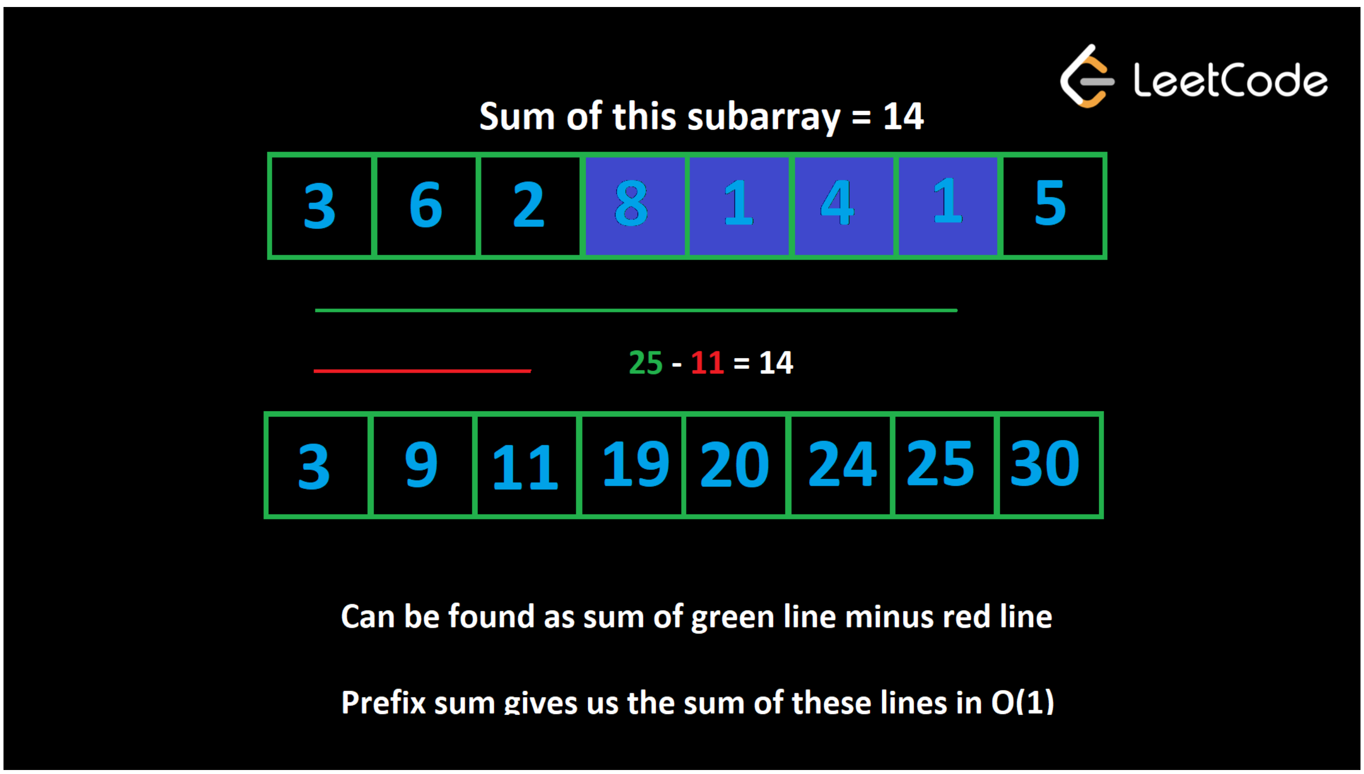
Prefix sum

The idea of a prefix sum is to create an array `prefix` where `prefix[i]` is the sum of all elements up to the index `i` (inclusive). For example, given `nums = [5, 2, 1, 6, 3, 8]`, we would have `prefix = [5, 7, 8, 14, 17, 25]`.

Prefix sums allows us to find the sum of any subarray in $O(1)$ (assuming we have the prefix sums).
The sum of the subarray from `i` to `j` (inclusive) is `prefix[j] - prefix[i - 1]`, or `prefix[j] - prefix[i] + nums[i]` if you don't want to deal with the out of bounds case when `i = 0`.



Given an array nums,

```
prefix = [nums[0]]
for (int i = 1; i < nums.length; i++)
    prefix.append(nums[i] + prefix[prefix.length - 1])
```

Example 1: Given an integer array `nums`. an array `queries` where `queries[i] = [x, y]` and an integer `limit`, return a boolean array that represents the answer to each query. A query is `true` if the sum of the subarray from `x` to `y` is less than `limit`, or `false` otherwise.

For example, given `nums = [1, 6, 3, 2, 7, 2]`, `queries = [[0, 3], [2, 5], [2, 4]]`, and `limit = 13`, the answer is `[true, false, true]`. For each query, the subarray sums are `[12, 14, 12]`.

Let's build a prefix sum and then use the method described above to answer each query in $O(1)$
.

```
def answer_queries(nums, queries, limit):
    prefix = [nums[0]]
    for i in range(1, len(nums)):
        prefix.append(nums[i] + prefix[-1])

    ans = []
    for x, y in queries:
        curr = prefix[y] - prefix[x] + nums[x]
        ans.append(curr < limit)

    return ans
```

Without the prefix sum, answering each query would be $O(n)$ in the worst case, where n is the length of `nums`. If `m = queries.length`, that would give a time complexity of $O(n * m)$. With the prefix sum, it costs $O(n)$ to build, but then answering each query is $O(1)$. This gives a much better time complexity of $O(n + m)$. We use $O(n)$ space to build the prefix sum.

Example 2: 2270 - Number of Ways to Split Array

Given an integer array `nums`, find the number of ways to split the array into two parts so that the first section has a sum greater than or equal to the sum of the second section. The second section should have at least one number.

Brute force recalculating sums for each i would yield a $O(n^2)$ time complexity. If we build a prefix sum, we could reduce that to one linear sum calculation followed by subtractions. This would reduce the time complexity to $O(n)$ but would also raise the space complexity to $O(n)$.

But we don't actually need the array. We can just initialize `leftSection = 0` and then calculate it on the fly by adding the current element to it at each iteration.

Right sum will always `total - leftSection` so we just need to calculate the sum of the whole array once at the start to get `total`.

```
class Solution:
    def waysToSplitArray(self, nums: List[int]) -> int:
        ans = left_section = 0
        total = sum(nums)

        for i in range(len(nums) - 1):
            left_section += nums[i]
            right_section = total - left_section
            if left_section >= right_section:
                ans += 1

        return ans
```

We have improved the space complexity to $O(1)$ from $O(n)$, which is a great improvement.

SEE PROBLEMS 1480, 1413, 2090