**LINEAR SEARCH:**

def linear(l,n):

global e

e=0

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

if (l[i]==n):

print("the value located at ",i)

e=1

break

x=int(input("enter the size of array:"))

list=[]

for i in range(0,x):

b=int(input("enter element:"))

list.append(b)

n=int(input("enter the key:"))

r=linear(list,n)

if(e!=1):

print("value not present in the array")

**BINARY SEARCH:**

def bin(l,x,low,high):

global e

e=0

while(low<=high):

mid=(low+high)//2

if(l[mid]==x):

print("the value found at ",mid)

e=1

break

elif(l[mid]>x):

high=mid-1

else:

low=mid+1

n=int(input("enter the size of array:"))

list=[]

for i in range(0,n):

b=int(input("enter element:"))

list.append(b)

x=int(input("enter the key to search:"))

bin(list,x,0,n-1)

if(e!=1):

print("value not present in the array"

**BUBBLE SORT:**

def bubble(l):

temp=0

for i in range (n-1):

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

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

temp=l[j]

l[j]=l[j+1]

l[j+1]=temp

return l

n=int(input("enter the size of array:"))

list=[]

for i in range(0,n):

b=int(input("enter element:"))

list.append(b)

print(bubble(list))

**SELECTION SORT:**

def selection(l):

temp=0

for i in range(n):

p=i

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

if(l[p]>l[j]):

p=j

if(p!=i):

temp=l[p]

l[p]=l[i]

l[i]=temp

return l

n=int(input("enter the size of array:"))

list=[]

for i in range(0,n):

b=int(input("enter element:"))

list.append(b)

print(selection(list))

**INSERTION SORT:**

def insertion(l):

for i in range(1,n):

temp=l[i]

j=i-1

while(j>=0 and temp<l[j]):

l[j+1]=l[j]

j=j-1

l[j+1]=temp

return l

n=int(input("enter the size of array:"))

list=[]

for i in range(0,n):

b=int(input("enter element:"))

list.append(b)

print(insertion(list))

**QUICKSORT:**def quicksort(l,low,high):

if(low<high):

p=partion(l,low,high)

quicksort(l,low,p-1)

quicksort(l,p+1,high)

def partion(l,low,high):

pivot=l[high]

i=low-1

for j in range(low,high):

if(l[j]<=pivot):

i=i+1

l[i],l[j]=l[j],l[i]

l[i+1],l[high]=l[high],l[i+1]

return i+1

n=int(input("enter the size of array:"))

list=[]

for i in range(0,n):

b=int(input("enter element:"))

list.append(b)

quicksort(list,0,n-1)

print(list)

**MERGE SORT:**

def merge(l):

if(len(l)>1):

mid=len(l)//2

left=l[:mid]

right=l[mid:]

merge(left)

merge(right)

i,j,k=0,0,0

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

if(left[i]<right[i]):

l[k]=left[i]

i=i+1

else:

l[k]=right[j]

j=j+1

k=k+1

while(i<len(left)):

l[k]=left[i]

i=i+1

k=k+1

while(j<len(right)):

l[k]=right[j]

j=j+1

k=k+1

return l

n=int(input("enter the size of array:"))

list=[]

for i in range(0,n):

b=int(input("enter element:"))

list.append(b)

print(merge(list))

KNAPSACK USING DP:

def knapsack(w,wt,val):

table=[]

for i in range(n+1):

c=[]

for j in range(w+1):

a=None

c.append(a)

table.append(c)

for i in range(n+1):

for j in range(w+1):

if(i==0 or j==0):

table[i][j]=0

elif(wt[i-1]<=j):

table[i][j]= max(table[i-1][j], val[i-1]+table[i-1][j-wt[i-1]])

else:

table[i][j]=table[i-1][j]

return table[n][w]

n=int(input("enter the size :"))

val=[]

for i in range(0,n):

b=int(input("enter value:"))

val.append(b)

wt=[]

for i in range(0,n):

b=int(input("enter wieight:"))

wt.append(b)

w=int(input("enter the total weightf:"))

print(knapsack(w,wt,val))

**TSP:**

from sys import maxsize

from itertools import permutations

v=int(input("enter the size:"))

def tsp(graph,start):

vertex=[]

for i in range(v):

if i != s:

vertex.append(i)

minpath=maxsize

nextpermutations=permutations(vertex)

for i in nextpermutations:

cost=0

k=s

for j in i:

cost=cost+graph[k][j]

k=j

cost=cost+graph[k][s]

minpath=min(minpath,cost)

return minpath

graph=[]

print("enter the graph........")

for i in range(v):

c=[]

for j in range(v):

b=int(input())

c.append(b)

graph.append(c)

s=0

print("the minimum path is :",tsp(graph,s))

**BINOMIAL COEFFICIENT:**

from math import factorial

row=int(input("enter the no of rows:"))

for i in range(row):

for j in range(row-i+1):

print(end=' ')

for j in range(i+1):

print(factorial(i)//(factorial(j)\*factorial(i-j)),end=' ')

print()

def bin(n,k):

if k>n:

return 0

if k==1 or k==n:

return 1

c=[[0 for i in range(k+1)]for j in range(n+1)]

for i in range(n+1):

for j in range(k+1):

if(j==0):

c[i][j]=1

else:

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

return c[n][k]

n=int(input("enter the n value:"))

k=int(input("enter the K value:"))

print(bin(n,k))

**MTH MIN AND MTH MAX:**

n=int(input("enter the size :"))

x=[]

for i in range(n):

p=int(input("enter the element:"))

x.append(p)

x.sort

m=int(input("enter the mth value:"))

y=int(input("enter the nth value:"))

a=m-1

print("the ",m," th minimum is:",x[a])

x.reverse()

b=y-1

print("the ",y,"th maximum is :",x[b])

**REVERSE A NUMBER:**

x=int(input("enter the number:"))

n=0

while(x!=0):

rem=x%10

n=n\*10+rem

x=x//10

print("the reverse number is :",n)

**SUM OF SUBSETS:**

def subset(x,t):

a=[]

while x:

n=x.pop()

d=t-n

if d in x :

a.append((n,d))

return a

x=[1,2,3,4]

t=5

print(subset(x,t))

**ARMSTRONGNUMBER:**

n=int(input("how many digits number you are going to enter:"))

x=int(input("enter the number:"))

a=x

sum=0

while(x!=0):

rem=x%10

sum=sum+(rem\*\*n)

x=x//10

if(sum==a):

print("it is armstrong number")

else:

print("it is not armstrong")

**STRASENS MATRIX:**

r=int(input("enter the no of rows:"))

c=int(input("enter the no of coluomns:"))

x=[[0 for i in range(r)]for j in range(c)]

a=[[1,2],

[3,4]]

b=[[1,2],

[3,4]]

m1=(a[0][0]+a[1][1])\*(b[0][0]+b[1][1])

m2=(a[1][0]+a[1][1])\*b[0][0]

m3=a[0][0]\*(b[0][1]-b[1][1])

m4=a[1][1]\*(b[1][0]-b[0][0])

m5=(a[0][0]+a[0][1])\*b[1][1]

m6=(a[1][0]-a[1][1])\*b[0][0]+b[0][1]

m7=(a[0][1]-a[1][1])\*(b[1][0]+b[1][1])

x[0][0]=m1+m4-m5+m7

x[0][1]=m3+m5

x[1][0]=m2+m4

x[1][1]=m1+m3-m2+m6

for i in x:

print(i)

**GCD AND LCM:**

n1=int(input("enter the element 1:"))

n2=int(input("enter the element 2:"))

a=[]

b=[]

c=[]

for i in range (1,n1+1):

if(n1%i==0):

a.append(i)

print(a)

for j in range(1,n2+1):

if(n2%j==0):

b.append(j)

print(b)

for i in a:

if i in b:

c.append(i)

print(c)

x=c[-1]

print("the gcd is :",x)

lcm=(n1\*n2)//x

print("the lcm is :",lcm)

**MST :**

INF=99999

N=5

g=[[0,19,5,0,0],

[19,0,5,9,2],

[5,5,0,2,6],

[0,9,2,0,1],

[0,2,6,1,0]]

visited=[0,0,0,0,0]

e=0

s=0

visited[0]=True

print("edge:weight")

while(e<N-1):

min=INF

a=0

b=0

for i in range(N):

if(visited[i]):

for j in range(N):

if (not visited[j] ) and g[i][j]:

if min>g[i][j]:

min=g[i][j]

a=i

b=j

s=s+g[a][b]

print(a,"-->",b,":",g[a][b])

visited[b]=True

e=e+1

print("the minimum cost is :",s)

**OBST:**

def opcost(f,i,j):

if j<i:

return 0

if j==i:

return f[i]

fsum=sum(f,i,j)

min=9999

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

cost=opcost(f,i,r-1)+opcost(f,r+1,j)

if min>cost:

min=cost

return min+fsum

def obst(f,k,n):

return opcost(f,0,n-1)

def sum(f,i,j):

s=0

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

s=s+f[r]

return s

k=[10,12,16,21]

f=[4,2,6,3]

n=len(k)

print("the minimum cost is :",obst(f,k,n))

**HAMILTONIAN CIRCUIT:**

def valid(path,v,graph,p):

if(graph[path[p-1]][v]==0):

return False

if v in path:

return False

return True

def hami(path,graph,p):

if(p==len(graph)):

if(graph[path[p-1]][path[0]]==1):

return True

return False

for v in range(1,len(graph)):

if(valid(path,v,graph,p)):

path[p]=v

if(hami(path,graph,p+1)):

return True

return False

graph=[[0,1,1,0,1],

[1,0,1,1,1],

[1,1,0,1,0],

[0,1,1,0,1],

[1,1,0,1,0]]

path=[ 0 for i in range(len(graph))]

path[0]=0

if(hami(path,graph,1)):

print(path)

else:

print('no solution')

**N-QUEENS:**

def safe(board,r,c):

for i in range(c):

if(board[r][i]==1):

return False

for x,y in zip(range(r,-1,-1),(range(c,-1,-1))):

if (board[x][y]==1):

return False

for x,y in zip(range(r,n,1),range(c,-1,-1)):

if board[x][y]==1:

return False

return True

def generate(board,c):

global n

if c>=n:

return True

for i in range(n):

if(safe(board,i,c))==True:

board[i][c]=1

if (generate(board,c+1)==True):

return True

return False

n=int(input("enter the size:"))

board=[[0 for i in range(n)]for j in range(n)]

if(generate(board,0)==False):

print("no solution")

else:

print(board)

**FLOYDS ALGORITHM:**

v=4

INF=999

def floyds(graph):

dist=list(graph)

for k in range(v):

for i in range(v):

for j in range(v):

dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j])

printsol(dist)

def printsol(dist):

for i in range(v):

for j in range(v):

print(dist[i][j],end=' ')

print('')

graph=[[0,9,-4,INF],

[6,0,INF,2],

[INF,5,0,INF],

[INF,INF,1,0]]

floyds(graph)

**MIN AND MAX BY DIVIDE AND CONQUOR METHOD:**

def divide(l):

if(len(l)>1):

mid=len(l)//2

left=l[:mid]

right=l[mid:]

divide(left)

divide(right)

i,j,k=0,0,0

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

if(left[i]<right[i]):

l[k]=left[i]

i=i+1

else:

l[k]=right[j]

j=j+1

k=k+1

while(i<len(left)):

l[k]=left[i]

i=i+1

k=k+1

while(j<len(right)):

l[k]=right[j]

j=j+1

k=k+1

return l

n=int(input("enter the size of array:"))

list=[]

for i in range(0,n):

b=int(input("enter element:"))

list.append(b)

divide(list)

print("the minimum element is :",list[0])

print("the maximum element is :",list[-1])