Menger Sponge

Like all fractals, the Menger sponge exhibits self-similar structure, and so this problem is inviting a recursive solution. Let's consider the base case first, and then build a full solution. We will store the coordinates as an exact rational number (using the `Rational` package in Python, for instance, or a custom class in C++) in array `coords[3]`.

Base Case: $L = 1$

By inspecting the $L = 1$ cube, we notice a pattern in the subcubes that have been deleted: all cubes that are in *at least two* of the middle row, middle column, or middle slice of the $3 \times 3 \times 3$ grid of subcubes are deleted. We can turn this insight into a predicate that evaluates whether a point is inside the $L = 1$ cube:

```
function inCube(coords)
    count = 0
    for i = 0...2 do
        if 1/3 < coords[i] < 2/3 then
            count = count + 1
        end if
    end for
    return count < 2
end function
```

Note the use of *strict* inequality, to correctly implement the requirement in the problem statement that points exactly on the boundary of cubes count as being in the cube.

Recursive Case

For a cube with $L > 1$, we can first check if the query point is in the level-1 cube. If not, the point is definitely not in the level L cube. Notice that each of the 20 different $\frac{1}{3} \times \frac{1}{3} \times \frac{1}{3}$ subcubes of the level L cube are themselves Menger sponges of level $L - 1$, shrunk by a factor of 3 and translated from the origin to the appropriate spot in the $3 \times 3 \times 3$ grid of subcubes.

We can undo this translation and shrinking to recursively query whether a point in the level-1 cube is in the level-L cube:

```
function menger(L, coords)
    if L == 0 then
        return true
    end if
    if not inCube(coords) then
        return false
    end if
    for i = 0...2 do
        if coords[i] < 1/3 then
            shift[i] = 0
        else if coords[i] < 2/3 then
            shift[i] = 1/3
        else
            shift[i] = 2/3
        end if
        coords[i] = 3 · (coords[i] − shift[i])
    end for
    return menger(L − 1, coords)
end function
```

This solution has time complexity $O(L)$. The problem statement guarantees that $L \leq 10^5$, and so the above recursive solution should fit within the stack space of most programming languages. It's also straightforward to turn the above recursive code into an interative algorithm, if necessary.

Alternate Approach: Ternary Expansion

Notice a couple of facts about the ternary decimal [sic] representation of `coords`$[i]$ which are particularly convenient for solving this problem:

- The condition $\frac{1}{3} < \text{coords}[i] < \frac{2}{3}$ is equivalent to $0.1_3 < \text{coords}[i] < 0.2_3$;
- Shifting coords$[i]$ and multiplying by three is the same as a *left shift* operation on coords$[i]$ in ternary.

To make it easier to iterate over them, let us store the numerators and denominators in arrays `num[3]` and `denom[3]` (so that `num[0]` $= x_{\text{num}}$ etc.). We can then use the above insights to lazily expand `coords[i]` in ternary one digit at a time to see if the query point is in the level-$0, \ldots, L$ cube:

```
for i = 0...L do
    count = 0
    for j = 0...2 do
        quot = num[j] / denom[j]
        rem = num[j] % denom[j]
        if quot == 1 && rem ≠ 0 then          ▷ The second condition enforces strict inequality
            count = count + 1
        end if
        num[j] = 3 · rem                       ▷ Left-shift the ternary decimal
    end for
    if count ≥ 2 then
        return false
    end if
end for
return true
```

This solution also runs in time $O(L)$, but has the advantage of not needing any kind of exact rational number class.