# Problem 1: Greedy algorithm

There is one meeting room in a firm. There are N meetings in the form of (start[i], end[i]) where start[i] is start time of meeting i and end[i] is finish time of meeting i.  
What is the maximum number of meetings that can be accommodated in the meeting room when only one meeting can be held in the meeting room at a particular time?

Note: Start time of one chosen meeting can't be equal to the end time of the other chosen meeting.

Example 1:

Input:

N = 6

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

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

Output:

4

Explanation:

Maximum four meetings can be held with

given start and end timings.

The meetings are - (1, 2),(3, 4), (5,7) and (8,9)

Example 2:

Input:

N = 3

start[] = {10, 12, 20}

end[] = {20, 25, 30}

Output:

1

Explanation:

Only one meetings can be held

with given start and end timings.

Your Task :  
Complete the function maxMeetings()that takes two arrays start[] and end[] along with their size N as input parameters and returns the maximum number of meetings that can be held in the meeting room.

Expected Time Complexity : O(N\*LogN)  
Expected Auxilliary Space : O(N)

Constraints:  
1 ≤ N ≤ 105  
0 ≤ start[i] < end[i] ≤ 105

# Problem 2: Greedy algorithm

Given a set of N jobs where each jobi has a deadline and profit associated with it.

Each job takes *1* unit of time to complete and only one job can be scheduled at a time. We earn the profit associated with job if and only if the job is completed by its deadline.

Find the number of jobs done and the maximum profit.

Note: Jobs will be given in the form (Jobid, Deadline, Profit) associated with that Job. Deadline of the job is the time before which job needs to be completed to earn the profit.

Example 1:

Input:

N = 4

Jobs = {(1,4,20),(2,1,10),(3,1,40),(4,1,30)}

Output:

2 60

Explanation:

Job1 and Job3 can be done with

maximum profit of 60 (20+40).

Example 2:

Input:

N = 5

Jobs = {(1,2,100),(2,1,19),(3,2,27),

(4,1,25),(5,1,15)}

Output:

2 127

Explanation:

2 jobs can be done with

maximum profit of 127 (100+27).

Your Task :  
You don't need to read input.. Your task is to complete the function JobScheduling() which takes an integer N and an array of Jobs(Job id, Deadline, Profit) as input and returns the count of jobs and maximum profit as a list or vector of 2 elements.

Expected Time Complexity: O(NlogN)  
Expected Auxilliary Space: O(N)

Constraints:  
1 <= N <= 105  
1 <= Deadline <= N  
1 <= Profit <= 500

# Problem 3: repeat previous problem 2 using Dynamic Programming

# Problem 4: Greedy algorithm

You are given N pairs of numbers. In every pair, the first number is always smaller than the second number. A pair (c, d) can follow another pair (a, b) if b < c. Chain of pairs can be formed in this fashion. You have to find the longest chain which can be formed from the given set of pairs.

Example 1:

Input:

N = 5

P[] = {5 24 , 39 60 , 15 28 , 27 40 , 50 90}

Output: 3

Explanation: The given pairs are { {5, 24}, {39, 60},

{15, 28}, {27, 40}, {50, 90} },the longest chain that

can be formed is of length 3, and the chain is

{{5, 24}, {27, 40}, {50, 90}}

Example 2:

Input:

N = 2

P[] = {5 10 , 1 11}

Output: 1

Explanation:The max length chain possible is only of

length one.

Your Task:  
You don't need to read input or print anything. Your task is to Complete the function maxChainLen() which takes a structure p[] denoting the pairs and n as the number of pairs and returns the length of the longest chain formed.

Expected Time Complexity: O(Nlog(N))  
Expected Auxiliary Space: O(1)  
  
Constraints:  
1<=N<=105

# Problem 5: Same Problem 4 but using dynamic programming

# Problem 6: Backtracking

Consider a rat placed at (0, 0) in a square matrix of order N \* N. It has to reach the destination at (N - 1, N - 1). Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are 'U'(up), 'D'(down), 'L' (left), 'R' (right). Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it.  
Note: In a path, no cell can be visited more than one time. If the source cell is 0, the rat cannot move to any other cell.

Example 1:

Input:

N = 4

m[][] = {{1, 0, 0, 0},

{1, 1, 0, 1},

{1, 1, 0, 0},

{0, 1, 1, 1}}

Output:

DDRDRR DRDDRR

Explanation:

The rat can reach the destination at

(3, 3) from (0, 0) by two paths - DRDDRR

and DDRDRR, when printed in sorted order

we get DDRDRR DRDDRR.

Example 2:

Input:

N = 2

m[][] = {{1, 0},

{1, 0}}

Output:

-1

Explanation:

No path exists and destination cell is

blocked.

Your Task:   
You don't need to read input or print anything. Complete the function printPath() which takes N and 2D array m[ ][ ] as input parameters and returns the list of paths in lexicographically increasing order.   
Note: In case of no path, return an empty list. The driver will output "-1" automatically.

Expected Time Complexity: O((3N^2)).  
Expected Auxiliary Space: O(L \* X), L = length of the path, X = number of paths.

Constraints:  
2 ≤ N ≤ 5  
0 ≤ m[i][j] ≤ 1

# Problem 7: Backtracking Map coloring

Given an undirected graph and an integer M. The task is to determine if the graph can be colored with at most M colors such that no two adjacent vertices of the graph are colored with the same color. Here coloring of a graph means the assignment of colors to all vertices. Print 1 if it is possible to colour vertices and 0 otherwise.

Example 1:

Input:

N = 4

M = 3

E = 5

Edges[] = {(0,1),(1,2),(2,3),(3,0),(0,2)}

Output: 1

Explanation: It is possible to colour the

given graph using 3 colours.

Example 2:

Input:

N = 3

M = 2

E = 3

Edges[] = {(0,1),(1,2),(0,2)}

Output: 0

Your Task:  
Your task is to complete the function graphColoring() which takes the 2d-array graph[], the number of colours and the number of nodes as inputs and returns true if answer exists otherwise false. 1 is printed if the returned value is true, 0 otherwise. The printing is done by the driver's code.  
Note: In Example there are Edges not the graph.Graph will be like, if there is an edge between vertex X and vertex Y graph[] will contain 1 at graph[X-1][Y-1], else 0. In 2d-array graph[ ], nodes are 0-based indexed, i.e. from 0 to N-1.Function will be contain 2-D graph not the edges.  
  
Expected Time Complexity: O(MN).  
Expected Auxiliary Space: O(N).

Constraints:  
1 ≤ N ≤ 20  
1 ≤ E ≤ (N\*(N-1))/2  
1 ≤ M ≤ N

# Problem 8: Backtracking

Given a string S containing only digits, Your task is to complete the function genIp() which returns a vector containing all possible combinations of valid IPv4 IP addresses and takes only a string S as its only argument.  
Note: Order doesn't matter. A valid IP address must be in the form of A.B.C.D, where A, B, C, and D are numbers from 0-255. The numbers cannot be 0 prefixed unless they are 0.  
  
  
For string 11211 the IP address possible are   
1.1.2.11  
1.1.21.1  
1.12.1.1  
11.2.1.1

Example 1:

Input:

S = 1111

Output: 1.1.1.1

Example 2:

Input:

S = 55

Output: -1

Your Task:

Your task is to complete the function genIp() which returns a vector containing all possible combinations of valid IPv4 IP addresses or -1 if no such IP address could be generated through the input string S, the only argument to the function.

Expected Time Complexity: O(N \* N \* N)  
Expected Auxiliary Space: O(N \* N \* N \* N)

Constraints:  
1<=N<=16  
here, N = length of S.  
S only contains digits(i.e. 0-9)

# Problem 9: Backtracking

Given a dictionary of distinct words and an M x N board where every cell has one character. Find all possible words from the dictionary that can be formed by a sequence of adjacent characters on the board. We can move to any of 8 adjacent characters

Note: While forming a word we can move to any of the 8 adjacent cells. A cell can be used only once in one word.

Example 1:

Input:

N = 1

dictionary = {"CAT"}

R = 3, C = 3

board = {{C,A,P},{A,N,D},{T,I,E}}

Output:

CAT

Explanation:

C A P

A N D

T I E

Words we got is denoted using same color.

Example 2:

Input:

N = 4

dictionary = {"GEEKS","FOR","QUIZ","GO"}

R = 3, C = 3

board = {{G,I,Z},{U,E,K},{Q,S,E}}

Output:

GEEKS QUIZ

Explanation:

G I Z

U E K

Q S E

Words we got is denoted using same color.

Your task:  
You dont need to read input or print anything. Your task is to complete the function wordBoggle() which takes the dictionary contaning N space-separated strings and R\*C board as input parameters and returns a list of words that exist on the board in lexicographical order.

Expected Time Complexity: O(N\*W + R\*C^2)  
Expected Auxiliary Space: O(N\*W + R\*C)

Constraints:  
1 ≤ N ≤ 15  
1 ≤ R, C ≤ 50  
1 ≤ length of Word ≤ 60  
Each word can consist of both lowercase and uppercase letters.

# Problem 10: Divide and Conquer

Given two sorted arrays arr1 and arr2 of size N and M respectively and an element K. The task is to find the element that would be at the kth position of the final sorted array.

Example 1:

Input:

arr1[] = {2, 3, 6, 7, 9}

arr2[] = {1, 4, 8, 10}

k = 5

Output:

6

Explanation:

The final sorted array would be -

1, 2, 3, 4, 6, 7, 8, 9, 10

The 5th element of this array is 6.

Example 2:

Input:

arr1[] = {100, 112, 256, 349, 770}

arr2[] = {72, 86, 113, 119, 265, 445, 892}

k = 7

Output:

256

Explanation:

Final sorted array is - 72, 86, 100, 112,

113, 119, 256, 265, 349, 445, 770, 892

7th element of this array is 256.

Your Task:   
You don't need to read input or print anything. Your task is to complete the function kthElement() which takes the arrays arr1[], arr2[], its size N and M respectively and an integer K as inputs and returns the element at the Kth position.

Expected Time Complexity: O(Log(N) + Log(M))  
Expected Auxiliary Space: O(Log (N))

Constraints:  
1 <= N, M <= 106  
0 <= arr1i, arr2i < INT\_MAX  
1 <= K <= N+M

# Problem 11: Divide and Conquer

You have N books, each with A[i] number of pages. M students need to be allocated contiguous books, with each student getting at least one book.  
Out of all the permutations, the goal is to find the permutation where the sum of maximum number of pages in a book allotted to a student should be minimum, out of all possible permutations.

Note: Return -1 if a valid assignment is not possible, and allotment should be in contiguous order (see the explanation for better understanding).

Example 1:

Input:

N = 4

A[] = {12,34,67,90}

M = 2

Output:113

Explanation:Allocation can be done in

following ways:

{12} and {34, 67, 90} Maximum Pages = 191

{12, 34} and {67, 90} Maximum Pages = 157

{12, 34, 67} and {90} Maximum Pages =113.

Therefore, the minimum of these cases is 113,

which is selected as the output.

Example 2:

Input:

N = 3

A[] = {15,17,20}

M = 2

Output:32

Explanation: Allocation is done as

{15,17} and {20}

Your Task:  
You don't need to read input or print anything. Your task is to complete the function findPages() which takes 2 Integers N, and m and an array A[] of length N as input and returns the expected answer.

Expected Time Complexity: O(NlogN)  
Expected Auxilliary Space: O(1)

Constraints:  
1 <= N <= 105  
1 <= A [ i ] <= 106  
1 <= M <= 105

# Problem 12: Divide and Conquer

Given two integers m and n, try making a special sequence of numbers seq of length n such that

* seqi+1 >= 2\*seqi
* seqi > 0
* seqi <= m

Your task is to determine total number of such special sequences possible.

Example 1:

Input:

m = 10

n = 4

Output:

4

Explaination:

There should be n elements and

value of last element should be at-most m.

The sequences are {1, 2, 4, 8}, {1, 2, 4, 9},

{1, 2, 4, 10}, {1, 2, 5, 10}.

Example 2:

Input:

m = 5

n = 2

Output:

6

Explaination:

The sequences are {1, 2},

{1, 3}, {1, 4}, {1, 5}, {2, 4}, {2, 5}.

Your Task:  
You do not need to read input or print anything. Your task is to complete the function numberSequence() which takes the number m and n as input parameters and returns the number of possible special sequences.

Expected Time Complexity: O(m\*n)  
Expected Auxiliary Space: O(m\*n)

Constraints:  
1 ≤ m, n ≤ 100