# 题目

编写一段程序来查找第n个超级丑数。

超级丑数是指其所有质因数都是长度为 k 的质数列表 primes 中的正整数。

**示例:**

**输入:** n = 12, primes = [2,7,13,19]

**输出:** 32

**解释:** 给定长度为4的质数列表primes = [2,7,13,19]，前12个超级丑数序列为：[1,2,4,7,8,13,14,16,19,26,28,32] 。

**说明:**

1 是任何给定 primes 的超级丑数。

 给定 primes 中的数字以升序排列。

0 < k≤100, 0 < n≤106, 0 < primes[i] < 1000。

第 n 个超级丑数确保在32位有符整数范围内。

**丑数相关题目：**

面试题49. 丑数

263. 丑数

264. 丑数 II

313. 超级丑数

1201. 丑数 III

# 分析

## 方法一：堆/优先队列

**思路：**

一套模板：优先级队列搞定所有丑数

大顶堆：每个结点的值都大于或等于其左右结点的值

小顶堆：每个结点的值都小于或等于其左右结点的值

无法直接获得整个数组元素

1）一般这种问题都是可以通过获取所有序列后，通过排序获得的；

2）在部分场景下，无法获取所有序列，或者获取序列的时间成本和空间成本过大。但是数据获取的顺序是有规律的，因此可以使用大顶堆或者小顶堆；

质数因子求所有序列中第k个元素。因为从1开始与所有因子相乘，压入栈内做BFS可以获得所有数字序列，但是

1） 无法穷举

2） 没有顺序

因此用普通的排序无法获得第K个小的元素，但是有一个规律：

基于贪心，最小的元素\*最小的因子 = 本轮最小的元素。

Min（本轮最小的元素，已经生成序列最小的元素） = 当前最小的元素

即，需要弹出进行下一轮运算的元素。

按题目要求：

1）队列的元素来源：堆顶元素\*质数列表，将新生成的元素放入队列，弹出的第n个元素就是我们求的超级丑数。但是，弹出的元素不一定是第n个最小的质数，因此我们弹出前需要排序。

2）第n个质数，则要求队列是有序的，且是从小到大排列的，因此我们取小顶堆

**丑数模板：**

1）使用优先级队列

2）题目要求超级丑数在32位有符整数范围内，因此当乘积大于INT32\_MAX时，不需要压入队列

3）重复元素不能加入队列（使用set数据结构去重）

**代码：**

class Solution {

public:

int nthSuperUglyNumber(int n, vector<int> &primes)

{

priority\_queue<long long, vector<long long>, greater<long long>> buff;

unordered\_set<long long> primeset;

buff.push(1);

primeset.insert(1);

long long i = 1;

int count = 0;

while (count < n) {

count++;

i = buff.top();

buff.pop();

for (long long prime : primes) {

long long next = i \* prime;

if (next<=INT32\_MAX && !primeset.count(next)) {

buff.push(next);

primeset.insert(next);

}

}

}

return i;

}

};

## 方法二：动态规划

代码：

class Solution {

public:

int nthSuperUglyNumber(int n, vector<int>& primes) {

vector<long> dp(n + 1);

int m = primes.size();

vector<int> pointers(m, 0);

vector<long> nums(m, 1);

for (int i = 1; i <= n; i++) {

long minNum = INT\_MAX;

for (int j = 0; j < m; j++) {

minNum = min(minNum, nums[j]);

}

dp[i] = minNum;

for (int j = 0; j < m; j++) {

if (nums[j] == minNum) {

pointers[j]++;

nums[j] = dp[pointers[j]] \* primes[j];

}

}

}

return dp[n];

}

};

**复杂度分析：**

时间复杂度：O(nm)，其中n是待求的超级丑数的编号，m是数组primes的长度。需要计算数组dp中的n个元素，每个元素的计算都需要O(m)的时间。

空间复杂度：O(n+m)，其中n是待求的超级丑数的编号，m是数组primes的长度。需要创建数组dp、数组pointers 和数组nums，空间分别是O(n)、$O(m)和O(m)。