Documentation for Solving the nth Super Ugly Number Problem

Problem Overview

The problem involves finding the n-th super ugly number, which is defined as a positive integer whose prime factors are restricted to a given list of prime numbers. Starting from 1, each subsequent super ugly number is generated by multiplying previously generated super ugly numbers with these primes. The sequence must be generated in ascending order, and duplicates are not allowed.

Approach Explanation

The solution is implemented using a **min-heap** (priority queue) to efficiently manage the smallest super ugly numbers. The heap ensures that the smallest element is always extracted first, which is crucial for generating the sequence in ascending order. To avoid duplicate entries in the heap, a **set** is used, which keeps track of all numbers that have been added to the heap. This prevents redundant calculations and ensures efficient memory usage.

Key Steps

1. **Initialization**: Start with the number 1, which is universally considered the first super ugly number. Add it to both the heap and the set.

2. **Iterative Generation**: For each number in the sequence, extract the smallest number from the heap. Multiply it with each prime from the input list to generate new candidates. Push these candidates into the heap only if they are not already present in the set.

3. **Termination:** Repeat this process until nn super ugly numbers are extracted. The last extracted number is the nn-th super ugly number.

Advantages of the Approach

The min-heap approach is optimal because it ensures that the numbers are processed in ascending order without sorting the entire sequence. The use of a set eliminates duplicate entries, which reduces unnecessary computations and heap operations. This method efficiently balances computational and memory constraints, even when nn and the size of the primes list are large.

Algorithm Complexity

The **algorithm's time complexity** is O(n·logk), where n is the number of super ugly numbers to generate, and k is the size of the primes list. The heap operations (insertion and extraction) contribute to the logarithmic factor. The **space complexity** is O(n+k)O(n + k), which accounts for the size of the heap, the set, and the primes list.

Example to Illustrate

For example, with n=12n = 12 and primes = [2, 7, 13, 19], the sequence of super ugly numbers is [1, 2, 4, 7, 8, 13, 14, 16, 19, 26, 28, 32]. The algorithm begins by initializing the heap with 1. At each step, the smallest element is multiplied by all primes, and the resulting numbers are added to the heap if they are not duplicates. After 12 iterations, the 12^{th} super ugly number, 32, is identified.

Constraints and Edge Cases

The constraints of the problem ensure that the algorithm operates within reasonable bounds. $1 \leq n \leq 10^5$ and the primes are guaranteed to be unique and sorted. However, edge cases include:

- When n=1n = 1, the result is always 1, as no multiplication is needed.

- Larger values of n and k require efficient handling of the heap and set to maintain performance.

Practical Applications

This problem is relevant in scenarios involving ordered data generation and priority-based tasks. For example, it can be used to model systems where entities are processed based on a hierarchical order determined by multiple criteria (primes in this case). The heap-based approach and efficient use of memory make it a foundational algorithm for various applications in computer science, such as task scheduling and optimization problems.