264. Ugly Number II

Medium | Topics | Companies | Hint

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

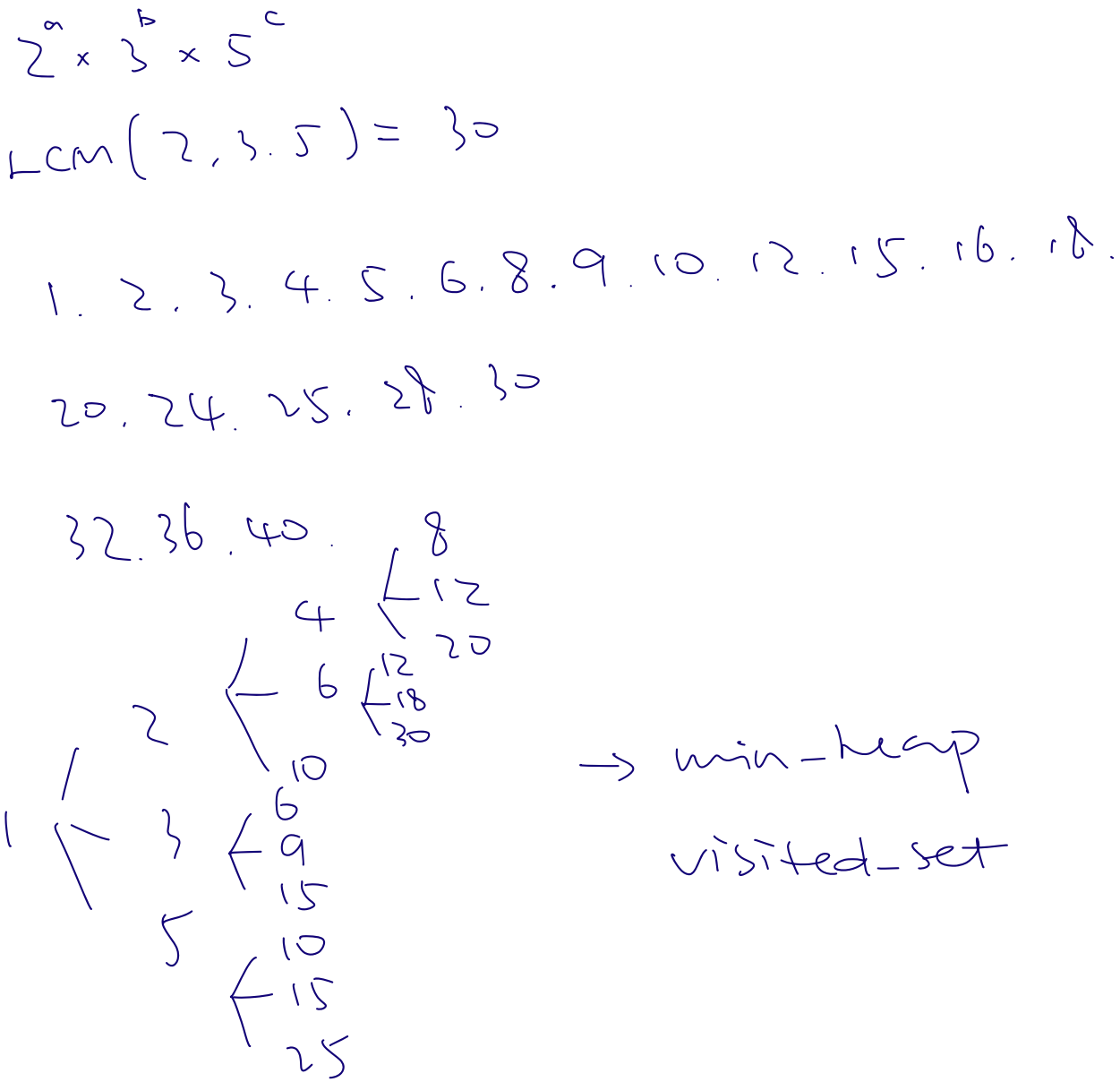
Given an integer `n`, return *the* `n^th` ***ugly number***.

Example 1:

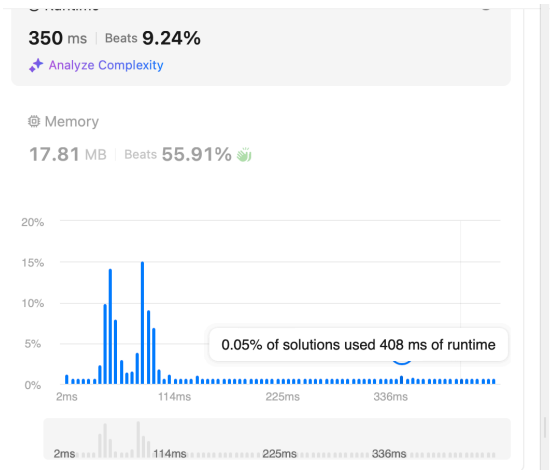
```
Input: n = 10
Output: 12
Explanation: [1, 2, 3, 4, 5, 6, 8, 9, 10, 12] is the
sequence of the first 10 ugly numbers.
```

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.
```



方法1：min 方法 consuming.

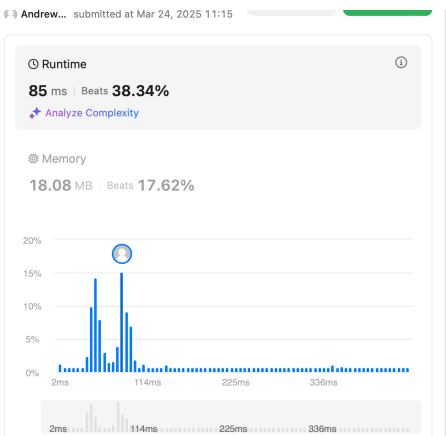



```python
class Solution:
    def nthUglyNumber(self, n: int) -> int:
        q = set()
        q.add(1)
        curr = 1
        for i in range(n):
            curr = min(q)
            q.remove(curr)
            q.add(curr * 2)
            q.add(curr * 3)
            q.add(curr * 5)

        return curr
```

→ $O(n^2)$

方法2：min-heap / priority_queue




```python
class Solution:
    def nthUglyNumber(self, n: int) -> int:
        mh = []
        seen = set()
        factors = [2, 3, 5]

        heapq.heappush(mh, 1)
        seen.add(1)
        curr = 1

        for i in range(n):
            curr = heapq.heappop(mh)

            for f in factors:
                new = f * curr
                if new not in seen:
                    heapq.heappush(mh, new)
                    seen.add(new)

        return curr
```

Ln 20, Col 20 | Saved

Testcase | Test Result

方法3：DP.

Algorithm

1. Initialize a vector `uglyNumbers` of size `n` to store the ugly numbers, with the first ugly number set to `1`.
2. Set up three pointers (`indexMultipleOf2` , `indexMultipleOf3` , `indexMultipleOf5`) to track the next multiples of 2, 3, and 5, respectively.
3. Assign initial values to `nextMultipleOf2` , `nextMultipleOf3` , and `nextMultipleOf5` (i.e., `2` , `3` , and `5`).
4. For `i` from `1` to `n-1` :
   - Determine the next ugly number by taking the minimum of `nextMultipleOf2` , `nextMultipleOf3` , and `nextMultipleOf5` .
   - Store this value in `uglyNumbers[i]` .
   - Update the corresponding pointer and multiple:
     - If the next ugly number equals `nextMultipleOf2` , increment `indexMultipleOf2` and update `nextMultipleOf2` .
     - If the next ugly number equals `nextMultipleOf3` , increment `indexMultipleOf3` and update `nextMultipleOf3` .
     - If the next ugly number equals `nextMultipleOf5` , increment `indexMultipleOf5` and update `nextMultipleOf5` .
5. After completing the loop, return the last element in `uglyNumbers` , which is the `n` th ugly number.

```python

class Solution:
    def nthUglyNumber(self, n: int) -> int:
        ugly_numbers = [0] * n  # DP array to store ugly numbers
        ugly_numbers[0] = 1  # The first ugly number is 1

        # Three pointers for the multiples of 2, 3, and 5
        index_multiple_of_2, index_multiple_of_3, index_multiple_of_5 = 0, 0, 0
        next_multiple_of_2, next_multiple_of_3, next_multiple_of_5 = 2, 3, 5

        # Generate ugly numbers until we reach the nth one
        for i in range(1, n):
            # Find the next ugly number as the minimum of the next multiples
            next_ugly_number = min(
                [next_multiple_of_2, next_multiple_of_3, next_multiple_of_5]
            )
            ugly_numbers[i] = next_ugly_number

            # Update the corresponding pointer and next multiple
            if next_ugly_number == next_multiple_of_2:
                index_multiple_of_2 += 1
                next_multiple_of_2 = ugly_numbers[index_multiple_of_2] * 2
            if next_ugly_number == next_multiple_of_3:
                index_multiple_of_3 += 1
                next_multiple_of_3 = ugly_numbers[index_multiple_of_3] * 3
            if next_ugly_number == next_multiple_of_5:
                index_multiple_of_5 += 1
                next_multiple_of_5 = ugly_numbers[index_multiple_of_5] * 5

        return ugly_numbers[n - 1]  # Return the nth ugly number
```

$O(n)$