Maximum Product of Three Numbers: Algorithm Comparison

Method 1: Linear Scan (Recursive Approach)

📝 Pseudocode

```
function MaxProductLinearScanRecursive(nums):
    return Recurse(nums, 0, -∞, -∞, -∞, ∞, ∞)

function Recurse(nums, index, max1, max2, max3, min1, min2):
    if index == length(nums):
        product1 = max1 * max2 * max3
        product2 = min1 * min2 * max1
        return max(product1, product2)

    num = nums[index]

    if num >= max1:
        max3 = max2
        max2 = max1
        max1 = num
    else if num >= max2:
        max3 = max2
        max2 = num
    else if num >= max3:
        max3 = num

    if num <= min1:
        min2 = min1
        min1 = num
    else if num <= min2:
        min2 = num

    return Recurse(nums, index + 1, max1, max2, max3, min1, min2)
```

🔍 Approach

This method tracks the three largest and two smallest numbers recursively. Two possible products are considered:

- Product of the **three largest** numbers
- Product of the **two smallest** numbers (potentially negative) and the **largest** number

⏱️ Time Complexity

- **Recurrence Relation**: T(n) = T(n-1) + O(1)
  - T(n) represents the time complexity for an array of size n
  - T(n-1) is the recursive call with one fewer element
  - O(1) is the constant time comparison operations at each step
  - Base case: T(0) = O(1)
- Each element is processed exactly once: **O(n)**

Method 2: Heap-Based Approach

📝 Pseudocode

```
function MaxProductHeap(nums):
    largestThree = FindTopK(nums, 3, maxHeap = true)
    smallestTwo = FindTopK(nums, 2, maxHeap = false)

    product1 = largestThree[0] * largestThree[1] * largestThree[2]
    product2 = smallestTwo[0] * smallestTwo[1] * largestThree[0]

    return max(product1, product2)

function FindTopK(nums, k, maxHeap):
    if maxHeap:
        heap = buildMaxHeap(nums)
        extract = extractMax
    else:
        heap = buildMinHeap(nums)
        extract = extractMin

    result = []
    for i from 1 to k:
        result.append(extract(heap))

    return result
```

🔍 Approach

This method extracts the top 3 largest and 2 smallest values using heaps. The same two product possibilities are considered as in the recursive method.

⏱️ Time Complexity

- Building heap: **O(n)**
- Extracting k elements: **O(k log n)**
- Total: **O(n + k log n)** → **O(n)** for fixed small k

📊 Algorithm Comparison

Criteria	Linear Scan (Recursive)	Heap-Based Approach
Time Complexity	O(n)	O(n + k log n) ≈ O(n)
Handles Negatives?	Yes	Yes
Code Simplicity	Simple logic, recursive	More abstract, uses heaps
Generalizable to k?	Not easily	Easily (Find top-k or bottom-k)
Best Use Case	Static arrays, small size	Large arrays, frequent top-k ops

💡 Key Insights

1. **Linear Scan Approach**:
    - More straightforward implementation
    - Better space efficiency with iterative implementation
    - Ideal for one-time processing of smaller arrays
2. **Heap-Based Approach**:
    - More flexible and generalizable to other "top-k" problems
    - Provides a structured data organization
    - Better for scenarios where you need to find extremes frequently
3. **Important Edge Cases**:
    - Arrays with negative numbers (potentially large product from two negatives)
    - Arrays with zeros
    - Arrays with fewer than 3 elements (special handling needed)