Processing Operations on an Infinite Integer Number Line

Problem Statement:

You are given an infinite integer number line where you can place obstacles and check for block placement.

You need to support the following operations:

1. [1, x] - Place an obstacle at coordinate x.

2. [2, x, size] - Check if a block of given size can be centered at x without overlapping an obstacle.

Return a binary string representing the results of all [2, x, size] operations.

Approach & Explanation:

1. Use a `set` to store obstacles for O(1) lookup time.

2. For [1, x], insert x into the obstacle set.

3. For [2, x, size], determine the range [x - (size // 2), x + (size // 2)].

4. If any value in the range exists in the set, return "0"; otherwise, return "1".

5. Concatenate results of [2, x, size] queries to form the output.

Python Implementation:

```
def process_operations(operations):
    obstacles = set()
    result = []

    for op in operations:
        if op[0] == 1:
            obstacles.add(op[1])
        elif op[0] == 2:
            x, size = op[1], op[2]
            half_size = (size - 1) // 2
            left, right = x - half_size, x + half_size

            if any(pos in obstacles for pos in range(left, right + 1)):
                result.append("0")
            else:
                result.append("1")

    return "".join(result)
```

Example Walkthrough:

Input:

operations = [[1, 3], [2, 0, 3], [1, 0], [2, 0, 3]]

Step-by-step Execution:

- [1, 3]: Place obstacle at 3.
- [2, 0, 3]: Check range [-1, 0, 1] -> No obstacles -> Output "1".
- [1, 0]: Place obstacle at 0.
- [2, 0, 3]: Check range [-1, 0, 1] -> 0 is blocked -> Output "0".

Final Output: "10"

Time Complexity Analysis:

- Inserting an obstacle: O(1) (set insertion)
- Checking block placement: O(K), where K is the block size (usually small)
- Overall, the solution runs efficiently in O(N) for N operations.

Space Complexity Analysis:

- We use a set to store obstacles, which takes O(M) space, where M is the number of obstacles placed.
- The result list takes O(Q) space, where Q is the number of queries.
- Overall, space complexity is O(M + Q).

Conclusion:

This approach efficiently handles obstacle placement and block validation using a set for quick lookups.

The solution scales well and correctly handles edge cases while ensuring optimal time and space complexity.