Error in Competitive programming Online Code

Runtime error:

Segmentation fault:
 Override memory and running to fill more memory.

output limit exceeded:
Your program has printed too much data to output.

floating point error:
This usually occurs when you're trying to divide a number by 0, or trying to take the square root of a negative number.

NZEC
(non-zero exit code)

1. Trying to allocate too much memory during code execution may also be one of the reasons.
2. It could happen if your program threw an exception which was not caught.
3. this message means that the program exited returning a value different from 0 to the shell.

##

Geeks MMI (Multiplicative onverse): do by making table finding q and r and the t=t1-qt2

z = (x/y) % M;
instead we should perform
y_inv = findMMI(y, M);
z = (x * y_inv) % M;

##

**BEST LINK FOR ALL BITWISE OPERATION (LEETCODE)

https://leetcode.com/problems/sum-of-two-integers/discuss/84278/A-summary:-how-to-use-bit-manipulation-to-solve-problems-easily-and-efficiently

@@BIt Maniputation@@https://www.hackerearth.com/practice/basic-programming/bit-manipulation/basics-of-bit-manipulation/tutorial/

SUM WITHOUT + operator

```
#include<stdio.h>

int Add(int x, int y)
{
    // Iterate till there is no carry
    while (y != 0)
    {
        // carry now contains common set bits of x and y
        int carry = x & y;

        // Sum of bits of x and y where at least one of the bits is not set
        x = x ^ y;

        // Carry is shifted by one so that adding it to x gives the required sum
        y = carry << 1;
    }
    return x;
}

int main()
{
    printf("%d", Add(15, 32));
    return 0;
}
```

x^y -> it gives common digits.

x&y -> carry is the common set of bits and it is suited to left by 1 so that when we take xor it will give exact sum.

Reverse actual bits of the given number

Input : 11
Output : 13
(11)10 = (1011)2.
After reversing the bits we get:
(1101)2 = (13)10.

```
// C++ implementation to reverse bits of a number
#include <bits/stdc++.h>

using namespace std;

// function to reverse bits of a number
unsigned int reverseBits(unsigned int n)
{
    unsigned int rev = 0;

    // traversing bits of 'n' from the right
    while (n > 0)
    {
        // bitwise left shift
        // 'rev' by 1
        rev <<= 1;

        // if current bit is '1'
        if (n & 1 == 1)
            rev ^= 1;

        // bitwise right shift
        // 'n' by 1
        n >>= 1;

    }

    // required number
    return rev;
}

// Driver program to test above
int main()
{
    unsigned int n = 11;
    cout << reverseBits(n);
    return 0;
}
```

1) How to check if a given number is a power of 2 ?

The same problem can be solved using bit manipulation. Consider a number x that we need to check for being a power for 2. Now think about the binary representation of (x-1). (x-1) will have all the bits same as x, except for the rightmost 1 in x and all the bits to the right of the rightmost 1.

Let, x = 4 = $(100)_2$

x - 1 = 3 = $(011)_2$

Let, $x = 6 = (110)_2$

$x - 1 = 5 = (101)_2$

It might not seem obvious with these examples, but binary representation of (x-1) can be obtained by simply flipping all the bits to the right of rightmost 1 in x and also including the rightmost 1.

Now think about x & (x-1). x & (x-1) will have all the bits equal to the x except for the rightmost 1 in x.

Let, $x = 4 = (100)_2$

$x - 1 = 3 = (011)_2$

$x \& (x-1) = 4 \& 3 = (100)_2 \& (011)_2 = (000)_2$

Let, $x = 6 = (110)_2$

$x - 1 = 5 = (101)_2$

$x \& (x-1) = 6 \& 5 = (110)_2 \& (101)_2 = (100)_2$

Properties for numbers which are powers of 2, is that they have one and only one bit set in their binary representation. If the number is neither zero nor a power of two, it will have 1 in more than one place. **So if x is a power of 2 then x & (x-1) will be 0.**

Implementation: (POWER OF 2)

```
 bool isPowerOfTwo(int x)
   {
      // x will check if x == 0 and !(x & (x - 1)) will check if x is a power of 2 or
not
      return (x && !(x & (x - 1)));
   }
```

Implementation: (POWER OF 4) (***IMP*****)**

```
public class Solution {
   public boolean isPowerOfFour(int num) {
      return(num > 0 && ((num & num-1) == 0) && ((Math.log(num)/
Math.log(2)) %2 == 0));
   }
}
```

$$$

Count the number of ones in the binary representation of the given number.

The basic approach to evaluate the binary form of a number is to traverse on it and count the number of ones. But this approach takes $\log_2 N$ of time in every case.

Why $\log_2 N$?

As to get a number in its binary form, we have to divide it by 2, until it gets 0,

which will take $\log_2 N$ of time.
With bitwise operations, we can use an algorithm whose running time depends on the number of ones present in the binary form of the given number. This algorithm is much better, as it will reach to logN, only in its worst case.

```
int count_one (int n)
    {
        while( n )
        {
        n = n&(n-1);
           count++;
        }
        return count;
}
```

$$

Pascal's triangle:

(i-1)C(j-1)
$$

EXAMPLE

$$\frac{1}{2}+\frac{2}{4}+\frac{3}{8}+\frac{4}{16}+\frac{5}{32}+\cdots=?$$

Let us assume the given series to be S, then

$$S=\frac{1}{2}+\frac{2}{4}+\frac{3}{8}+\frac{4}{16}+\frac{5}{32}+\cdots.$$

On multiplying S by $\frac{1}{2}$, we get

$$\frac{S}{2}=\frac{1}{4}+\frac{2}{8}+\frac{3}{16}+\frac{4}{32}+\frac{5}{64}+\cdots.$$

Now subtracting $\frac{S}{2}$ from S, we get

$$\begin{aligned} S &= \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \frac{5}{32} + \cdots \\ \frac{S}{2} &= 0 + \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \frac{4}{32} + \frac{5}{64} + \cdots \\ \hline S\left(1-\frac{1}{2}\right) &= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots \\ \Rightarrow \frac{S}{2} &= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots, \end{aligned}$$

which is a GP. Therefore, using the formula for the sum of infinite terms of GP, we get

$$S=2\times\left(\frac{\frac{1}{2}}{1-\frac{1}{2}}\right)=2. \square$$

We are now ready to state the sum of an infinite AGP, and will present the proof below:

THEOREM

The sum of infinite terms of an AGP is given by $S_\infty=\frac{a}{1-r}+\frac{dr}{(1-r)^2}$, where $|r|<1$. $\square$

$$

Recursion is more time consuming and space consuming that iteration

$$

NOTE:

calloc() zero-initializes the buffer, while malloc() leaves the memory uninitialized.

$$$$$$$$$$$$$$$$$$$$$. VERY IMPORTANT $$$$$$$$$$$$$$$$$ $$$$

https://leetcode.com/problems/single-number-ii/description/

Read Explanation:
https://leetcode.com/problems/single-number-ii/discuss/43295/Detailed-explanation-and-generalization-of-the-bitwise-operation-method-for-single-numbers

$$$$$$$$$$$$$$$$$$$$$. VERY IMPORTANT $$$$$$$$$$$$$$$$$ $$$$

NUMBER OF ONES
https://leetcode.com/problems/number-of-digit-one/

For example ‘8192’:
1-999 -> countDigitOne(999)
1000-1999 -> 1000 of 1s + countDigitOne(999)
2000-2999 -> countDigitOne(999)
.
.
7000-7999 -> countDigitOne(999)
8000-8192 -> countDigitOne(192)
Count of 1s : _*countDigitOne(999)8 + 1000 + countDigitOne(192)_
Noticed that, if the target is ‘1192’:
Count of 1s : _*countDigitOne(999)1 + (1192 - 1000 + 1) + countDigitOne(192)_
(1192 - 1000 + 1) is the 1s in thousands from 1000 to 1192.
Same codes as above, maybe much easier to understand.

/*
233. Number of Digit One

Given an integer n, count the total number of digit 1 appearing in all non-negative integers less than or equal to n.

For example:
Given n = 13,
Return 6, because digit 1 occurred in the following numbers: 1, 10, 11, 12, 13.

```
*/
class Solution {
public:
    int lengthDigit(int n){
        int c=0;
        while(n>0){
            c++;
            n=n/10;
        }
        return c;
    }
    int countDigitOne(int n) {
        if(n<=0){
            return 0;
        }else if(n<10){
            return 1;
        }
        int length=lengthDigit(n);
        int base=pow(10,length-1);
        int answer=n/base;
        int remainder=n%base;
        int m;
        if(answer==1){
            m=n-base+1;
        }else{
            m=base;
        }
        return countDigitOne(base-1)*answer+m+countDigitOne(remainder);
    }
};
```