Find Kth Smallest Sum of Sub Array

Given an array of integers, find the Kth Smallest sum subarray.

Please note that a subarray is the sequence of consecutive elements of the array.

Example: [3,4,2,8,9,1], k = 4

Output: The 4 largest sum of subarray: 4 (4)

Approach 1: Function to find the kth smallest sum of subarrays using a brute force approach

- **Function Purpose:** This approach finds the Kth smallest sum of subarrays using a brute force method. It calculates all possible subarray sums and returns the Kth smallest sum.
- **Explanation:**
    1. Calculate all possible subarray sums by iterating through each starting and ending index.
    2. Store these sums in a list.
    3. Sort the list of sums in ascending order.
    4. Return the Kth smallest sum, indexed from 1 to K.
- **Time Complexity: O(n^2)**
    - ***Explanation:*** **Calculating all possible subarray sums takes O(n^2) time because for each starting index, we compute the sum for each ending index.**
- **Space Complexity: O(n^2)**
    - ***Explanation:*** **All subarray sums are stored in a vector, resulting in a space complexity of O(n^2).**

Approach 2: Function to find the kth smallest sum of subarrays using a max heap

- **Function Purpose:** This approach efficiently finds the Kth smallest sum of subarrays by maintaining a max heap (priority queue) with the K smallest sums.
- **Explanation:**
    1. Calculate all subarray sums and insert them into a max heap.
    2. Ensure that the heap contains only the K smallest sums by popping elements if the heap size exceeds K.

3. Retrieve the Kth smallest sum from the max heap.

- **Time Complexity: O(n^2 * log n)**
    - ***Explanation:*** **The time complexity arises from inserting subarray sums into the max heap, which takes O(log n) time for each insertion. As there are O(n^2) subarray sums, the overall time complexity is O(n^2 * log n).**
- **Space Complexity: O(n^2)**
    - ***Explanation:*** **The max heap stores all possible subarray sums, leading to a space complexity of O(n^2).**

Approach 3: Function to find the kth smallest sum of subarrays using a min heap

- **Function Purpose:** Similar to Approach 2, this approach efficiently finds the Kth smallest sum of subarrays by maintaining a min heap (priority queue) with the K smallest sums.
- **Explanation:**
    1. Calculate all subarray sums and insert them into a min heap.
    2. Ensure that the heap contains only the K smallest sums.
    3. Retrieve the Kth smallest sum from the min heap.
- **Time Complexity: O(n^2 * log n)**
    - ***Explanation:*** **The time complexity is similar to Approach 2, O(n^2 * log n), due to the insertion of subarray sums into the min heap.**
- **Space Complexity: O(n^2) (worst-case)**
    - ***Explanation:*** **The space complexity depends on K, but in the worst case, it's O(n^2) because the min heap stores all possible subarray sums.**

Approach 4: Function to find the kth smallest sum of subarrays using an optimized approach

- **Function Purpose:** This approach efficiently identifies the Kth smallest sum of subarrays by maintaining a max heap with the K smallest sums while iterating through subarrays.
- **Explanation:**
    1. Maintain a max heap to keep track of the K smallest sums.
    2. During iteration, compare new subarray sums with the largest sum in the max heap. If the new sum is smaller, replace the largest sum.

3. After processing all subarrays, the Kth smallest sum is present in the max heap.

- **Time Complexity: O(n^2 * log k)**
    - ***Explanation:*** **The time complexity is determined by the number of elements in the max heap, which is limited to K. Since there are O(n^2) subarray sums, the time complexity is O(n^2 * log k), where K <= n^2.**
- **Space Complexity: O(K)**
    - ***Explanation:*** **The space complexity depends on the value of K, representing the size of the max heap. In the worst case, it's O(n^2) when K = n^2.**

Conclusion

In conclusion, we have explored four approaches for finding the Kth smallest sum of subarrays within an array. Each approach has its strengths and weaknesses, and the choice depends on the specific requirements and constraints of your problem. **Approach 4 is the most efficient when K is relatively small compared to the array size.**