Daily Coding Problem #243

Problem

This problem was asked by Etsy.

Given an array of numbers `N` and an integer k, your task is to split `N` into k partitions such that the maximum sum of any partition is minimized. Return this sum.

For example, given `N = [5, 1, 2, 7, 3, 4]` and `k = 3`, you should return 8, since the optimal partition is `[5, 1, 2], [7], [3, 4]`.

Solution

We can begin by identifying a few facts. First, we know that the sum cannot be smaller than the largest element in the list, because that has to be in one of the partitions. On the other hand, it cannot be largest than the sum of the whole list.

We could proceed by checking each value between the maximum element and the array sum until finding the smallest one that works. A more efficient way, however, would be to perform a binary search.

```
def max_partition_sum(array, k):
    left, right = max(array), sum(array)
```

```
    while left < right:
        mid = (left + right) // 2
        if can_partition(array, mid, k):
            right = mid
        else:
            left = mid + 1

    return left
```

We still need to implement `can_partition`. That is, we must check, for a given candidate limit, whether an array can be split into k partitions such that no partition exceeds this limit. But here there is a greedy solution: we traverse the list, adding elements into the current partition until its sum exceeds the limit. In that case, we start a new partition with the current element. If the number of partitions required for such a task exceeds k, we know it is impossible.

```
def can_partition(array, limit, k):
    total = 0
    partitions = 1

    for num in array:
        if total + num > limit:
            total = num
            partitions += 1
            if partitions > k:
                return False
        else:
            total += num

    return True
```

Each call to `can_partition` runs in `O(N)` time, since we must traverse the whole array in the worst case. The number of calls is at most `log R`, where `R` is the difference between the array sum and the maximum element. So overall the time complexity is `O(N log R)`.