## Frog Jump (GeeksForGeeks)

Ex:- If given 'HEIGHT' array is [10, 20, 30, 10]

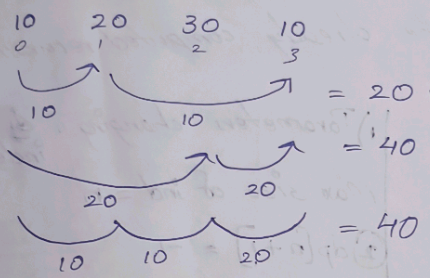
O/P = 20

Explanation: Frog can jump from 1st stair to 2nd stair = (|20-10|) = 10

Frog can jump from 2nd stair to last stair = (|10-20|) = 10

So Total energy lost = 20 (Minimum energy lost)

### Other Possible Ways




1) Represent indexes

10 20 30 10
0 1 2 3
(0 based index of height)

f(n-1){
// Min energy required to reach (n-1) from 0 based index
}

2) Do All Stuffs acc to the problem

1) If can jump 1 step & calculate energy lost ⇒

jump1 = f(ind-1) + abs(a[ind] - a[ind-1])

2) If can jump 2 steps & calculate energy lost ⇒

jump2 = f(ind-2) + abs(a[ind] - a[ind-2])

3) Return Minimum ⇒ return min(jump1, jump2)

```
f(ind){
if(ind==0) return 0;
jump1 = f(ind-1) + abs(a[ind] - a[ind-1]);
if(ind>1){
jump2 = f(ind-2) + abs(a[ind] - a[ind-2]);
}
return min(jump1, jump2);
```

Base Cases

```
if(ind==0) return 0;
```

2. If you are at 1 & go to -2 ⇒ -1 posi

```
if(ind>1){
// take jump2
}
```

# How to Convert Recursion into DP?

## Memoization

1) Look at the parameters changing)
2) Declare an array with max size of parameter & initialize it to -1
3) Use the array & store the already computed recursive values.

## FROG JUMP:

```
f(ind){
if(ind==0) return 0;
jump1 = f(ind-1) + abs(a[ind]-a[ind-1]);
if(ind>1){
jump2 = f(ind-2) + abs(a[ind]-a[ind-2]);
}
return min(jump1, jump2);
```

1) Parameters changing is index
Max size of ind = n
2) dp[n+1] = -1
1) Store & return.
return dp[ind] = min(jump1, jump2);
2) Whenever calling recursion check if it is previously computed or not
if(dp[ind] != -1) return dp[ind];

## Code for Memoization

```
f(ind){
if(ind==0) return 0;
if(dp[ind]!=-1) return dp[ind];
jump1 = f(ind-1) + abs(a[ind] - a[ind-1]);
if(ind>1){ jump2 = f(ind-2) + abs(a[ind]-a[ind-2]); }
return dp[ind] = min(jump1, jump2);
```

## Tabulation

In recursion you start from the top f(5) and go to the bottom hence it is a Top-Down Approach, Tabulation is the exact opposite i.e.., we calculate & Bottom-Up

### Steps

1) Initialize the dp with dp[n+1] = -1
2) Check for Base Cases & write in the form of dp

In Frog Jump if(ind == 0) return 0 ⟹ dp[0] = 0;

In Memorization/Recursion jump1, jump2 operations are being performed from f(5) ---> f(0) (since it is Top Down Approach)

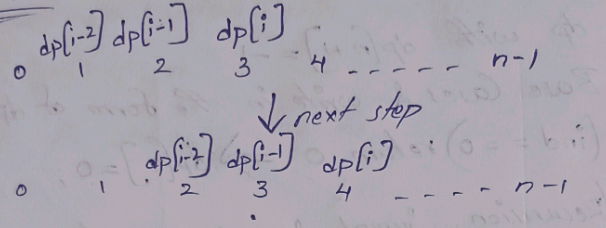
In Tabulation

```
for (i = 1 → n-1) {
    jump1 = dp[ind-1] + abs(a[ind] - a[ind-1]);
    if (i > 1) {
        jump2 = dp[ind-2] + abs(a[ind] - a[ind-2]);
    dp[ind] = min(jump1, jump2);
}
return dp[n-1];
```

## Space Optimization

wherever you see dp[ind-1], dp[ind-2], you can perform space optimization. In space optimization instead of using array you store the values in variables.

### FROG JUMP




At any position you only need dp[i-1] & dp[i-2] & you don't need anyone else. Therefore you store dp[i-1] & dp[i-2] in variables as prev1, prev2 instead of using array

As next step: you update prev1 = curr.
prev2 = prev1;

### Code

```
int dp[n] = -1;
int prev1 = 0, prev2 = 0;
for(i=1 → n-1){
 jump1 = prev1 + abs(a[i] - a[i-1]);
 if(i>1) {jump2 = prev2 + abs(a[i] - a[i-2]);
 int curr = min(jump1, jump2);
 prev2 = prev1;
 prev1 = curr;
}
return prev;
}
```