Power of Three

Given an integer, write a function to determine if it is a power of three.

Follow up:
Could you do it without using any loop / recursion?

Credits:
Special thanks to @dietpepsi for adding this problem and creating all test cases.

Solution 1

Well, this problem doesn't seem to be quite interesting or worthwhile to think about at a first glance. I had the same feeling at the beginning. However, after seeing a couple of posts, I saw a couple of interesting ways. So here is a summary post and hope you learn something from others' solutions.

Two trivial solutions first:

Recursive Solution

```
public boolean isPowerOfThree(int n) {
    return n>0 && (n==1 || (n%3==0 && isPowerOfThree(n/3)));
}
```

Iterative Solution

update following Stefan's answer below:

```
public boolean isPowerOfThree(int n) {
    if(n>1)
        while(n%3==0) n /= 3;
    return n==1;
}
```

my original code: public boolean isPowerOfThree(int n) { while(n>1) { if(n%3!=0) return false; n /= 3; } return n<=0 ? false : true; }

It's all about MATH...

Method 1

Find the maximum integer that is a power of 3 and check if it is a multiple of the given input. (related post)

```
public boolean isPowerOfThree(int n) {
    int maxPowerOfThree = (int)Math.pow(3, (int)(Math.log(0x7fffffff) / Math.log(
3)));
    return n>0 && maxPowerOfThree%n==0;
}
```

Or simply hard code it since we know `maxPowerOfThree = 1162261467`:

```
public boolean isPowerOfThree(int n) {
    return n > 0 && (1162261467 % n == 0);
}
```

It is worthwhile to mention that Method 1 works only when the base is prime. For example, we cannot use this algorithm to check if a number is a power of 4 or 6 or any other composite number.

Method 2

If `log10(n) / log10(3)` returns an int (more precisely, a double but has 0 after decimal point), then n is a power of 3. (original post). But **be careful here**, you cannot use `log` (natural log) here, because it will generate round off error for `n=243`. This is more like a coincidence. I mean when `n=243`, we have the following results:

```
log(243) = 5.493061443340548    log(3) = 1.0986122886681098
   ==> log(243)/log(3) = 4.999999999999999

log10(243) = 2.385606273598312    log10(3) = 0.47712125471966244
   ==> log10(243)/log10(3) = 5.0
```

This happens because `log(3)` is actually slightly larger than its true value due to round off, which makes the ratio smaller.

```
public boolean isPowerOfThree(int n) {
    return (Math.log10(n) / Math.log10(3)) % 1 == 0;
}
```

Method 3 related post

```
public boolean isPowerOfThree(int n) {
    return n==0 ? false : n==Math.pow(3, Math.round(Math.log(n) / Math.log(3)));
}
```

Method 4 related post

```
public boolean isPowerOfThree(int n) {
    return n>0 && Math.abs(Math.log10(n)/Math.log10(3)-Math.ceil(Math.log10(n)/Ma
th.log10(3))) < Double.MIN_VALUE;
}
```

`Cheating` Method

This is not really a good idea in general. But for such kind of `power` questions, if we need to check many times, it might be a good idea to store the desired powers into an array first. (related post)

```
public boolean isPowerOfThree(int n) {
    int[] allPowerOfThree = new int[]{1, 3, 9, 27, 81, 243, 729, 2187, 6561, 1968
3, 59049, 177147, 531441, 1594323, 4782969, 14348907, 43046721, 129140163, 387420
489, 1162261467};
    return Arrays.binarySearch(allPowerOfThree, n) >= 0;
}
```

or even better with HashSet:

```
public boolean isPowerOfThree(int n) {
    HashSet<Integer> set = new HashSet<>(Arrays.asList(1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683, 59049, 177147, 531441, 1594323, 4782969, 14348907, 43046721, 129140163, 387420489, 1162261467));
    return set.contains(n);
}
```

New Method Included at 15:30pm Jan-8th

Radix-3 original post

The idea is to convert the original number into radix-3 format and check if it is of format `10*` where `0*` means `k` zeros with `k>=0`.

```
public boolean isPowerOfThree(int n) {
    return Integer.toString(n, 3).matches("10*");
}
```

Any other interesting solutions?

written by GWTW original link here

Solution 2

```
public class Solution {
public bool IsPowerOfThree(int n) {
     return n > 0 && (1162261467 % n == 0);
  }
}
```

written by andrei3 original link here

Solution 3

If `N` is a power of `3`:

- It follows that `3^X == N`
- It follows that `log (3^X) == log N`
- It follows that `X log 3 == log N`
- It follows that `X == (log N) / (log 3)`
- For the basis to hold, `X` must be an integer.

However, due to precision issues that arise from the fact that log 3 cannot be precisely represented on a binary computer; X is considered to be an integer if it's decimal component falls within a guard range of `+/-0.00000000000001`.

```
public boolean isPowerOfThree(int n) {
  double a = Math.log(n) / Math.log(3);
  return Math.abs(a - Math.rint(a)) <= 0.00000000000001;
}
```

written by oluwasayo original link here