1780. Check if Number is a Sum of Powers of Three

Given an integer `n`, return `true` *if it is possible to represent* `n` *as the sum of distinct powers of three.* Otherwise, return `false`.

An integer `y` is a power of three if there exists an integer `x` such that `y == 3x`.

Example 1:

```
Input: n = 12
Output: true
Explanation: 12 = 31 + 32
```

Example 2:

```
Input: n = 91
Output: true
Explanation: 91 = 30 + 32 + 34
```

Example 3:

```
Input: n = 21
Output: false
```

Constraints:

- $1 \le n \le 10^7$

1780. Check if Number is a Sum of Powers of Three

Overview

We are given an integer `n` and need to determine if it can be written as a sum of **distinct** powers of 3. In other words, we want to know if we can choose some of the numbers $3^0, 3^1, 3^2, \ldots$, each used at most once, such that their sum equals n. A generalized mathematical way to express this is:

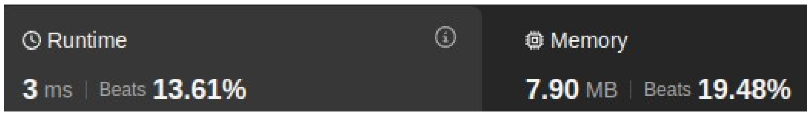
$$n = 3^{x_1} + 3^{x_2} + \cdots + 3^{x_k}$$

where all exponents $x_1, x_2, \ldots, x_k$ are unique and non-negative.

We need to return $true$ if such a sum exists, otherwise $false$.

1780. Check if Number is a Sum of Powers of Three

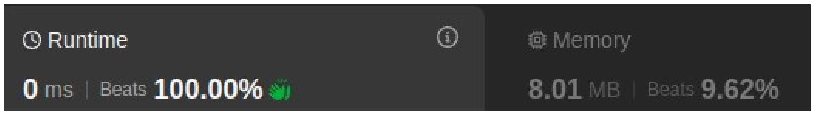
```
/*
    Brute force: Recursion: Include/Exclude technique
    Time complexity: $O(\log_3(n).2^{\log_3(n)})$
    Space complexity: $O(\log_3(n))$
*/
class Solution {
    public:
        // Exponential power
        // Time complexity: O(log b)
        // Space complexity: O(1)
        int power(int a,int b){
            int res=1;
            while (b>0) {
                if (b&1) res*=a;
                a*=a;
                b/=2;
            }
            return res;
        }
```


Runtime
3 ms | Beats 13.61%
Memory
7.90 MB | Beats 19.48%

```
 bool checkPowersOfThree(int n){
     // Determine de maximum power of $3$ of $n$
     int max_level=log(n)/log(3);

     // Recursive function
     // The Include/Exclude technique ensure the exploration all distinct  possible powers of three
     auto solve=[&](int level,int sum,auto& self)->bool{
       // If we reach the maximum level
       if(level<0){
         if(sum==n) return true; // If the sum is equal to $n$ ,return false
         return false; // Otherwise, return false
       }

       // If we reach the sum, before reaching the level
       if(sum>n) return false; // If sum is greater than $n$
       if(sum==n) return true; // If the sum is equal to $n$

       // Include $3^{level}$ to the sum, than pass the next level || Exclude $3^{level}$ from the sum,
       // than pass the next level
       return self(level-1,power(3,level)+sum,self) || self(level-1,sum,self);
     };

     return solve(max_level,0,solve);
   }
};
```

1780. Check if Number is a Sum of Powers of Three

```
/*
   Math
   Time complexity: $O(\log_3(n).\log(\log_3(n)))$
   Space complexity: O(1)
*/
```


Runtime
0 ms | Beats 100.00%
Memory
8.01 MB | Beats 9.62%

```
class Solution {
public:
   // Exponential power, Time complexity: O(log b, Space complexity: O(1)
    int power(int a,int b) {
      int res=1;
      while (b>0) {
         if (b&1) res*=a;
         a*=a;
         b/=2;
      }
      return res;
   }
   bool checkPowersOfThree(int n){
      // Determine de maximum power of 3 of n
      int x=log(n)/log(3);

      int p=0;
      while(n>0){
         // $3^x$
         p=power(3,x);

         // If $n$ still greater than $3^x$ , reduce $3^x$ from $n$
         if(n>=p) n-=p;

         // If $n$ is still greater than $3^x$ , means than $x$ will be used twice in the current sum
         // so return false
         if(n>=p) return false;

         // Pass to the next power
         x--;
      }
      // means n==0 (return)
      return true;
   }
};
```

1780. Check if Number is a Sum of Powers of Three

```
/*
    Ternary representation
    Time complexity: O(log_3(n))
    Space complexity: O(1)
*/
class Solution {
public:
    /*
        In base 3, each digit in a number represents the number of copies of that power of 3.
        That is, the first digit tells you how many ones you have;
        the second tells you how many 3s you have;
        the third tells you how many 3x3 you have;
        the fourth tells you how many 3x3x3 you have;
        and so on.
    */
    bool checkPowersOfThree(int n){
        while(n>0){
            // If remainder=k, means that we have k powers of three
            // if remainder>=2, means that we have a repeated power of 3
            if(n%3==2) return false;

            n/=3;
        }
        return true;
    }
};
```