))

**Q8. Reverse Vowels of a String**

def reverseVowels(s):

v=set("aeiouAEIOU")

s=list(s)

l=0

r=len(s)-1

while l<r:

if s[l] not in v:

l+= 1

elif s[r] not in v:

r-= 1

else:

s[l],s[r]=s[r],s[l]

l+= 1

r-= 1

return "".join(s)

s="IceCreAm"

print(reverseVowels(s))

**DAY:11**

**Q1. Valid Palindrome II**

def validPalindrome(s):

def isPal(s,l,r):

while l<r:

if s[l]!=s[r]:

return False

l+=1

r-=1

return True

l=0

r=len(s)-1

while(l<r):

if s[l]!=s[r]:

return isPal(s,l+1,r) or isPal(s,l,r-1)

l+=1

r-=1

return True

s="abca"

print(validPalindrome(s))

**Q2. Check if the list is sorted and doesn't have any duplicate elements**

l=[1,2,3,4,5,7]

f=0

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

if l[i]>=l[i+1]:

f+=1

break

if f==1:

print("No")

else:

print("Yes")

**Q3. Fixed Sliding Window**

l=[4,3,4,5,1,3,2,1,5,2,3,5]

k=4

sum=0

for i in range(k):

sum+=l[i]

m=sum

for i in range(len(l)-k):

sum=sum+l[i+k]-l[i]

m=max(m,sum)

print(m)

**Subarray:** Without changing any order whereas **Subsequence** we can skip an element

**Example:** [1,2,3]

Sub array: [1],[1,2]

Subsequence: [1,2,3],[1,2],[1],[2,3],[2],[3],[]

**Q4. Dynamic Sliding Window**

a=[2,5,1,7,10]

k=14

l,r,sum,m=0,0,0,0

while r<len(a):

sum+=a[r]

if sum>=k:

sum-=a[l]

l+=1

if sum<k:

m=max(m,(r-l)+1)

r+=1

print(m)

**Q5. Maximum Points you can obtain from Cards: There are several cards arranged in a row, and each card has an associated number of points. The points are given in the integer array cardPoints. In one step, you can take one card from the beginning or from the end of the row. You have to take exactly k cards. Your score is the sum of the points of the cards you have taken. Given the integer array cardPoints and the integer k, return the maximum score you can obtain.**

def maxScore(cardPoints):

n=len(cardPoints)

l\_s=sum(cardPoints[:k])

r\_s=0

m=l\_s

for i in range(k):

l\_s-=cardPoints[k-i-1]

r\_s+=cardPoints[n-i-1]

m=max(m,l\_s+r\_s)

return m

cardPoints=[1,2,3,4,5,6,1]

k=3

print(maxScore(cardPoints))

**Q6. Jump Game II: You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0]. Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where: 0 <= j <= nums[i] and i + j < n Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].**

def jump(nums):

l,r,j=0,0,0

while r<len(nums)-1:

m=0

for i in range(l,r+1):

if i+nums[i]>m:

m=i+nums[i]

l=r+1

r=m

j=j+1

return j

nums=[2,3,1,1,4]

print(jump(nums))

**DAY:12**

**Q1. Minimum number of doctors required at any time**

arr=[900,945,955,1100,1500,1800]

dep=[920,1200,1130,1150,1900,2000]

arr.sort()

dep.sort()

i,j,doct,m=1,0,1,0

while i<len(arr) and j<len(dep):

if arr[i]<=dep[j]:

doct+=1

i+=1

else:

doct-=1

j+=1

m=max(m,doct)

print(m)

**Q2. Find the length of the longest palindromic substring**

s="ababab"

m=0

for i in range(len(s)):

# Odd length palindrome

l=i

r=i

while l>=0 and r<len(s) and s[l]==s[r]:

m=max(m,(r-l)+1)

l-=1

r+=1

# Even length palindrome

l=i

r=i+1

while l>=0 and r<len(s) and s[l]==s[r]:

m=max(m,(r-l)+1)

l-=1

r+=1

print(m)

**Q3. Longest Substring without repeating characters**

def lengthofLongestSubstring(s):

l,m,d=0,0,{}

for r in range(len(s)):

v=s[r]

if v in d and d[v]>=l:

l=d[v]+1

d[v]=r

m=max(m,r-l+1)

    return m

s="abcabcbb"

print(lengthofLongestSubstring(s))

**Q4. Fruits into Basket: You are visiting a farm that has a single row of fruit trees arranged from left to right. The trees are represented by an integer array fruits where fruits[i] is the type of fruit the ith tree produces. You want to collect as much fruit as possible. However, the owner has some strict rules that you must follow: You only have two baskets, and each basket can only hold a single type of fruit. There is no limit on the amount of fruit each basket can hold. Starting from any tree of your choice, you must pick exactly one fruit from every tree (including the start tree) while moving to the right. The picked fruits must fit in one of your baskets. Once you reach a tree with fruit that cannot fit in your baskets, you must stop. Given the integer array fruits, return the maximum number of fruits you can pick.**

def totalFruit(fruits):

n=len(fruits)

l,m=0,0

dict={}

k=2

for i in range(n):

r=fruits[i]

if r in dict:

dict[r]+=1

else:

dict[r]=1

while len(dict)>k:

v=fruits[l]

dict[v]-=1

if dict[v]==0:

dict.pop(v)

l+=1

m=max(m,(i-l)+1)

return m

fruits=[1,2,1]

print(totalFruit(fruits))

**DAY:13**

**Q1. Basic Linked List Operations: (append, display, sum, length, evensum, oddsum, sum of even indices, sum of odd indices, largest number, second largest number, mid point of a linked list)**

class Node:

def \_\_init\_\_(self,val):

self.data=val

self.next=None

class Linked\_list:

def \_\_init\_\_(self):

self.head=None

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def display(self):

temp=self.head

while temp!=None:

print(temp.data,end="->")

temp=temp.next

print("None")

def sum1(self):

temp=self.head

t=0

while temp!=None:

t+=temp.data

temp=temp.next

return t

def count(self):

temp=self.head

c=0

while temp!=None:

c+=1

temp=temp.next

return c

def evensum(self):

temp=self.head

t=0

while temp!=None:

if temp.data%2==0:

t+=temp.data

temp=temp.next

return t

def oddsum(self):

temp=self.head

t=0

while temp!=None:

if temp.data%2!=0:

t+=temp.data

temp=temp.next

return t

def even\_insum(self):

temp=self.head

i=0

t=0

while temp:

if i%2==0:

t+=temp.data

temp=temp.next

i+=1

return t

def odd\_insum(self):

temp=self.head

i=0

t=0

while temp:

if i%2!=0:

t+=temp.data

temp=temp.next

i+=1

return t

def largest(self):

temp=self.head

m=temp.data

while temp is not None:

if temp.data>m:

m=temp.data

temp=temp.next

return m

def second\_largest(self):

temp=self.head

l=temp.data

temp=temp.next

sl=-1

while temp:

if temp.data>l:

sl=l

l=temp.data

elif temp.data!=l and temp.data>sl:

sl=temp.data

temp=temp.next

return sl

def mid(self): #mid point- Time Complexity: O(n+m)

temp=self.head

temp2=self.head

c=0

while temp!=None:

c+=1

temp=temp.next

m=c//2

for i in range(m):

temp2=temp2.next

return temp2.data

l1=Linked\_list()

l1.head=Node(10)

l1.append(20)

l1.append(30)

l1.append(5)

l1.display()

print(l1.sum1())

print(l1.count())

print(l1.evensum())

print(l1.oddsum())

print(l1.even\_insum())

print(l1.odd\_insum())

print(l1.largest())

print(l1.second\_largest())

print(l1.mid())

**Q2. Middle of Linked List**

class ListNode:

def \_\_init\_\_(self,val=0,next=None):

self.val=val

self.next=next

class Solution:

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def middleNode(self,head):

fast=head

slow=head

while (fast!=None and fast.next!=None):

fast=fast.next.next

slow=slow.next

return slow

h=ListNode()

h.next=ListNode(10)

h.next.next=ListNode(20)

h.next.next.next=ListNode(30)

h.next.next.next.next=ListNode(5)

s=Solution()

m=s.middleNode(h)

print(m.val)

**Q3. Cycle in a Linked List**

class Node:

def \_\_init\_\_(self,val):

self.data=val

self.next=None

class Linked\_list:

def \_\_init\_\_(self):

self.head=None

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def cycle(self):

slow=self.head

fast=self.head

while fast and fast.next:

slow=slow.next

fast=fast.next.next

if slow==fast:

return True

return False

l2=Linked\_list()

l2.head=Node(3)

l2.append(2)

l2.append(0)

l2.append(-4)

l2.head.next.next.next.next=l2.head.next

print(l2.cycle())

**Q4. Linked List Cycle II: Given the head of a linked list, return the node where the cycle begins. If there is no cycle, return null. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to (0-indexed). It is -1 if there is no cycle. Note that pos is not passed as a parameter.**

class Node:

def \_\_init\_\_(self,val):

self.data=val

self.next=None

class Linked\_list:

def \_\_init\_\_(self):

self.head=None

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def cycle(self):

slow=self.head

fast=self.head

while fast and fast.next:

slow=slow.next

fast=fast.next.next

if slow==fast:

break

else:

return False

slow=self.head

while slow!=fast:

slow=slow.next

fast=fast.next

return slow.data

l2=Linked\_list()

l2.head=Node(3)

l2.append(2)

l2.append(0)

l2.append(-4)

l2.head.next.next.next.next=l2.head.next

print(l2.cycle())

**Q5. Length of the cycle in a linked list**

class Node:

def \_\_init\_\_(self,val):

self.data=val

self.next=None

class Linked\_list:

def \_\_init\_\_(self):

self.head=None

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def cyclelen(self):

slow=self.head

fast=self.head

while fast and fast.next:

slow=slow.next

fast=fast.next.next

if slow==fast:

break

else:

return False

slow=self.head

while slow!=fast:

slow=slow.next

fast=fast.next

s=slow

l=1

temp=s.next

while temp!=s:

temp=temp.next

l+=1

return l

l2=Linked\_list()

l2.head=Node(3)

l2.append(2)

l2.append(0)

l2.append(-4)

l2.head.next.next.next.next=l2.head.next

print(l2.cyclelen())

**Q6. Print the K-th element from the last**

class Node:

def \_\_init\_\_(self,val):

self.data=val

self.next=None

class Linked\_list:

def \_\_init\_\_(self):

self.head=None

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def klast(self,k):

fast=self.head

slow=self.head

for i in range(k):

fast=fast.next

while fast!=None:

fast=fast.next

slow=slow.next

return slow.data

l2=Linked\_list()

l2.head=Node(3)

l2.append(2)

l2.append(0)

l2.append(-4)

k=3

print(l2.klast(k))

**Q7. Remove N-th Node from the end of the linked list**

class Node:

def \_\_init\_\_(self,val):

self.data=val

self.next=None

class Linked\_list:

def \_\_init\_\_(self):

self.head=None

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def klast(self,k):

temp=self.head

fast=temp

slow=temp

for i in range(k+1):

if fast:

fast=fast.next

while fast!=None:

fast=fast.next

slow=slow.next

slow.next=slow.next.next

return temp.next

def display(self):

temp=self.head

while temp!=None:

print(temp.data,end="->")

temp=temp.next

print("None")

l2=Linked\_list()

l2.head=Node(3)

l2.append(2)

l2.append(0)

l2.append(-4)

k=2

l2.klast(k)

l2.display()

**Q8. Find if k is the sum of two consecutive nodes in the linked list**

class Node:

def \_\_init\_\_(self,val):

self.data=val

self.next=None

class Linked\_list:

def \_\_init\_\_(self):

self.head=None

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def sum1(self,k):

temp=self.head

s=0

flag=0

while temp.next!=None:

if temp.data+temp.next.data==k:

flag=1

temp=temp.next

if flag==1:

return True

return False

l2=Linked\_list()

l2.head=Node(3)

l2.append(2)

l2.append(0)

l2.append(4)

k=5

print(l2.sum1(k))

**Q9. Find if k is the sum of any two nodes in the linked list**

class Node:

def \_\_init\_\_(self,val):

self.data=val

self.next=None

class Linked\_list:

def \_\_init\_\_(self):

self.head=None

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def sum1(self,k):

s=set()

temp=self.head

while temp!=None:

if k-temp.data in s:

return True

s.add(temp.data)

temp=temp.next

return False

l2=Linked\_list()

l2.head=Node(3)

l2.append(2)

l2.append(0)

l2.append(4)

k=6

print(l2.sum1(k))

**Q10. Swap nodes in pairs**

class Node:

def \_\_init\_\_(self,val):

self.data=val

self.next=None

class Linked\_list:

def \_\_init\_\_(self):

self.head=None

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def swap(self):

temp=self.head

while temp and temp.next:

temp.data,temp.next.data=temp.next.data,temp.data

temp=temp.next.next

def display(self):

temp=self.head

while temp!=None:

print(temp.data,end="->")

temp=temp.next

print("None")

l2=Linked\_list()

l2.head=Node(3)

l2.append(2)

l2.append(0)

l2.append(4)

l2.swap()

l2.display()

**DAY:14**

**Q1. Intersection of two linked lists**

class Solution:

def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> Optional[ListNode]:

i=headA

j=headB

while i!=j:

if i:

i=i.next

else:

i=headB

if j:

j=j.next

else:

j=headA

return i

**Q2. Reverse a linked list**

class Solution:

def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:

pre=None

c=head

while c!=None:

nn=c.next

c.next=pre

pre=c

c=nn

return pre

**Q3. Palindrome in linked list**

class Solution:

def isPalindrome(self, head: Optional[ListNode]) -> bool:

slow,fast=head,head

while (fast!=None and fast.next!=None):

slow=slow.next

fast=fast.next.next

pre=None

c=slow

while c!=None:

temp=c.next

c.next=pre

pre=c

c=temp

l,r=head,pre

while r!=None:

if l.val!=r.val:

return False

l=l.next

r=r.next

return True

**Q4. Bubble Sort in linked list**

class Node:

def \_\_init\_\_(self,val):

self.data=val

self.next=None

class Linked\_list:

def \_\_init\_\_(self):

self.head=None

def append(self,val):

temp=self.head

while temp.next!=None:

temp=temp.next

temp.next=Node(val)

def bubble(self):

e=None

while self.head.next!=e:

c=self.head

while c.next!=e:

if c.data>c.next.data:

c.data,c.next.data=c.next.data,c.data

c=c.next

e=c

def display(self):

temp=self.head

while temp!=None:

print(temp.data,end="->")

temp=temp.next

print("None")

l1=Linked\_list()

l1.head=Node(10)

l1.append(20)

l1.append(30)

l1.append(5)

l1.display()

l1.bubble()

l1.display()

**Q5. Add two numbers using a linked list**

class Solution:

def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:

temp2=ListNode()

temp=temp2

c=0

while l1 or l2 or c:

s=c

if l1:

s+=l1.val

l1=l1.next

if l2:

s+=l2.val

l2=l2.next

c=s//10

temp.next=ListNode(s%10)

temp=temp.next

return temp2.next

**Q6. Valid Parentheses**

def isValid(s):

st=[]

for i in s:

if i in "({[":

st.append(i)

else:

if not st:

return False

l=st.pop()

if l=='(':

if i!=')':

return False

elif l=='{':

if i!='}':

return False

elif l=='[':

if i!=']':

return False

return not st

s="()[]{}"

print(isValid(s))

**Q7. Removing stars from a string**

def removeStars(s):

st=[]

for i in s:

if i=="\*":

st.pop()

else:

st.append(i)

return "".join(st)

s="leet\*\*cod\*e"

print(removeStars(s))

**Q8. Number of students unable to eat lunch**

def countStudents(students,sandwiches):

c=len(students)

while students and sandwiches and sandwiches[0] in students:

if students[0]!=sandwiches[0]:

students.append(students[0])

students.pop(0)

else:

students.pop(0)

sandwiches.pop(0)

c-=1

return c

students=[1,1,1,0,0,1]

sandwiches=[1,0,0,0,1,1]

print(countStudents(students,sandwiches))

**DAY:15**

**Q1. Next Greater Element: The next greater element of some element x in an array is the first greater element that is to the right of x in the same array. You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2. For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the next greater element of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1. Return an array ans of length nums1.length such that ans[i] is the next greater element as described above.**

def nextGreaterElement(nums1,nums2):

g={n:i for i,n in enumerate(nums1)}

st=[]

res=[-1]\*len(nums1)

for i in range(len(nums2)):

cur=nums2[i]

while st and cur>st[-1]:

val=st.pop()

index=g[val]

res[index]=cur

if cur in g:

st.append(cur)

return res

nums1=[4,1,2]

nums2=[1,3,4,2]

print(nextGreaterElement(nums1,nums2))

**Q2. Online Stock Span**

l=[1,2,3,4]

st=[]

for i in l:

st.append(i)

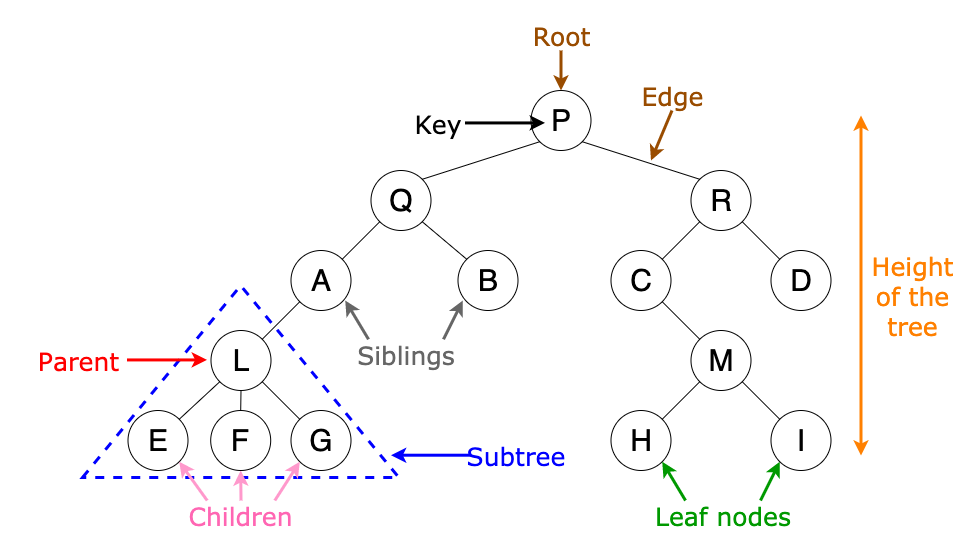
st.pop()

print(st[-1])

print(st)

**Tree:** Non-linear datastructure.

**Key Terms:**



**Tyes of Trees:**

* Binary Tree
* Binary search Tree
* AVL Tree
* Red-Black Tree
* Trie
* Heap
* Complete Tree
* Perfect Binary Tree
* Skewed Tree

**Traversal Types:**

* **Inorder**: Left→Root→ Right
* **Postorder**: Left→ Right→ Root
* **Preorder**:Root→ Left→ Right

**Q3. Binary search Tree Traversal**

class Node:

def \_\_init\_\_(self,key):

self.left=None

self.right=None

self.val=key

def insert(root,key):

if root is None:

return Node(key)

else:

if key<root.val:

root.left=insert(root.left,key)

else:

root.right=insert(root.right,key)

return root

def inorder(root):

if root is None:

return

inorder(root.left)

print(root.val,end=" ")

inorder(root.right)

def preOrder(root):

if root is None:

return

print(root.val,end=" ")

preOrder(root.left)

preOrder(root.right)

def postOrder(root):

if root is None:

return

postOrder(root.left)

postOrder(root.right)

print(root.val,end=" ")

def levelOrder(root):

q=[]

q.append(root)

while len(q)!=0:

rt=q.pop(0)

print(rt.val,end=" ")

if rt.left is not None:

q.append(rt.left)

if rt.right is not None:

q.append(rt.right)

c=Node(7)

c=insert(c,key=3)

c=insert(c,key=5)

c=insert(c,key=2)

c=insert(c,key=6)

inorder(c)

preOrder(c)

postOrder(c)

levelOrder(c)

**Q4. Is it Binary Search Tree**

def check\_binary\_search\_tree\_(root):

res=[]

def inorder(root):

if root is None:

return

inorder(root.left)

res.append(root.data)

inorder(root.right)

inorder(root)

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

if res[i]>=res[i+1]:

return False

return True

**Q5. Height of a Binary Tree**

def height(root):

if root is None:

return -1

left=height(root.left)

right=height(root.right)

return max(left,right)+1

**Q6. Top View in Binary Tree**

def topView(root):

#Write your code here

q=[]

d=dict()

root.level=0

q.append(root)

while q:

root=q.pop(0)

if root.level not in d:

d[root.level]=root.info

if root.left is not None:

q.append(root.left)

root.left.level=root.level-1

if root.right is not None:

q.append(root.right)

root.right.level=root.level+1

for key in sorted(d.key()):

print(d[key],end=" ")

**Q7. Combination of two values upto n**

n=10

q=[]

q.append("5")

q.append("6")

c=0

while c<n:

t=q.pop(0)

print(t,end=" ")

q.append(t+"5")

q.append(t+"6")

c+=1

**DAY:16**

**Q1. BST: Lowest Common Ancestor**

class Node:

def \_\_init\_\_(self, info):

self.info = info

self.left = None

self.right = None

self.level = None

def lca(root, v1, v2):

if v1<root.info and v2<root.info:

return lca(root.left,v1,v2)

elif v1>root.info and v2>root.info:

return lca(root.right,v1,v2)

else:

return root

**Q2. Sum of all the Nodes in the BST**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def sum1(self):

t=self.data

if self.left:

t+=self.left.sum1()

if self.right:

t+=self.right.sum1()

return t

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(root.sum1())

**Q3. Sum of even Nodes in BST**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def even\_sum(self):

t=0

if self.data%2==0:

t=self.data

if self.left is not None:

t+=self.left.even\_sum()

if self.right is not None:

t+=self.right.even\_sum()

return t

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(root.even\_sum())

**Q4. Sum of even Nodes in BST**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def odd\_sum(self):

t=0

if self.data%2!=0:

t=self.data

if self.left is not None:

t+=self.left.odd\_sum()

if self.right is not None:

t+=self.right.odd\_sum()

return t

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(root.odd\_sum())

**Q5. Print sum of prime Nodes in BST**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def prime(self):

def is\_prime(n):

if n<2:

return False

for i in range(2,int(n\*\*0.5)+1):

if n%i==0:

return False

return True

t=0

if is\_prime(self.data):

t=self.data

if self.left is not None:

t+=self.left.prime()

if self.right is not None:

t+=self.right.prime()

return t

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(root.prime())

**Q6. Print all Leaf Nodes in BST**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def print\_leaf(self):

if self.left is None and self.right is None:

print(self.data,end=" ")

if self.left:

self.left.print\_leaf()

if self.right:

self.right.print\_leaf()

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(root.print\_leaf())

**Q7. K-th largets element in BST**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def klargest(self,k):

def inorder(node,l):

if node is None:

return

inorder(node.right,l)

l.append(node.data)

inorder(node.left,l)

l=[]

inorder(self,l)

return l[k-1]

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(root.klargest(3))

**Q8. K-th smallest element in BST**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def ksmallest(self,k):

def inorder(node,l):

if node is None:

return

inorder(node.left,l)

l.append(node.data)

inorder(node.right,l)

l=[]

inorder(self,l)

return l[k-1]

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(root.ksmallest(3))

**Q9. Smae tree: Given the roots of two binary trees p and q, write a function to check if they are the same or not. Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.**

class Solution:

def isSameTree(self, p: Optional[TreeNode], q: Optional[TreeNode]) -> bool:

if not p and not q:

return True

if not p or not q or p.val!=q.val:

return False

return self.isSameTree(p.left,q.left) and self.isSameTree(p.right,q.right)

**Q10. Invert Binary Tree**

class Solution:

def invertTree(self,root):

if not root:

return None

root.left,root.right=self.invertTree(root.right),self.invertTree(root.left)

return root

**Q11. Count of leaf Nodes in BST**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def count\_leaf(self):

if self.left is None and self.right is None:

return 1

c=0

if self.left:

c+=self.left.count\_leaf()

if self.right:

c+=self.right.count\_leaf()

return c

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(root.count\_leaf())

**Q12. Top View in Binary Tree**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def topView(root):

if not root:

return

q=[]

d=dict()

q.append((root,0))

while q:

node,e=q.pop(0)

if e not in d:

d[e]=node.data

if node.left:

q.append((node.left,e-1))

if node.right:

q.append((node.right,e+1))

for key in sorted(d):

print(d[key],end=" ")

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

topView(root)

**Q13. Bottom View of Binary Tree**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def bottomView(root):

if not root:

return

q=[]

d=dict()

q.append((root,0))

while q:

node,e=q.pop(0)

d[e]=node.data

if node.left:

q.append((node.left,e-1))

if node.right:

q.append((node.right,e+1))

for key in sorted(d):

print(d[key],end=" ")

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print()

bottomView(root)

**Q14. Left View of Binary Tree**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def leftView(root):

if not root:

return

q=[]

d=dict()

q.append((root,0))

while q:

node,e=q.pop(0)

if e not in d:

d[e]=node.data

if node.left:

q.append((node.left,e+1))

if node.right:

q.append((node.right,e+1))

for key in sorted(d):

print(d[key],end=" ")

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print()

leftView(root)

**Q15. Right View of Binary tree**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def rightView(root):

if not root:

return

q=[]

d=dict()

q.append((root,0))

while q:

node,e=q.pop(0)

d[e]=node.data

if node.left:

q.append((node.left,e+1))

if node.right:

q.append((node.right,e+1))

for key in sorted(d):

print(d[key],end=" ")

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print()

rightView(root)

**Q16. Huffman Decoding**

def decodeHuff(root, s):

#Enter Your Code Here

temp=root

res=[]

for i in s:

if i=="0":

temp=temp.left

else:

temp=temp.right

if temp.left is None and temp.right is None:

res.append(temp.data)

temp=root

print("".join(res))

**DAY17:**

**Q1. Search key element in BST**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def search(root,key):

if root is None:

return False

if root.data==key:

return True

elif root.data>key:

return search(root.left,key)

else:

return search(root.right,key)

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(search(root,key=1))

**Q2.Search key element in Binary Tree**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def search\_bi(root,key):

if root is None:

return False

if root.data==key:

return True

return search\_bi(root.left,key) or search\_bi(root.right,key)

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(search\_bi(root,key=1))

**Q3. Print all the paths from the root to the leaf nodes**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def path(root,p=[]):

if root is None:

return

p.append(str(root.data))

if root.left is None and root.right is None:

print(" ".join(p))

path(root.left,p)

path(root.right,p)

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

path(root)

**Q4. Max sum of the path**

class Node:

def \_\_init\_\_(self,data):

self.data=data

self.left=None

self.right=None

def path\_sum(root,p=[],s=0):

if root is None:

return

p.append(str(root.data))

s+=root.data

if root.left is None and root.right is None:

return s

ls=path\_sum(root.left,p,s)

rs=path\_sum(root.right,p,s)

return max(s,ls,rs)

root=Node(5)

root.left=Node(2)

root.right=Node(7)

root.left.left=Node(1)

root.left.right=Node(3)

root.right.left=Node(6)

root.right.right=Node(8)

print(path\_sum(root))

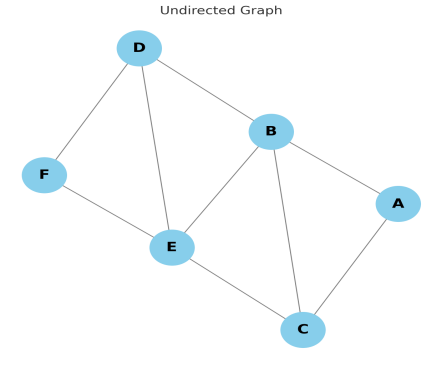
**Graphs:** A non-linear structure consisting of nodes and edges. Widely used ot represent networks.

**Types of Graphs:**

1. **Directed Graph:** Edges have a direction(one-way).
2. **Undirected Graph:** Edges have no direction (bidirectional).
3. **Weighted Graph:** Edges have associated weights.
4. **Unweighted Graph:** Edges do not have weights.
5. **Cyclic Graph:** Contains at least one cycle.
6. **Acyclic Graph:** No cycle exist.
7. **Connected Graph:** All nodes are reachable from any other nodes.
8. **Disconnected Graph:** Some nodes are not reachable from others.

**We can represent graph in two ways:**

* Adjecency List
* Adjecency Matrix



**Adjecency List:**

A:[B,C]

B:[A,C,D,E]

C:[A,B,E]

D:[B,E,F]

E:[C,B,D,F]

F:[D,E]

**Adjecency Matrix:**

**A B C D E F**

**A** 0 1 1 0 0 0

**B**  1 0 1 1 1 0

**C** 1 1 0 0 1 0

**D**  0 1 0 0 1 1

**E** 0 1 1 1 0 1

**F** 0 0 0 1 1 0

**Adjacency List →** Best for large graphs with few connections (saves space).

**Adjacency Matrix →** Best for dense graphs with many connections (fast lookups).

If your graph is huge but mostly empty, use a list. If your graph is full of connections, use a matrix.

**Q5. DFS Traversal of a Graph**

def dfs(graph,n,v=set()):

if n not in v:

print(n,end="")

v.add(n)

for i in graph[n]:

dfs(graph,i,v)

graph={'A':['B','C'],

'B':['A','D','E'],

'C':['A','F'],

'D':['B'],

'E':['B','F'],

'F':['C','E']}

dfs(graph,n='A')

**Q6. BFS Traversal of a Graph**

def bfs(graph,start):

v=set()

q=[start]

while q:

n=q.pop(0)

if n not in v:

print(n,end=" ")

v.add(n)

q.extend(graph[n])

graph={'A':['B','C'],

'B':['A','D','E'],

'C':['A','F'],

'D':['B'],

'E':['B','F'],

'F':['C','E']}

bfs(graph,start='A')

**Q7. Directed Graph if path exist or not**

from collections import defaultdict

edges=[(0,1),(0,2),(1,3),(2,4),(3,5),(4,5)]

graph=defaultdict(list)

for u,v in edges:

graph[u].append(v)

graph[v].append(u)

def path(graph,start,end):

v=set()

def dfs(val):

if val==end:

return True

v.add(val)

for n in graph[val]:

if n not in v:

if dfs(n):

return True

return False

return dfs(start)

start,end=0,5

print(path(graph,start,end))

**Q8. Directed graph: BFS Traversal**

from collections import defaultdict

edges=[(0,1),(0,2),(1,3),(2,4),(3,5),(4,5)]

graph=defaultdict(list)

for u,v in edges:

graph[u].append(v)

graph[v].append(u)

def path(graph,start,end):

v=set()

q=[]

q.append(start)

v.add(start)

while q:

node=q.pop(0)

if node==end:

return True

for i in graph[node]:

if i not in v:

v.add(i)

q.append(i)

return False

start,end=0,5

print(path(graph,start,end))

**Q9. Directed graph: Print all the paths**

from collections import defaultdict

edges=[(0,1),(0,2),(1,3),(2,4),(3,5),(4,5)]

graph=defaultdict(list)

for u,v in edges:

graph[u].append(v)

def path(start,end,p=[]):

p.append(start)

if start==end:

print(p)

else:

for i in graph[start]:

if i not in p:

path(i,end,p)

p.pop()

start,end=0,5

path(start,end)

**Q10. Find Center of Star Graph**

def findCenter(edges):

if (edges[0][0] in edges[1]):

return edges[0][0]

else:

return edges[0][1]

**DAY:18**

**Q1. Find the Town Judge:**

def findJudge(n,trust):

if n==1:

return 1

d=collections.defaultdict(list)

for i,j in trust:

d[i].append(j)

d[j]

for i in d:

if len(d[i])==0:

j=i

break

else:

return -1

for i in d:

if j not in d[i] and i!=j:

return -1

return j

**Q2. Number of Provinces:**

def findCircleNum(isConnected):

if not isConnected:

return 0

n=len(isConnected)

v=[0]\*n

def dfs(node):

for ne in range(n):

if isConnected[node][ne]==1 and not v[ne]:

v[ne]=1

dfs(ne)

c=0

for i in range(n):

if not v[i]:

v[i]=1

c+=1

dfs(i)

return c

**Q3. Count the number of paths in a graph**

from collections import defaultdict

edges=[(0,1),(0,2),(1,3),(2,4),(3,5),(4,5)]

graph=defaultdict(list)

for u,v in edges:

graph[u].append(v)

def path\_count(start,end,p=None,c=0):

if p is None:

p=[]

p.append(str(start))

if start==end:

print(p)

c+=1

else:

for i in graph[start]:

if i not in p:

c=path\_count(i,end,p,c)

p.pop()

return c

start,end=0,5

print(path\_count(start,end))

**Q4. Print whether a graph is having cycle or not**

def cycle(l):

v=[False]\*len(l)

for i in range(len(l)):

if not v[i]:

q=[(i,-1)]

while q:

node,prev=q.pop(0)

v[node]=True

for i in l[node]:

if not v[i]:

q.append((i,node))

elif i!=prev:

return True

return False

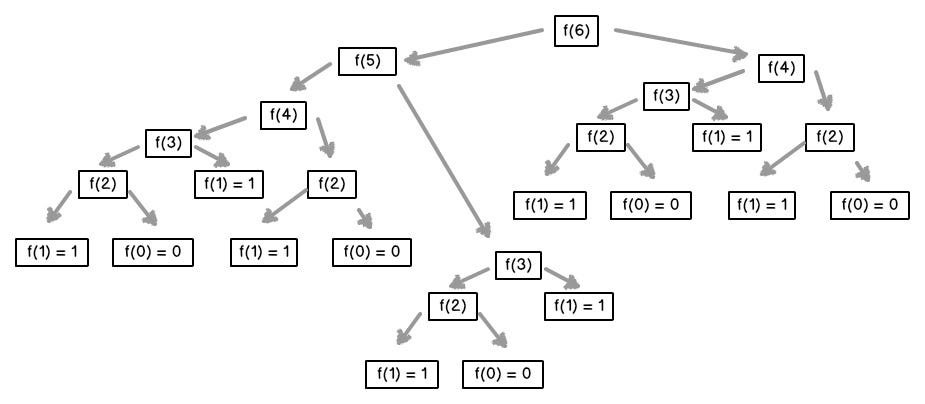
l=[[1],[0,2,3],[1,3],[1,2]]

print(cycle(l))

**DAY:19**

Dynamic Programming

Fibonaaci Series: Time Complexity- O(2n)



def fib(n):

if n==0 or n==1:

return n

return fib(n-1)+fib(n-2)

n=6

print(fib(n))

1. **Memoization:** Recursion (Top-down approach)

Optimized Code: O(n), Space Complexity: O(2n)

**Q1. Fibnacci Series using Dynamic Programming**

def fib(n):

if n==1 or n==0:

return n

if memo[n]!=-1:

return memo[n]

memo[n]=fib(n-1)+fib(n-2)

return memo[n]

n=6

memo=[-1]\*(n+1)

print(fib(n))

1. **Tabulation:** Bottom-Up approach using loops

Time Complexity- O(n)

Space Complexity- O(n)

**Q2. Fibonacci series using tabulation**

n=6

dp=[-1]\*(n+1)

dp[1]=1

dp[0]=0

for i in range(2,n+1):

dp[i]=dp[i-1]+dp[i-2]

print(dp[n])

**Q3. Optimized code by removing the table.**

Time Complexity: O(n), Space Complexity: O(1)

n=6

a,b=0,1

for \_ in range(2,n+1):

fib=a+b

a=b

b=fib

print(fib)

**How to find a DP Problem:**

→Count all the ways

→Max Output

→Min Output

**How to solve:**

→Convert the problem in terms of indices

→Do all the possible operations on the index

→ Count all ways: Find the sum

→ Max Output: Find the max

→ Min Output: Find the min

f(ind)

f(ind==0)

return 1

j1=f(ind-1)--step-1

j2=(ind-2)--step-2

return j1+j2--step-3

**Q4. Climbing Stairs**

def climbStairs(n):

if n==2:

return 2

if n==1:

return 1

p1=2

p2=1

c=0

for i in range(2,n):

c=p1+p2

p2=p1

p1=c

return c

n=3

print(climbStairs(n))

**Q5. Frog Jump**

Recursion: Time Complexity- O(2n)

f(ind)

if(ind==0)return

j1=f(ind-1)+abs(a[ind]-a[ind-1])

j2=f(ind-2)+abs(a[ind]-a[ind-2])

return min(j1,j2)

def frog(ind):

if ind==0:

return 0

j1=frog(ind-1)+abs(rock[ind]-rock[ind-1])

if ind>1:

j2=frog(ind-2)+abs(rock[ind]-rock[ind-2])

return min(j1,j2)

else:

return j1

rock=[30,10,60,10,60,60]

print(frog(len(rock)-1))

**Using Memoization:**

def frog(ind,memo):

if ind==0:

return 0

if memo[ind]!=-1:

return memo[ind]

j1=frog(ind-1,memo)+abs(rock[ind]-rock[ind-1])

if ind>1:

j2=frog(ind-2,memo)+abs(rock[ind]-rock[ind-2])

memo[ind]=min(j1,j2)

return memo[ind]

else:

memo[ind]=j1

return memo[ind]

rock=[30,10,60,10,60,60]

n=len(rock)

memo=[-1]\*(n+1)

print(frog(n-1,memo))

**Using Tabulation:**

rock=[30,10,60,10,60,60]

n=len(rock)

dp=[0]\*n

for i in range(1,n):

j1=dp[i-1]+abs(rock[i]-rock[i-1])

j2=float('inf')

if i>1:

j2=dp[i-2]+abs(rock[i]-rock[i-2])

dp[i]=min(j1,j2)

print(dp[n-1])

**Q6. Coin Change**

a=[1,2,5,10]

t=12

n=len(a)

dp=[[False]\*(t+1) for \_ in range(n+1)]

for i in range(n+1):

dp[i][0]=True

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

for j in range(1,t+1):

if a[i-1]>j:

dp[i][j]=dp[i-1][j]

else:

dp[i][j]=dp[i-1][j] or dp[i-1][j-a[i-1]]

if dp[n][t]:

print("Yes")

else:

print("No")

**DAY:20**

**Q1. Longest Common Subsequence**

**Using Recursion**

def lcs(s1,s2,i,j):

if i==0 or j==0:

return 0

if s1[i-1]==s2[j-1]:

return 1+lcs(s1,s2,i-1,j-1)

else:

return max(lcs(s1,s2,i-1,j),lcs(s1,s2,i,j-1))

s1="abcd"

s2="abce"

print(lcs(s1,s2,len(s1),len(s2)))

**Using Tabulation**

S1=”babbab” , s2=”abaaba”

0 b a b b a b

0 0 0 0 0 0 0 0

a 0 0 1 1 1 1 1

b 0 1 1 2 2 2 2

a 0 1 2 2 2 3 3

a 0 1 2 2 2 3 3

b 0 1 2 3 3 3 4

a 0 1  2  3  3  4  4

s1="abcd"

s2="abce"

n,m=len(s1),len(s2)

dp=[[""]\*(m+1) for \_ in range(n+1)]

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

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

if s1[i-1]==s2[j-1]:

dp[i][j]=dp[i-1][j-1]+s1[i-1]

else:

dp[i][j]=max(dp[i-1][j],dp[i][j-1])

print(len(dp[n][m]))

print(dp[n][m])

**Q2. Infinite Coin Change problem**

2,5----TARGET=11

0 1 2 3 4 5 6 7 8 9 10 11

0 1 0 0 0 0 0 0 0 0 0 0 0

2 1 0 1 0 1 0 1 0 1 0 1 0

5 1 0 1 0 1 1 0  1  0  1  1   0

(2,3,5)----TARGET=13

1 2 3 4 5 6 7 8 9 10 11 12 13

0 0 0 0 0 0 0 0 0 0 0 0 0 0

2 0 0 1 0 2 0 3 0 4 0 5 0 6 0

3 0 0 1 1 2 2 2 0 3 3 5 4 4 5

5 0 0 1 1 2 1 2 2 2 3   2    3    3   3

coins=[1,2,5]

amt=18

dp=[float('inf')]\*(amt+1)

dp[0]=0

for coin in coins:

for i in range(coin,amt+1):

dp[i]=min(dp[i],dp[i-coin]+1)

if dp[amt]!=float('inf'):

print(dp[amt])

else:

print(-1)

**Tries:**

