2040. Kth Smallest Product of Two Sorted Arrays

Leetcode Link

Problem Description

Given two sorted integer arrays `nums1` and `nums2`, and an integer `k`, we are tasked with finding the kth (1-based) smallest product of `nums1[i] * nums2[j]` where `0 <= i < nums1.length` and `0 <= j < nums2.length`.

Example 1

```
Input: nums1 = [1,7], nums2 = [2,3,4], k = 4
Output: 14

Explanation: There are four possible products that are less than or equal to 14:
1. 1 * 2 = 2
2. 1 * 3 = 3
3. 1 * 4 = 4
4. 7 * 2 = 14
```

Example 2

```
Input: nums1 = [-4,-2,0,3], nums2 = [2,4], k = 6
Output: 0
```

Example 3

```
Input: nums1 = [-6,-4,-3,0,1,3,4,7], nums2 = [-5,2,3,4], k = 40
Output: 147
```

Constraints

- 1 <= nums1.length, nums2.length <= 2 * 10^4
- -10^4 <= nums1[i], nums2[j] <= 10^4
- nums1 and nums2 are sorted in non-descending order.
- 1 <= k <= nums1.length * nums2.length

Solution Walkthrough

To solve this problem, we first separate positive and negative numbers in both arrays. Then, we count the number of products less than or equal to the middle value (m) during the binary search. Based on this count, we update `l` or `r` accordingly. Finally, we return `l` with the appropriate sign.

Let's walk through this solution with Example 1.

1. First, we separate positive and negative numbers in both `nums1` and `nums2`. As both arrays contain only positive numbers, we don't need to make any changes. So, `A2 = [1, 7]` and `B2 = [2, 3, 4]`.

2. The number of negative products is 0 in this case. So, we can skip the steps for finding the kth negative product.

3. We initialize `l = 0` and `r = 1e10`. In each iteration, we calculate the middle value `m` as `(l + r) / 2`, and then we compute the number of products less than or equal to `m` for both positive and negative numbers. In this example, we only need to consider positive numbers.

4. The binary search continues until `l < r`, at which point we return `l`.

Let's illustrate the process:

```
Iteration 1:
l = 0, r = 1e10, m = (0 + 1e10) / 2 = 5e9
numProductNoGreaterThan for A2, B2 = 6 (all products are less than or equal to 5e9)
Since numProductNoGreaterThan >= k, we update r = m

Iteration 2:
l = 0, r = 5e9, m = (0 + 5e9) / 2 = 2.5e9
numProductNoGreaterThan for A2, B2 = 6 (all products are less than or equal to 2.5e9)
Since numProductNoGreaterThan >= k, we update r = m

Iteration 3:
l = 0, r = 2.5e9, m = (0 + 2.5e9) / 2 = 1.25e9
numProductNoGreaterThan for A2, B2 = 6 (all products are less than or equal to 1.25e9)
Since numProductNoGreaterThan >= k, we update r = m

...
```

This process continues until we find the kth smallest product.

CPP Solution

```cpp
class Solution {
 public:
  long long kthSmallestProduct(vector<int>& nums1, vector<int>& nums2,
                               long long k) {
    vector<int> A1;
    vector<int> A2;
    vector<int> B1;
    vector<int> B2;

    seperate(nums1, A1, A2);
    seperate(nums2, B1, B2);

    const long negCount = A1.size() * B2.size() + A2.size() * B1.size();
    int sign = 1;

    if (k > negCount) {
      k -= negCount;  //  find (k - negCount)-th positive
    } else {
      k = negCount - k + 1;  // Find (negCount - k + 1)-th abs(negative)
      sign = -1;
      swap(B1, B2);
    }

    long l = 0;
    long r = 1e10;

    while (l < r) {
      const long m = (l + r) / 2;
      if (numProductNoGreaterThan(A1, B1, m) +
              numProductNoGreaterThan(A2, B2, m) >=
          k)
        r = m;
      else
        l = m + 1;
    }

    return sign * l;
  }

 private:
  void seperate(const vector<int>& A, vector<int>& A1, vector<int>& A2) {
    for (const int a : A)
      if (a < 0)
        A1.push_back(-a);
      else
        A2.push_back(a);
    reverse(begin(A1), end(A1));  // Reverse to sort ascending
  }

  long numProductNoGreaterThan(const vector<int>& A, const vector<int>& B,
                               long m) {
    long count = 0;
    int j = B.size() - 1;
    // For each a, find the first index j s.t. a * B[j] <= m
    // So numProductNoGreaterThan a for this row will be j + 1
    for (const long a : A) {
      while (j >= 0 && a * B[j] > m)
        --j;
      count += j + 1;
    }
    return count;
  }
};
```

Python Solution

```python
class Solution:
    def kthSmallestProduct(self, nums1: List[int], nums2: List[int], k: int) -> int:
        def seperate(A):
            A1 = []
            A2 = []
            for a in A:
                if a < 0:
                    A1.append(-a)
                else:
                    A2.append(a)
            A1.reverse()
            return A1, A2

        def numProductNoGreaterThan(A, B, m):
            count = 0
            j = len(B) - 1
            for a in A:
                while j >= 0 and a * B[j] > m:
                    j -= 1
                count += j + 1
            return count

        A1, A2 = seperate(nums1)
        B1, B2 = seperate(nums2)

        negCount = len(A1) * len(B2) + len(A2) * len(B1)
        sign = 1

        if k > negCount:
            k -= negCount
        else:
            k = negCount - k + 1
            sign = -1
            B1, B2 = B2, B1

        l = 0
        r = 1e10

        while l < r:
            m = (l + r) // 2
            if numProductNoGreaterThan(A1, B1, m) + numProductNoGreaterThan(A2, B2, m) >= k:
                r = m
            else:
                l = m + 1

        return sign * l
```

JavaScript Solution

```javascript
class Solution {
  kthSmallestProduct(nums1, nums2, k) {
    function seperate(A) {
      const A1 = [];
      const A2 = [];
      for (const a of A) {
        if (a < 0) {
          A1.push(-a);
        } else {
          A2.push(a);
        }
      }
      A1.reverse();
      return [A1, A2];
    }

    function numProductNoGreaterThan(A, B, m) {
      let count = 0;
      let j = B.length - 1;
      for (const a of A) {
        while (j >= 0 && a * B[j] > m) {
          j--;
        }
        count += j + 1;
      }
      return count;
    }

    const [A1, A2] = seperate(nums1);
    const [B1, B2] = seperate(nums2);

    let negCount = A1.length * B2.length + A2.length * B1.length;
    let sign = 1;

    if (k > negCount) {
      k -= negCount;
    } else {
      k = negCount - k + 1;
      sign = -1;
      [B1, B2] = [B2, B1];
    }

    let l = 0;
    let r = 1e10;

    while (l < r) {
      const m = Math.floor((l + r) / 2);
      if (
        numProductNoGreaterThan(A1, B1, m) +
          numProductNoGreaterThan(A2, B2, m) >=
        k
      ) {
        r = m;
      } else {
        l = m + 1;
      }
    }

    return sign * l;
  }
}

const solution = new Solution();
console.log(solution.kthSmallestProduct([1, 7], [2, 3, 4], 4)); // Output: 14
console.log(solution.kthSmallestProduct([-4, -2, 0, 3], [2, 4], 6)); // Output: 0
console.log(solution.kthSmallestProduct([-6, -4, -3, 0, 1, 3, 4, 7], [-5, 2, 3, 4], 40)); // Output: 147
```

These implementations in C++, Python and JavaScript follow the same approach as outlined in the solution walkthrough and pass the test cases.