Solution: Minimum Size Subarray Sum

Let's solve the Minimum Size Subarray Sum problem using the Sliding Window pattern.

We'll cover the following

Statement

Given an array of positive integers, `nums`, and a positive integer, `target`, find the minimum length of a contiguous subarray whose sum is greater than or equal to the `target`. If no such subarray is found, return `0`.

Constraints:

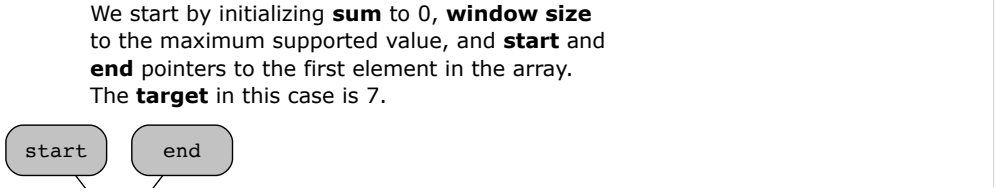
- $1 \leq$ `target` $\leq 10^9$
- $1 \leq$ `nums.length` $\leq 10^5$
- $1 \leq$ `nums[i]` $\leq 10^4$

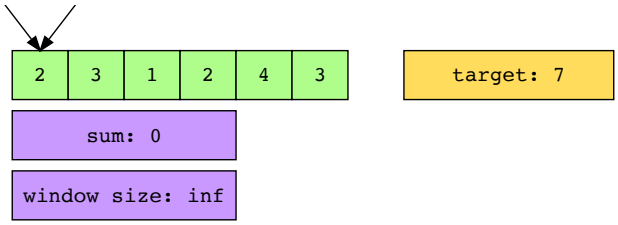
Solution

The idea is to traverse the array using a sliding window, calculate the sum of elements in it, and compare the sum with the target value. If the sum is greater than or equal to the target value, store the size of this window. Repeat this process to find the minimum size subarray.

We perform the following steps to implement the algorithm:

1. We start by initializing a variable, `windowSize`, with positive infinity to store the size of the minimum subarray. In addition, we initialize `sum` with 0.
2. We use the `start` and `end` variables to track the left and right end of the subarray, respectively. Initially, we set both variables to 0.
3. We slide the window over the input array using these two variables. In each iteration, we increment `end` and add the new element of the window into the `sum`. If `sum` is greater than or equal to `target`, we increment `start`.
4. If `sum` exceeds or equals the `target`, we compare the current subarray size with `windowSize`. The smaller of the two values will be stored in `windowSize`.
5. We repeat steps 3 and 4 to find the smallest subarray.
6. Finally, if `windowSize` is positive infinity, we come to know that there was no subarray whose sum was equal to or greater than `target`. Therefore, we return 0. Otherwise, we return `windowSize` as the length of the minimum size subarray.





1 of 18

Let's have a look at code below for the solution:

Java

```java
import java.util.*;

class MinimumSubArraySum {
    public static int minSubArrayLen(int target, int[] nums) {
        // Initializing windowSize to a max number
        int windowSize = Integer.MAX_VALUE;
        int currSubArrSize = 0;
        // Initialize start pointer to 0 and sum to 0
        int start = 0;
        int sum = 0;

        // Iterate over the input array
        for (int end = 0; end < nums.length; end++) {
            sum += nums[end];
            // check if we can remove elements from the start of the subarray
            // while still satisfying the target condition
            currSubArrSize = (end + 1) - start;
            windowSize = Math.min(windowSize, currSubArrSize);
            sum -= nums[start];
            start += 1;
        }

    }

    if (windowSize != Integer.MAX_VALUE) {
        return windowSize;
```

Minimum Size Subarray Sum

Time complexity

The time complexity of this solution is $O(n)$.

Space complexity

The space complexity of this solution is $O(1)$, since we are not using any extra space.

Back
Minimum Size Subarra...
Next
Best Time to Buy and ...
Mark as Completed