



Yesterday I showed you all code for solution. After I pulled it op and started talking, I saw that this was NOT my code but another solution I had found to compare to my code. I wanted to make sure I dodn’t take credit for this solution since it had a neat way of doing 2n. Below is my analysis and a comparison of my solution and the other solution.

**Analysis:**

If you consider each snapper as one bit in a bit string and a “1” as being output on, then the action of connected snappers is simply adding one progressively to the bitstring. The “1” does NOT mean that there is power at the output. Switch is thrown but no power to switch.

i.e.

3 snappers

0 0 0 - All off

1 - 0 0 1 - snapper 1 turns on provides power to snapper 2

2 - 0 1 0 - snapper 2 turns on but 1 goes off so S2 has no power

3 - 0 1 1 - snapper 1 goes on so NOW S2 has power at output which goes to S3

4 - 1 0 0 - S3 turns on but S1 and S2 go off so no power at S3 output

5 - 1 0 1

6 - 1 1 0

7 - 1 1 1 - Al snapper on so S3 finally has power at output

8 - 0 0 0 - All turn off and cycle starts again

9 - 0 0 1 - snapper 1 turns on provides power to snapper 2

10 - 0 1 0 - snapper 2 turns on but 1 goes off so S2 has no power

11 - 0 1 1 - snapper 1 goes on so NOW S2 has power at output which goes to S3

12 - 1 0 0 - S3 turns on but S1 and S2 go off so no power at S3 output

13 - 1 0 1

14 - 1 1 0

15 - 1 1 1 - All snapper on so S3 finally has power at output

For the first cycle, the final snapper has power at snap 2n – 1 so this means that if we are calculating the number of snaps, k, k= 2n – 1. So if k – (2n – 1) = 0 then final snapper has power. Simplifying, if k-2n+1 = 0 then we have power.

For snaps greater than 2n +1, the next all on is at 2n +1 + 2n, so in general if (k-2n+1) % 2n = 0, then last snapper will have power at outlet

My solution:

int twon = pow(2,n);

if((k-twon+1) % twon == 0 || (k-twon+1) == 0) {

cout << "Case #" << i << ": ON\n";

}else {

cout << "Case #" << i << ": OFF\n";

}

Using bit shift to get 2n:

cin >> n >> k;  
ll d = 1LL << n;  
if (k - (d - 2) - 1 == 0 || (k - (d - 2) - 1) % d == 0) {  
 cout << "Case #" << i << ": ON\n";  
}  
else {  
 cout << "Case #" << i << ": OFF\n";  
}

Your Solution:

std::cin >> numSnappers;  
std::cin >> timesSnapped;  
auto start = std::chrono::steady\_clock::now();  
bool flag = true;  
for (int j = 0; j < numSnappers; j++) {  
 if (!(timesSnapped&(1<<j))) {  
 flag = false;  
 break;  
 }  
}

**Timing Analysis: 4 snappers 63 snaps**

Best Code as is: ~~ 6500 ns

Remove ll and LL: about same

Remove extra includes: ~~ 6400 ns

My code using pow(): ~~ 6400 // This really shocked me

Your code: ~~ 6400 // this is interesting

The point here is that pow() looks to be optimized.