Stair Case

Send Feedback

A child runs up a staircase with 'n' steps and can hop either 1 step, 2 steps or 3 steps at a time. Implement a method to count and return all possible ways in which the child can run-up to the stairs.

Input format :

The first and the only line of input contains an integer value, 'n', denoting the total number of steps.

Output format :

Print the total possible number of ways.

Constraints :

0 <= n <= 10 ^ 4

Time Limit: 1 sec

Sample Input 1:

4

Sample Output 1:

7

Sample Input 2:

10

Sample Output 2:

274

```
public
class
Solution
{
                    public static long staircase(int n) {
                           //Your code goes here
                     /*
                     //If we reach n=0, we have found a path
                     if (n==0)
                         return 1;

                     //If n<0, the steps we took till then don't work
                     else if(n<0)
                         return 0;

                     return staircase(n-1)+staircase(n-2)+staircase(n-3);
                           */

                     if(n<=1)
                         return 1;
                     if(n==2)
                         return 2;

                     long dpCount[] = new long[n+1];
                     dpCount[0]=1;
                     dpCount[1]=1;
                     dpCount[2]=2;

                     for (int i=3;i<=n;i++)
                     {
                         dpCount[i]=dpCount[i-1]+dpCount[i-2]+dpCount[i-3];
```

```
        }
        return dpCount[n];
    }
}
```

Min Steps To One

Send Feedback

Given a positive integer 'n', find and return the minimum number of steps that 'n' has to take to get reduced to 1. You can perform any one of the following 3 steps:
1.) Subtract 1 from it. (n = n - 1) ,
2.) If its divisible by 2, divide by 2.(if n % 2 == 0, then n = n / 2) ,
3.) If its divisible by 3, divide by 3. (if n % 3 == 0, then n = n / 3).
Write brute-force recursive solution for this.

Input format :

The first and the only line of input contains an integer value, 'n'.

Output format :

Print the minimum number of steps.

Constraints :

1 <= n <= 200

Time Limit: 1 sec

Sample Input 1 :

4

Sample Output 1 :

2

Explanation of Sample Output 1 :

For n = 4
Step 1 : n = 4 / 2 = 2
Step 2 : n = 2 / 2 = 1

Sample Input 2 :

7

Sample Output 2 :

3

Explanation of Sample Output 2 :

For n = 7
Step 1 : n = 7 - 1 = 6
Step 2 : n = 6 / 3 = 2
Step 3 : n = 2 / 2 = 1

```
public
class
Solution
{

    public static int countMinStepsToOne(int n) {
        //Your code goes here
     if (n<=1)
         return 0;
     else if (n==2 || n==3)
         return 1;

    int output1=countMinStepsToOne(n-1);
    int output2=Integer.MAX_VALUE;
    int output3=Integer.MAX_VALUE;

    if (n%2==0)
```

```
            output2=countMinStepsToOne(n/2);

        if (n%3==0)
            output3=countMinStepsToOne(n/3);

        return Math.min(Math.min(output2,output3),output1)+1;
        }
    }
```

Min Steps To 1 Using DP

Send Feedback

Given a positive integer 'n', find and return the minimum number of steps that 'n' has to take to get reduced to 1. You can perform any one of the following 3 steps:
1.) Subtract 1 from it. (n = n - 1) ,
2.) If n is divisible by 2, divide by 2.(if n % 2 == 0, then n = n / 2) ,
3.) If n is divisible by 3, divide by 3. (if n % 3 == 0, then n = n / 3).

Input format :

The first and the only line of input contains an integer value, 'n'.

Output format :

Print the minimum number of steps.

Constraints :

1 <= n <= 10 ^ 6
Time Limit: 1 sec

Sample Input 1 :

4

Sample Output 1 :

2

Explanation of Sample Output 1 :

For n = 4
Step 1 : n = 4 / 2 = 2
Step 2 : n = 2 / 2 = 1

Sample Input 2 :

7

Sample Output 2 :

3

Explanation of Sample Output 2 :

For n = 7
Step 1 : n = 7 - 1 = 6
Step 2 : n = 6 / 3 = 2
Step 3 : n = 2 / 2 = 1

```
public
class
Solution
{
    public static int countMinStepsToOne(int n) {
        //Your code goes here
     if (n==0 || n==1)
         return 0;
     else if (n==2 || n==3)
         return 1;

     int[] dp = new int[n+1];
     for (int i=0;i<n+1;i++)
         dp[i]=-1;

     //Setting base cases
```

```
        dp[1]=0;
        dp[2]=1;
        dp[3]=1;
        for (int i=4;i<=n;i++)
        {
            //System.out.println("Current i: "+i);
            int ans1=dp[i-1];
            int ans2=Integer.MAX_VALUE,ans3=Integer.MAX_VALUE;
            if (i%2==0)
            {
                ans2=dp[i/2];
            }
            if (i%3==0)
            {
                ans3=dp[i/3];
            }
            //System.out.println("ans1: "+ans1+", ans2: "+ans2+", ans3: "+ans3);
            dp[i]=Math.min(ans1,Math.min(ans2,ans3))+1;
            //System.out.println(i+": "+dp[i]);
        }
        return dp[n];
       }
    }
```

Byte Landian

Send Feedback

Byteland has a very strange monetary system.

Each Bytelandian gold coin has an integer number written on it. A coin n can be exchanged in a bank into three coins: n/2, n/3 and n/4. But these numbers are all rounded down (the banks have to make a profit).

You can also sell Bytelandian coins for American dollars. The exchange rate is 1:1. But you can not buy Bytelandian coins.

You have one gold coin. What is the maximum amount of American dollars you can get for it?

Input format :

The first and the only line of input contains a the integer value of 'n'. It is the number written on your coin.

Output format :

Print the the maximum amount of American dollars you can make.

Constraints :

0 <= n <= 10 ^ 9

Time Limit: 1 sec

Sample Input 1 :

12

Sample Output 1 :

13

Explanation of Sample Output 1 :

You can change 12 into 6, 4 and 3, and then change these into $6 + $4 + $3 = $13.

Sample Input 2 :

2

Sample Output 1 :

2

Explanation of Sample Output 2 :

If you try changing the coin 2 into 3 smaller coins, you will get 1, 0 and 0, and later you can get no more than $1 out of them. It is better just to change the 2 coin directly into $2.

```
import
java.util.HashMap;
                    public class Solution {

                            public static long bytelandian(long n, HashMap<Long, Long> memo) {
                             // Write your code here
                             //Handle base case for n=0,1
                             if (n<12)
                             {
                                 memo.put(n,n);
                                 return n;
                             }

                             if (!memo.containsKey(n))
                             {
                                 long ans1=bytelandian(n/2,memo);
                                   long ans2=bytelandian(n/3,memo);
                                   long ans3=bytelandian(n/4,memo);

                                   long currVal=ans1+ans2+ans3;
                                   if (currVal>n)
                                   memo.put(n,currVal);
                                   else
                                   memo.put(n,n);

                             }

                             return memo.get(n);

                             /*
                             //Recursive solution

                             memo.put((long)0,(long)0);
                             memo.put((long)1,(long)1);

                             for (int i=2;i<=n;i++)
                             {
                                 long ans1=memo.get(i/2);
                                 long ans2=memo.get(i/3);
                                 long ans3=memo.get(i/4);

                                 long currentVal=ans1+ans2+ans3;
                                 if (currentVal>i)
                                 {
                                     memo.put((long)i,currentVal);
                                 }
                                 else
                                 {
                                     memo.put((long)i,(long)i);
```

```
                }
            }

            return memo.get(n);
            */
            }
        }
```

Loot Houses

Send Feedback

A thief wants to loot houses. He knows the amount of money in each house. He cannot loot two consecutive houses. Find the maximum amount of money he can loot.

Input format :

The first line of input contains an integer value of 'n'. It is the total number of houses.

The second line of input contains 'n' integer values separated by a single space denoting the amount of money each house has.

Output format :

Print the the maximum money that can be looted.

Constraints :

0 <= n <= 10 ^ 4

Time Limit: 1 sec

Sample Input 1 :

6
5 5 10 100 10 5

Sample Output 1 :

110

Sample Input 2 :

6
10 2 30 20 3 50

Sample Output 2 :

90

Explanation of Sample Output 2 :

Looting first, third, and the last houses([10 + 30 + 50]) will result in the maximum loot, and all the other possible combinations would result in less than 90.

```
public
class
Solution
{
    public static int maxMoneyLooted(int[] houses) {
        //Your code goes here
     //Handling the base cases where number of houses = 0,1,2
     if (houses.length==0)
         return 0;

     if (houses.length==1)
         return houses[0];

     if (houses.length==2)
         return Math.max(houses[0],houses[1]);

     int n = houses.length;
     int[] dp = new int[n];
```

```
        dp[0]=houses[0];
        dp[1]=Math.max(houses[0],houses[1]);

        for (int i=2;i<n;i++)
        {
            //For every house, we consider two cases
            //Case 1 - Current house is counted as part of the max value => This
means the previous house cannot be counted
            int maxVal1=dp[i-2]+houses[i];

            //Case 2 - Current house is not counted as part of the max value =>
This means previous house can be counted
            int maxVal2=dp[i-1];

            //Max value till current house is maximum of the two possible max
values till now
            dp[i]=Math.max(maxVal1,maxVal2);
        }

        //Final element of dp stores max possible value for given number of houses
and their respective amounts of loot
        return dp[n-1];

       }
}
```

Coin Tower

Send Feedback

Whis and Beerus are playing a new game today. They form a tower of N coins and make a move in alternate turns. Beerus plays first. In one step, the player can remove either 1, X, or Y coins from the tower. The person to make the last move wins the game. Can you find out who wins the game?

Input format :

The first and the only line of input contains three integer values separated by a single space. They denote the value of N, X and Y, respectively.

Output format :

Prints the name of the winner, either 'Whis' or 'Beerus' (Without the quotes).

Constraints :

1 <= N <= 10 ^ 6
2 <= X <= Y<= 50

Time Limit: 1 sec

Sample Input 1 :

4 2 3

Sample Output 1 :

Whis

Sample Input 2 :

10 2 4

Sample Output 2 :

Beerus

Explanation to Sample Input 1:

Initially, there are 4 coins, In the first move, Beerus can remove either 1, 2, or 3 coins in all three cases, Whis can win by removing all the remaining coins.

```
public
class
Solution
```

```
{
        public static String findWinner(int n, int x, int y) {
            //Your code goes here
         int[] dp = new int[n+1];
         // 0 - Whis wins ; 1 - Beerus wins
         dp[0]=0;
         dp[1]=1;

         for (int i=2;i<=n;i++)
         {
             //Beerus wins if dp[i-1] or dp[i-x] or dp[i-y] is 0, i.e, Whis wins
when the number of coins is (i-1), (i-x) or (i-y)
             //If none of them are 0, then Beerus cannot win => This means Whis wins
for i number of coins
             if (dp[i-1]==0)
             {
                 dp[i]=1;
             }
             else if ((i-x)>=0 && dp[i-x]==0)
             {
                 dp[i]=1;
             }
             else if ((i-y)>=0 && dp[i-y]==0)
             {
                 dp[i]=1;
             }
             else
             {
                 dp[i]=0;
             }
             /*
             if (dp[i]==1)
               System.out.println("For n="+i+" coins, Winner is: Beerus");
               else
               System.out.println("For n="+i+" coins, Winner is: Whis");
             */
         }

         if (dp[n]==1)
             return "Beerus";
         else
             return "Whis";
        }
}
```