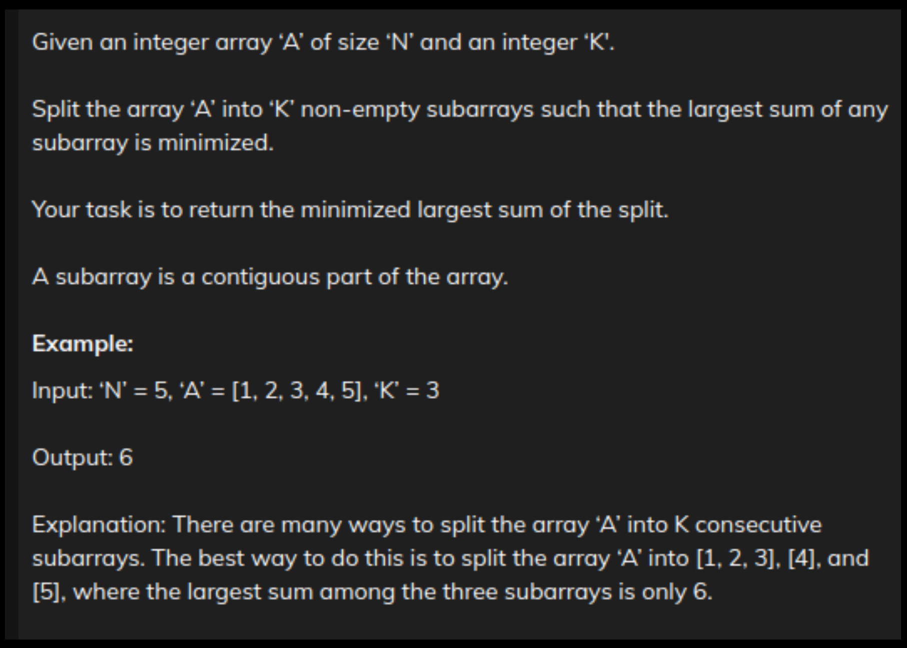
④ Largest Subarray Sum Minimized



Given an integer array 'A' of size 'N' and an integer 'K'.

Split the array 'A' into 'K' non-empty subarrays such that the largest sum of any subarray is minimized.

Your task is to return the minimized largest sum of the split.

A subarray is a contiguous part of the array.

Example:

Input: 'N' = 5, 'A' = [1, 2, 3, 4, 5], 'K' = 3

Output: 6

Explanation: There are many ways to split the array 'A' into K consecutive subarrays. The best way to do this is to split the array 'A' into [1, 2, 3], [4], and [5], where the largest sum among the three subarrays is only 6.

→ Brute force :-

The answer will lie b/w max(arr) & the sum of array.

Now take each the value b/w the above range and check if it can be our answer or not.

```
int countPartitions(vector<int> &a, int maxSum) {
    int n = a.size();
    int partitions = 1;
    long long subarraySum = 0;
    for (int i = 0; i < n; i++) {
        if (subarraySum + a[i] <= maxSum) {
            subarraySum += a[i];
        }
        else {
            partitions++;
            subarraySum = a[i];
        }
    }
    return partitions;
}
```

```
int largestSubarraySumMinimized(vector<int> &a, int k) {
    int low = *max_element(a.begin(), a.end());
    int high = accumulate(a.begin(), a.end(), 0);

    for (int maxSum = low; maxSum <= high; maxSum++) {
        if (countPartitions(a, maxSum) == k)
            return maxSum;
    }
    return low;
}
```

→ We can optimize this part using binary search.

```
int largestSubarraySumMinimized(vector<int> &a, int k) {
    int low = *max_element(a.begin(), a.end());
    int high = accumulate(a.begin(), a.end(), 0);
    while (low <= high) {
        int mid = (low + high) / 2;
        int partitions = countPartitions(a, mid);
        if (partitions > k) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
    }
    return low;
}
```