1. Rezolvați problema programării spectacolelor într-o singură sală utilizând o coadă cu priorități.

import queue  
  
fisier=input()  
f= open(fisier,"r")  
pq= queue.PriorityQueue()  
i=1  
for linie in f.readlines():  
 aux=linie.split(" ")  
 s=aux[1]  
 #scapam de \n  
 if(s[len(s)-1]=='\n'):  
 s=s[:-1]  
 #in pq avem tupluri de forma (ora final, ora inceput, numarul spectacolului)  
 pq.put((s,aux[0],i))  
 i+=1  
  
f.close()  
  
#selectie spectacole  
s=1  
t=pq.get()  
l=[(t[1],t[0],t[2])]  
while not pq.empty():  
 t=pq.get()  
 if(t[1]>=l[len(l)-1][1]):  
 l.append(t)  
 s+=1  
  
#afisare  
print("Numarul maxim de spectacole este ",s)  
for spectacol in l:  
 print(spectacol[0],"-",spectacol[1],"Spectacol",spectacol[2])

2. Se consideră o listă sortată crescător de numere întregi. Scrieți o funcție cu complexitate minimă care să furnizeze numărul de apariții ale unei valori în listă. De exemplu, în lista [1, 1, 2, 2, 2, 2, 4, 4, 4, 4, 5] valoarea 2 apare de 4 ori.

def cb\_st(x,v):  
 i=-1  
 l=len(v)  
 st=0  
 dr=l  
 while(st<=dr):  
 mij=(st+dr)//2  
 if(v[mij]>=x):  
 i=mij  
 dr=mij-1  
 else:  
 st=mij+1  
 return i  
  
def cb\_dr(x,v):  
 i = -1  
 l = len(v)  
 st = 0  
 dr = l  
 while (st <= dr):  
 mij = (st + dr) // 2  
 if (v[mij] <= x):  
 i = mij  
 st = mij + 1  
 else:  
 dr=mij-1  
 return i  
  
x=int(input("x="))  
v=[int(x) for x in input().split()]  
st=cb\_st(x,v)  
dr=cb\_dr(x,v)  
print(st,dr)  
print(dr-st+1)

3. Scrieți un algoritm de tip Divide et Impera pentru a rezolva problema turnurilor din Hanoi

def hanoi(n,sursa,aux,dest):  
 if(n==0):  
 return  
 hanoi(n-1,sursa,dest,aux)  
 print(n,":",sursa,"->",dest)  
 hanoi(n-1,aux,sursa,dest)  
  
n=int(input())  
hanoi(n,'A','B','C')

1. Fie A un multiset format din n numere naturale nenule și S un număr natural nenul.

Folosind metoda Backtracking, să se afișeze toate submultiseturile lui A care au suma

elementelor egală cu S.

from multiset import Multiset  
  
def bkt(k,scrt):  
 global s,v,r,folosit,l  
 if(scrt==s):  
 aux=[r[i] for i in range(1,k) if r[i]!=0]  
 aux= Multiset(aux)  
 if(aux not in l):  
 l.append(aux)  
 else:  
 for i in range(k,n):  
 if(scrt+v[i]<=s and folosit[i]==0):  
 r[k]=v[i]  
 folosit[i]=1  
 bkt(k+1,scrt+v[i])  
 folosit[i]=0  
 r[k]=0  
  
  
  
s=int(input())  
  
v=[0]  
aux=[int(x) for x in input().split()]  
n=len(aux)+1  
v.extend(aux)  
v=sorted(v)  
  
r=[0]\*n  
folosit=[0]\*n  
l=[]  
  
bkt(1,0)  
  
for set in l:  
 print(set)

2. Fie A un multiset format din n numere naturale nenule și S un număr natural nenul.

Folosind metoda programării dinamice, să se afișeze un submultiset a lui A care are suma

elementelor egală cu S.

from multiset import \*  
  
target=int(input())  
v=[int(x) for x in input().split()]  
n=len(v)  
dp=[[0]\*(target+1) for i in range(n+1)]  
for i in range(n+1):  
 dp[i][0]=1  
for i in range(1,n+1):  
 for j in range(1,target+1):  
 if(v[i-1]>j):  
 dp[i][j]=dp[i-1][j]  
 else:  
 if(dp[i-1][j-v[i-1]]==1):  
 dp[i][j]=1  
 elif(dp[i-1][j]==1):  
 dp[i][j]=1  
  
#for linie in dp:  
 #print(linie)  
if(dp[n][target]==1): #daca exista un submultiset, refacem un posibil traseu in matrice pentru a-l afla  
 for i in range(n+1):  
 if(dp[i][target]==1):  
 ii=i #prima linie de la care targetul a fost obtinut  
 break  
 jj=target  
 s=Multiset()  
 while(ii>0 and jj>0):  
 s.add(v[ii-1]) #adaugam elemntul curent in multiset  
 jj=jj-v[ii-1] #mergem la elementul corespunzator punctului d[i-1][j-v[i-1]]  
 ii-=1  
 print(s)

3. Folosind doar metoda backtracking, generați toate subșirurile crescătoare maximale

ale un șir format din n numere întregi.

def bkt(k,scrt,index):  
 global v,s,n,lmax,lista  
  
 if scrt>=lmax and scrt != 0:  
 r = [0] \* (scrt)  
 for i in range(scrt):  
 r[i] = s[i]  
 #print(r)  
 '''  
 if(index<n-1):  
 if(v[index]<v[index-1]):  
 #print("#",end="")  
 print(r)  
 elif(index == n):  
 # print("#", end="")  
 print(r)  
 '''  
  
 if lmax < scrt:  
 #print(r)  
 lista = []  
 lista.append(r)  
 lmax = scrt  
 if(k==0):  
 for i in range(n):  
 s[k]=v[i]  
 bkt(k+1,scrt+1,i+1)  
  
 else:  
 for i in range(index,n):  
 if v[i]>=s[k-1]:  
 s[k]=v[i]  
 bkt(k+1,scrt+1,i+1)  
  
  
  
lmax=-9  
v=[int(x) for x in input().split()]  
n=len(v)  
s=[-1]\*(n)  
lista=[] #lista cu toate sirurile de lungime maxima  
bkt(0,0,1)  
# k==pozitia pe stiva, scrt== lungimea curenta, index== pozitia din sir a ultimului element de pe stiva  
print(lmax)  
for sir in lista:  
 print(sir)

4. Optimizați algoritmul de la problema anterioară, utilizând metoda programării

dinamice pentru a determina lungimea maximă a unui subșir crescător al șirului dat .

v=[int(x) for x in input().split()]  
n=len(v)  
p=[0]\*n  
p[0]=1  
max=1  
  
for i in range(1,n):  
 pmax=0  
 for j in range(i):  
 if(v[j]<=v[i] and p[j]>pmax):  
 pmax=p[j]  
 p[i]=pmax+1  
 if(max<p[i]):  
 max=p[i]  
  
print(max)