LAB 1

Maximum Subarray Sum using Divide and Conquer algorithm

```
n = int(input("test cases: "))
for x in range(n):
    c = []
    d = []
    print("enter elements ")
    a = [int(i) for i in input().split()]
    for i in range(len(a)):
        for j in range(i,len(a)+1):
            b = a[i:j]
            c.append(b)
            d.append(sum(b))
    print("sub array with maximum sum is :
" ,c[d.index(max(d))])
    print("maximum sum is : " ,max(d))
```

Reverse Pairs

```
n = int(input("test cases: "))
for x in range(n):
    print("elements: ")
    a = [int(i) for i in input().split()]
    if len(a)>50000 or len(a)<2:
        print("enter proper array")
    else:
        count = 0
        for i in range(len(a)):
            for j in range(i+1,len(a)):
                if a[i]>2*a[j]:
                    print("(",i,j,")")
                    count +=1
        print("number of reverse pairs:
",count)
```

Closest Pair of Points using Divide and Conquer algorithm

```
import math
def distance(a,b):
    return math.sqrt(pow((a[0]-
b[0]),2)+pow((a[1]-b[1]),2))

n = int(input("test cases: "))
for x in range(n):
    print("enter points: ")
    a = []
    while(True):
        s = [int(i) for i in input().split()]
        if len(s)==0 or s=="":
            break
        if len(s)!=2 :
            print("enter proper point")
        else:
            a.append(s)
    d = []
    for i in range(len(a)):
        for j in range(i+1,len(a)):
            d.append(distance(a[i],a[j]))
    print("shortest distance: ",min(d)
```

LAB 2

Marc's Cake Walk

```
n = int(input("test cases: "))
for x in range(n):
    print("enter points: ")
    a = [int(i) for i in input().split()]
    a.sort()
    a.reverse()
    miles = 0
    for i in range(len(a)):
        miles += pow(2,i)*a[i]
    print("miles: ",miles)
```

Find maximum meetings in one room

```
n = int(input("test cases: "))
for x in range(n):
    print("enter start time array: ")
    s = [int(i) for i in input().split()]
    print("enter finish time array: ")
    f = [int(i) for i in input().split()]
    time = list(zip(s,f))
    time.sort(key=lambda x : x[1])
    print(time)
    a = 0
    b = 0
    count = 0
    l = []
    for (start,finish) in time:
        if start<finish and start > b:
            print(start , finish)
            b = finish
            count+=1
            k = list(zip(s,f))
            for i in range(len(k)):
```

```
            if k[i][0]==start and
k[i][1]==finish:
                print(k[i][0],":",k[i][1],"-
>",i+1, "meeting")
                l.append(str(i+1))
    print("total ",count,"can be conducted ."
)
    print("meetings are : ",end=" ")
    print(' '.join(l))
```

Connecting different towers in a city

```
import heapq
def minCost(arr, n):
    heapq.heapify(arr)
    res = 0
    while(len(arr) > 1):
        first = heapq.heappop(arr)
        second = heapq.heappop(arr)
        res += first + second
    heapq.heappush(arr, first + second)
    return res
lengths = [int(i) for i in input("entr the
lengths: ").split()] # 1 2 3 5 6 9
print("Total cost for connecting ropes is "
+str(minCost(lengths, len(lengths))))
```

Nanda's tour

```
def minDis(dist,t):
    m=999999
    for i in range(noOfCities):
        if dist[i]<m and t[i]!=1:
            m=dist[i]
            index=i
    return index

noOfCities = int(input("Enter no of cities:
"))
print("Enter the Cost Matrix")
cost_matrix =  []
for i in range(noOfCities):
    cost_matrix.append([int(i) for i in
input().split()])
starting_city = int(input("Enter the Starting
City "))
t = [0 for i in range(noOfCities)]
path=[-1 for i in range(noOfCities)]
path[starting_city]=-1
dist = [999999 for i in range(noOfCities)]
dist[starting_city]=0
for i in range(noOfCities):
    u = minDis(dist,t)
    t[u]=1
    for v in range(noOfCities):
        if cost_matrix[u][v]>0 and t[v]==0
and dist[u] + cost_matrix[u][v] < dist[v]:
            dist[v]=dist[u]+cost_matrix[u][v]
            path[v]=u
for i in range(noOfCities):
    if i!= starting_city:
        print("Distance of node {0} =
{1}".format(i,dist[i]))
        l=[i]
        p=i
        while(p!=starting_city):
            p=path[p]
            l.append(p)
        l= [str(p) for p in l]
        print("Path = {0}".format('<-'.join(l)))
```

LAB 3

Palindrome Partitioning

```
def isPalindrome(x):
        return x == x[::-1]

def palPart(a, i, j):
        if i >= j or isPalindrome(a[i:j+1]):
                return 0
        ans = float('inf')
        for k in range(i, j):

        count=(1+palPart(a,i,k)+palPart(a,k
+1,j))
                ans = min(ans, count)
        return ans
n = int(input("test cases: "))
for x in range(n):
    a = input("string: ")
    print("Min cuts needed for Partitioning
is ",palPart(a,0,len(a)-1))
```

Mobile Numeric Keypad Problem

```
keypad = [['1', '2', '3'],['4', '5', '6'],['7', '8',
'9'],['*', '0', '#']]
row = [0, 0, -1, 0, 1]
col = [0, -1, 0, 1, 0]
def getCountUtil(keypad, i, j, n):
    if (n == 1):
            return 1
    k = 0
    move = 0
    ro = 0
    co = 0
    totalCount = 0
    for move in range(5):
        ro = i + row[move]
        co = j + col[move]
        if (ro >= 0 and ro <= 3 and co >= 0
and co <= 2 and keypad[ro][co] != '*' and
keypad[ro][co] != '#'):
            totalCount +=
getCountUtil(keypad, ro, co, n - 1)
    return totalCount

def getCount(keypad, n):
    if (n == 1):
            return 10
    i = 0
    j = 0
    totalCount = 0
    for i in range(4):#loop for rows
        for j in range(3):#loop for cols
            if (keypad[i][j] != '*' and
keypad[i][j] != '#'):
                totalCount +=
getCountUtil(keypad, i, j, n)
    return totalCount

n = int(input("test cases: "))
for x in range(n):
    l = int(input("n: "))
    if l<=0:
        print(0)
    else:
        for i in range(1,l+1):
            print("Count for length ",i,"is : "
,getCount(keypad, i))
```

The painter's partition problem

```
def partition(arr, n, k):
    if k == 1:
        return sum(arr[:n])
    if n == 1:
        return arr[0]
    best = 100000000
    for i in range(1, n + 1):
        best = min(best,max(partition(arr, i, k
- 1),sum(arr[i:n])))
    return best

n = int(input("test cases: "))
for x in range(n):
    k = int(input("painters: "))
    a = [int(i) for i in input("elements:
").split()]
    print(partition(a,len(a),k))
```

Number of palindromic paths in a matrix

```
pal=[]
def Path(st, a, i, j, m, n):
    if (j<m-1 or i<n-1) :
        if (i<n-1):
            Path(st+a[i][j], a ,i+1, j, m, n)
        if (j < m-1):
            Path(st+a[i][j] , a ,i, j+1, m, n)
    else :
        st = st + a[n-1][m-1]
        if st==st[::-1]:
            print(st)
            pal.append(st)
    return pal
n = int(input("test cases: "))
for x in range(n):
    s = []
    print("data: ")
    while(True):
        a = input().split()
        if a==[]:
            break
        else:
            s.append(list(a))
    st = ""
    pal = Path(st, s, 0, 0, len(s[0]), len(s))
    print("Number of palindromic paths in a
matrix: ",len(pal))
```

'''
a a a b
b a a a
a b b a
'''

LAB 4

Length of the Longest Arithmetic Progression (LLAP)

```
from itertools import combinations
n = int(input("test cases: "))
for x in range(n):
    a = [int(i) for i in input("array: ").split()]
    l = []
    s = []
    c = []
    for i in range(2,len(a)+1):
        for j in combinations(a,i):
            c.append(list(j))
        for k in c:
            res = [k[i+1]-k[i] for i in
range(len(k)-1)]
            if len(set(res))==1:
                l.append(len(k))
                s.append(k)
    print("max length: ",max(l))
    print(s[l.index(max(l))])
```

Painting Fence Algorithm

```
n = int(input("n: "))
k = int(input("k: "))
dp = [0] * (n + 1)
total = k
dp[1] = k
dp[2] = k * k
for i in range(3,n+1):
    dp[i] = ((k - 1) * (dp[i - 1] + dp[i - 2]))
% 1000000007
print("total number of ways: ",dp[n])
```

Probability of reaching a point with 2 or 3 steps at a time

```
n = int(input("n: "))
p = float(input("p:"))
dp =[0] * (n + 1)
dp[0] = 1
dp[1] = 0
dp[2] = p
dp[3] = 1 - p
for i in range(4,n+1) :
    dp[i] = (p) * dp[i-2]+(1-p)*dp[i-3]
print("total probability : ",round(dp[n], 2))
```

Longest Geometric Progression

```
from itertools import combinations
n = int(input("test cases: "))
for x in range(n):
    a = [int(i) for i in input("array: ").split()]
    l = []
    s = []
    c = []
    for i in range(2,len(a)+1):
        for j in combinations(a,i):
            c.append(list(j))
        for k in c:
            res = [k[i+1]/k[i] for i in
range(len(k)-1)]
            if len(set(res))==1:
                l.append(len(k))
                s.append(k)
    print("max length: ",max(l))
    print(s[l.index(max(l))])
```

LAB 5

Knapsack (prelab)

```
def display(a):
    for i in range(len(w)):
        for j in range(m+1):
            print(a[i][j],end=" ")
        print()
w = [int(i) for i in input("weights:
").split()]
p = [int(i) for i in input("profits: ").split()]
w.insert(0,0)
p.insert(0,0)
m = int(input("bag weight: "))
m=8
a = [[0 for i in range(m+2)] for j in
range(len(w)+1)]
for i in range(1,len(w)):
    for j in range(1,m+1):
        k = j-w[i]
        if k<0:
            a[i][j] = a[i-1][j]
        else:
            a[i][j] = max(a[i-1][j] , a[i-1][k] +
p[i] )
display(a)
```

number of subarrays (bank problem)

```
def check(a):
    p = 0
    n = 0
    for i in range(len(a)):
        if a[i]>0:
            p+=1
        elif a[i]<0:
            n+=1
    return p,n
n = int(input("test cases: "))
for x in range(n):
    a = [int(i) for i in input("enter elements:
").split()]
    count = 0
    e = []
    for i in range(len(a)):
        for j in range(i,len(a)+1):
            b = a[i:j]
            c,d = check(b)
            if c==d and c!=0 and d!=0:
                count+=1
                e.append(b)
    print("number of subarrays: ",count)
    print("the sub arrays are: ",e)
```

0-1 knapsack (inlab 1)

```
n = int(input("number of items: "))
print("enter elements: ")
a = []
for i in range(n):
    s = [int(i) for i in input().split()]
    a.append(s)
w = [c[1] for c in a]
p = [c[2] for c in a]
m = int(input("bag size: "))
for i in range(len(p)):
    if p[i]==0 and min(p)!=1:
        p[i] = min(p)-1
    if p[i]==0 and min(p)==1:
        p[i]=1
w,p = zip(*sorted(zip(w, p), key=lambda t:
t[1]/t[0]))
u = m
profit = 0
c = [0]*len(w)
for i in range(len(w)):
    if w[i]<=u:
        c[i]=1
        profit+=p[i]
        u = u-w[i]
print("profit: ",profit)
print("inclusion array: ",c)
print("number of weights includeded:
",c.count(1))
```

OBST (Raju problem)

```
# cost of optimal binary search tree
def optCost(freq, i, j):
    if j < i:
        return 0
    if j == i:
        return freq[i]
    fsum = sum(freq[i:j+1])
    Min = 999999999999
    for r in range(i, j + 1):
        cost = (optCost(freq, i, r - 1) +
optCost(freq, r + 1, j))
        if cost < Min:
            Min = cost
    return Min + fsum
n = int(input("test cases: "))
for kaushik in range(n):
    k = [int(i) for i in input("keys: ").split()]
    f = [int(i) for i in input("freq: ").split()]
    print("Cost of Optimal BST is:
",optCost(f,0,len(k)-1))
```

Minimum number of jumps to reach end

```
def minJumps(arr, l, h):
    if (h == l):
        return 0
    if (arr[l] == 0):
        return float('inf')
    mini = float('inf')
    for i in range(l + 1, h + 1):
        if (i < l + arr[l] + 1):
            jumps = minJumps(arr, i, h)
            if (jumps != float('inf') and jumps +
1 < mini):
                mini = jumps + 1
    return mini
arr = [int(i) for i in input("enter elements:
").split()]
print('Minimum number of jumps are: ',
minJumps(arr, 0, len(arr)-1))
```

Cutting a Rod(Rod pieces price)

```
INT_MIN = -32767
def cutRod(price, n):
            val = [0 for x in range(n+1)]
            val[0] = 0
            for i in range(1, n+1):
                max_val = INT_MIN
                for j in range(i):
                            max_val =
max(max_val, price[j] + val[i-j-1])
                val[i] = max_val
            return val[n]
s = [int(i) for i in input("enter prices:
").split()]
size = len(s)
print("Maximum Obtainable Value is " +
str(cutRod(s, size)))
```

LAB 6

Traveling Swamp Thing Problem (CPP) (execution not sure)

```
# include <bits/stdc++.h>
using namespace std;
int n, m, e;
int dp[20][101][1<<15];
struct edge
{
   int v, d, e;
};
vector < edge > V[20];

int recursion(int last, int energy, int
visited)
{
   if(energy < 0)
      return 1e9;
   else if((visited == ((1<<n)-1)) &&
energy >= 0)
      return 0;
   int answer = 1e9;
   if(dp[last][energy][visited] != -1)
      return dp[last][energy][visited];
   for(int i=0; i<V[last].size(); i++)
   {
      if((visited & (1 << V[last][i].v)))
         continue;
      answer = min(answer, V[last][i].d +
recursion(V[last][i].v, energy - V[last][i].e,
(visited | (1<<V[last][i].v)))) ;
   }
   dp[last][energy][visited] = answer;
   return answer;
}

int main(void)
{
         cin>>n>>m>>e;
   //scanf("%d %d %d",&n, &m, &e);
   for(int i=0; i<20; i++)
      for(int j=0; j<101; j++)
         for(int k=0; k<(1<<15); k++)
            dp[i][j][k] = -1;
   struct edge temp, temp2;
   while(m--)
   {
      int a, b, d, e;
      cin>>a>>b>>d>>e;
      //scanf("%d %d %d %d", &a, &b,
&d, &e);
      a--, b--;
      temp.v = b, temp.d = d, temp.e = e;
      temp2.v = a, temp2.d = d, temp2.e =
e;
      V[a].push_back(temp);
      V[b].push_back(temp2);
   }
   int answer = 1e9;
   //for(int i=0;i<n;++i)
   answer = min(answer, recursion(0, e,
1));
   if(answer == 1e9)
      cout<<"-1\n";
   else
      cout<<answer<<endl;
   return 0;
}
```

Subsets (must not contain duplicate subsets.)

```
from itertools import combinations
n = int(input("test cases: "))
for x in range(n):
   a = [int(i) for i in input("elements:
").split()]
   b = []
   for i in range(len(a)+1):
      for j in combinations(a,i):
         b.append(list(j))
   for i in b:
      print(i)
   print("total subsets: ",len(b))
```

Counting Bits

```
n = int(input("test cases: "))
for x in range(n):
    a = int(input("number: "))
    b = []
    for i in range(a+1):
        i = bin(i)
        i = str(i)[2:]
        b.append(i.count('1'))
    print(b)
```

Integer Replacement

```
n = int(input("test cases: "))
for x in range(n):
    a = int(input("number: "))
    count = 0
    while(a!=1):
        if a%2==0:
            a = int(a/2)
        else:
            a = a+1
        count+=1
    print(count)
```

Number Complement

```
n = int(input("test cases: "))
for x in range(n):
    a = int(input("number: "))
    s = str(bin(a))
    res = 0
    num = 1
    for i in s[::-1]:
        if i == "b":
            break
        elif i =="0":
            res+=num
        num*=2
    print("result: ",res)
```

LAB 7

Check whether the word exist in the matrix or not.

```
class wordmatrix:
    def __init__(self,n):
        self.solution = [[0 for i in range(n)]
for j in range(n)]
        self.path = 1
    def searchword(self,mat,word):
        for i in range(len(mat)):
            for j in range(len(mat)):
                if
self.search(mat,word,i,j,0,len(mat)):
                    return True
        return False
    def
search(self,matrix,word,row,col,index,N):
        if (self.solution[row][col]!=0 or
word[index]!=matrix[row][col]):
            return False
        if (index == len(word)-1 ):
            self.solution[row][col] = self.path
            self.path+=1
            return True
        self.solution[row][col] = self.path
        self.path+=1
        if (row+1<N and self.search(matrix,
word, row + 1, col, index + 1, N)):
            return True
        if (row-1>=0 and self.search(matrix,
word, row - 1, col, index + 1, N)):
            return True
        if (col+1< N and self.search(matrix,
word, row, col + 1, index + 1, N)):
            return True
        if (col-1>=0 and self.search(matrix,
word, row, col - 1, index + 1, N)):
            return True
        if (row-1>=0 and col+1<N and
self.search(matrix, word, row-1, col+1,
index+1, N)):
            return True
        if (row-1>=0 and col-1>=0 and
self.search(matrix, word, row-1, col-1,
index+1, N)) :
            return True
        if (row+1<N and col-1>=0 and
self.search(matrix, word, row+1, col-1,
index+1, N)) :
            return True
        if (row+1<N and col+1<N and
self.search(matrix, word, row+1, col+1,
index+1, N)):
            return True
        self.solution[row][col] = 0
        self.path-=1
        return False
    def display(self):
        for i in range(len(self.solution)):
            for j in range(len(self.solution)):
                print(self.solution[i][j],end=" ")
            print()
a = []
print("elements: ")
while(True):
    s = list(input())
    if s!=[]:
        a.append(s)
    else:
        break

w = wordmatrix(len(a))
key = input("search word: ")
if w.searchword(a,key):
    w.display()
else:
    print("no match found")
'''
tzxcd
ahnzx
hwoio
ornrn
abrin
'''
```

8Queens (execution doubt)

```
def solve(matrix):
    rows = set()
    cols = set()
    diags = set()
    rev_diags = set()
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            if matrix[i][j]:
                rows.add(i)
                cols.add(j)
                diags.add(i - j)
                rev_diags.add(i + j)
```

```
    return len(rows) == len(cols) ==
len(diags) == len(rev_diags) ==
len(matrix)

n = int(input("test cases: "))
for x in range(n):
   print("data: ")
   a = []
   for i in range(8):
      a.append(int(list(input())[1]))
   m = [[0 for i in range(8)] for j in
range(8)]
   for i in range(8):
      m[i][a[i]-1] = 1
   for i in range(8):
      for j in range(8):
         print(m[i][j],end=" ")
      print()
   if(solve(m)):
      print("valid")
   else:
      print("not valid")
```

all possible subsets that are selected from a given array whose sum adds up to a Given number K (subarray sum)

```
from itertools import combinations
n = int(input("test cases: "))
for x in range(n):
   a = [int(i) for i in input("data: ").split()]
   k = int(input("k: "))
   b = []
   for i in range(len(a)):
      for j in combinations(a,i):
         b.append(list(j))
   for j in b:
      if sum(j)==k:
         print(j)
```

find the non-empty subset Such that its sum is closest to zero (subarray min sum)

```
from itertools import combinations
n = int(input("test cases: "))
for x in range(n):
   a = [int(i) for i in input("data: ").split()]
   b = []
   c = []
d =[]
for i in range(1,len(a)):
   for j in combinations(a,i):
      b = list(j)
      k = sum(b)-0
      if k>=0:
         c.append(k)
         d.append(b)
print(min(c))
print(d[c.index(min(c))])
```

LAB 8

m Coloring Problem

```
def isSafe(graph, color):
    for i in range(len(graph)):
        for j in range(i + 1, len(graph)):
            if (graph[i][j] and color[j] ==
color[i]):
                return False
    return True
def graphColoring(graph, m, i, color):
    if (i == len(graph)):
        if (isSafe(graph, color)):
            printSolution(color)
            return True
        return False
    for j in range(1, m + 1):
        color[i] = j
        if (graphColoring(graph, m, i + 1,
color)):
            return True
        color[i] = 0
    return False
def printSolution(color):
    print("Solution Exists:" " Following are
the assigned colors ")
    for i in range(len(color)):
        print(color[i],end=" ")

a = []
print("elements: ")
while(True):
    s = [int(i) for i in input().split()]
    if s!=[]:
        a.append(s)
    else:
        break
m = int(input("m: "))
color = [0 for i in range(len(a))]
if (not graphColoring(a, m, 0, color)):
            print ("Solution does not exist")
'''
0 1 1 1
1 0 1 0
1 1 0 1
1 0 1 0
'''
```

Hack the money

```
def solve(n,curr):
    if curr==n:
        return True
    if curr>n:
        return False
    return solve(n,curr*10) or
solve(n,curr*20)
n = int(input("test cases: "))
for x in range(n):
    a = int(input("number: "))
    if a==1:
        print("no")
    else:
        if(solve(a,1)):
            print("no")
        else:
            print("yes")
```

all possible paths from top left to bottom right of a mXn matrix

```
def printAllPathsUtil(mat, i, j, m, n, path,
pi):
    if (i == m - 1):
        for k in range(j, n):
            path[pi + k - j] = mat[i][k]
        for l in range(pi + n - j):
            print(path[l], end = " ")
        print()
        return
    if (j == n - 1):
        for k in range(i, m):
            path[pi + k - i] = mat[k][j]
        for l in range(pi + m - i):
            print(path[l], end = " ")
        print()
        return
    path[pi] = mat[i][j]
    printAllPathsUtil(mat, i + 1, j, m, n,
path, pi + 1)
    printAllPathsUtil(mat, i, j + 1, m, n,
path, pi + 1)
def printAllPaths(mat, m, n):
    path = [0 for i in range(m + n)]
    printAllPathsUtil(mat, 0, 0, m, n, path,
0)
```

```
a = []
print("elements: ")
while(True):
    s = [int(i) for i in input().split()]
    if s!=[]:
        a.append(s)
    else:
        break
printAllPaths(a, len(a), len(a[0]))
```

generate all combinations of well formed parentheses of length 2*n.

```
def generate(result, s, op, close, n):
    if op == n and close == n:
        result.append(s)
        return
    if op < n:
        generate(result, s + "(", op + 1, close,
n)
    if close < op:
        generate(result, s + ")", op, close + 1,
n)
t = int(input("test cases: "))
for x in range(t):
    n = int(input("n: "))
    if n>0:
        result = []
        generate(result,"", 0,0,n)
        for i in result:
            print(i)
    else:
        print("enter proper number")
```