**AtCoder Dynamic Programming**

Table of Contents

[**Questions** 2](#_Toc157580963)

[A. Frog 1 2](#_Toc157580964)

[B. Frog 2 2](#_Toc157580965)

[**Solutions** 3](#_Toc157580966)

[A. Frog 1 3](#_Toc157580967)

[B. Frog 2 4](#_Toc157580968)

# **Questions**

## Frog 1

[Question](https://atcoder.jp/contests/dp/tasks/dp_a) , [Solution](#_Frog_1)

## Frog 2

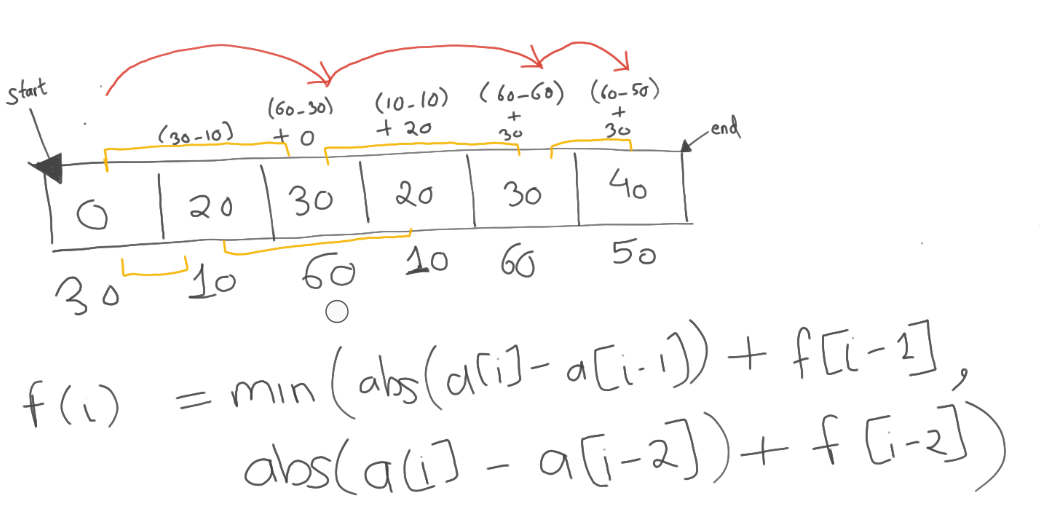
[Question](https://atcoder.jp/contests/dp/tasks/dp_b) , [Solution](#_Frog_1)

## Vacation

[Question](https://atcoder.jp/contests/dp/tasks/dp_c) , [Solution](#_Vacation)

# **Solutions**

## **Frog 1**



*# Bottom-up approach*

n = int(input())

a = list(map(int,input().split()))

dp = [0]\*n

dp[1] = abs(a[1]-a[0])

for i in range(2,n):

    dp[i] = min(dp[i-1]+abs(a[i]-a[i-1]) , dp[i-2]+abs(a[i]-a[i-2]))

print(dp[-1])

*# NOTE: AtCoder platform does not allow the use of recursive functions with deep recursion limits,*

*# which can lead to runtime errors such as a "Recursion Error: maximum recursion depth exceeded."*

*#Top-Down approach*

def frogTD(n,a,dp):

    if n==0:

        return 0

    elif n==1:

        return abs(a[1]-a[0])

    elif dp[n]!=-1:

        return dp[n]

    else:

        dp[n] =  min(abs(a[n]-a[n-1]) + frogTD(n-1,a,dp) , abs(a[n]-a[n-2]) + frogTD(n-2,a,dp))

        return dp[n]

n = int(input())

a = list(map(int,input().split()))

dp = [-1]\*(n)

print(frogTD(n-1,a,dp))

## **Frog 2**

*# Bottom-up approach (similar to Frog 1)*

n,k = map(int,input().split())

a = list(map(int,input().split()))

dp = [100000007]\*n

dp[0]=0

for i in range(1,n):

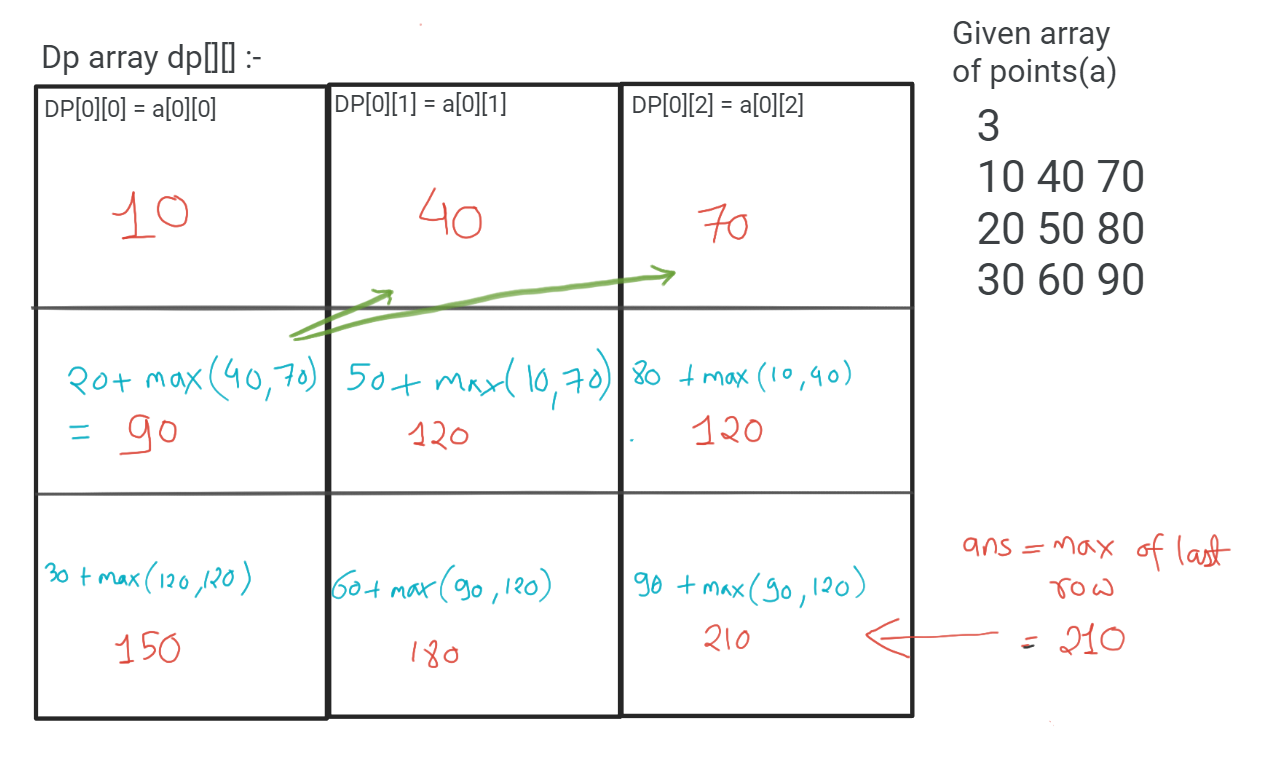
    for j in range(1,k+1):

        if i>=j:

            dp[i] = min(dp[i], dp[i-j]+abs(a[i]-a[i-j]))

print(dp[-1])

## **Vacation**



n = int(input())

a=[]

for \_ in range(n):

    arr = list(map(int,input().split()))

    a.append(arr)

dp=[[0,0,0] for \_ in range(n)]

dp[0]=a[0]

for i in range(1,n):

    dp[i][0] = a[i][0]+max(dp[i-1][1] , dp[i-1][2])

    dp[i][1] = a[i][1]+max(dp[i-1][0] , dp[i-1][2])

    dp[i][2] = a[i][2]+max(dp[i-1][0] , dp[i-1][1])

print(max(dp[-1]))

## **K. Stones**

If k==0 : any player who reaches this state will lose

If k<min(a): here also if any player reaches this state will lose

So winning or losing depends on state and independent of who plays. So if state=k is winning state, player one wins else player 2.

State K is winning state if any state ( k-a[i] ) is losing state for all a[i] in array a. Meaning if first player can push second player to any losing state then first player can win. But if all ( k-a[i] ) states are winning states, then kth state is losing state.

n , k = map(int,input().split())

a = list(map(int,input().split()))

dp = [-1]\*(k+1)

dp[0]=0 #at state 0 all will lose

#if a=[2,3] so at 2 and 3, player takes all stone and next player won't have any stone to pick.so that will be winning state.

for i in range(1,k+1):

    flag=0

    for j in a:

        if j<=i and dp[i-j]==0: #can send next player to any losing state

            flag=1

    dp[i]=flag

print("First" if dp[k]==1 else "Second")