264. Ugly Number II

An **ugly number** is a positive integer whose prime factors are limited to `2`, `3`, and `5`.

Given an integer `n`, return *the* `n`th ***ugly number***.

Example 1:

```
Input: n = 10
Output: 12
Explanation: [1, 2, 3, 4, 5, 6, 8, 9, 10, 12] is the sequence of the first 10 ugly numbers.
```

Example 2:

```
Input: n = 1
Output: 1
Explanation: 1 has no prime factors, therefore all of its prime factors are limited to 2, 3, and 5.
```

Constraints:

- `1 <= n <= 1690`

264. Ugly Number II

```
/*
    Naive
    Time complexity: O(n^2) - TLE
    Space complexity: O(1)
*/
class Solution {
public:
    bool are_prime_factors_2_3_5(int m){
        int d=2;
        while(m!=1){
            if(m%d==0){
                if(d!=2&&d!=3&&d!=5) return false;
                m/=d;
            }
            else d++;
        }
        return true;
    }
    int nthUglyNumber(int n){
        int cnt=0,m=0;
        while(cnt<n){
            m++;
            if(are_prime_factors_2_3_5(m)) cnt++;
        }
        return m;
    }
};
```

264. Ugly Number II

```
/*
    Dijkstra Hamming problem: Min heap
    Time complexity: O(n*6logn)=O(nlogn) - AC
    Space complexity: O(3n+3n)=O(n)
*/
class Solution {
    public:
        int nthUglyNumber(int n){

            std::priority_queue<long long,
                                std::vector<long long>,
                                std::greater<long long>> min_heap;
            min_heap.push(1);

            std::set<long long> visited;

            for (int i=0;i<n;i++) {
                int x=min_heap.top();
                min_heap.pop();

                if(i==n-1) return (int)x;

                if(visited.find((long long)x*2)==visited.end()){
                    min_heap.push((long long)x*2);
                    visited.insert((long long)x*2);
                }
                if(visited.find((long long)x*3)==visited.end()){
                    min_heap.push((long long)x*3);
                    visited.insert((long long)x*3);
                }
                if(visited.find((long long)x*5)==visited.end()){
                    min_heap.push((long long)x*5);
                    visited.insert((long long)x*5);
                }
            }

            return -1; // Never reached
        }
};
```

264. Ugly Number II

```
/*
    Dijkstra Hamming problem: Array
    Time complexity: O(n) - AC
    Space complexity: O(n)
*/
class Solution {
public:

    int nthUglyNumber(int n){
        std::vector<int>h(n);
        h[0]=1;

        int x2=2, x3=3, x5=5;

        int i2=0, i3=0, i5=0;

        for (int i=1;i<n;i++) {
            h[i]=std::min({x2,x3,x5});
            if (h[i]==x2) x2=2*h[++i2];
            if (h[i]==x3) x3=3*h[++i3];
            if (h[i]==x5) x5=5*h[++i5];
        }
        return h[n-1];
    }
};
```