Question: https://leetcode.com/problems/minimum-cost-to-merge-stones/

Intuition-  
So the problem is quite complicated, initially I tried bactracking where I added all possible k elements subarray and created a new array and recursively repeated this process untill the array was empty, but this naive approach had a time complexity of 2^n.

So I looked through discussion and youtube to understand the dp approach to this problem.

Youtube video link:https://www.youtube.com/watch?v=r83itwwSsEQ

Code:  
class Solution {

public int mergeStones(int[] stones, int k) {

int dp[][] = new int[stones.length][stones.length], n=stones.length, l=0;

if((n-1)%(k-1)!=0) return -1;

for(int z=0; z<n; z++){

int x=0, y=z;

while(x<n && y<n){

l=y-x;

int res=Integer.MAX\_VALUE;

if(l==0||l<k-1) res=0;

else if(l==k-1) res=sum(x,y,stones);

else if(l>=k-1){

for(int mid=x; mid<y; mid+=k-1){

res=Math.min(res,dp[x][mid]+dp[mid+1][y]);

}

if(l%(k-1)==0) res+=sum(x,y,stones);

}

dp[x][y]=res;

x++;

y++;

}

}

return dp[0][n-1];

}

public int sum(int x, int y, int[] stones){

int sum=0;

for(int i=x; i<=y; i++){

sum+=stones[i];

}

return sum;

}

}

Github Link :<https://lnkd.in/ecwtJeaz>