Set-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)
```

Set-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))
```

Set-3

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))))
```

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)
```

Set-4

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))
```

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))
```

Set-5

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()]
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))])
```

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)
```

Set-6

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))
```

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)
```

Set-7

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))
```

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))
```

Set-8

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)
```

Set-9

```
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
'''
```

Set-10

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")
```