# **Leetcode 410 – Split Array Largest Sum**

## Problem Understanding

You're given an integer array nums and an integer k.  
Split the array into k **non-empty, continuous subarrays**, such that the **largest sum among them is minimized**.

**Goal**: Return the **minimum possible value of the largest sum** among these k subarrays.

### Example

Input: nums = [7,2,5,10,8], k = 2

Output: 18

Explanation: Split into [7,2,5] and [10,8], sums = 14 and 18 → max = 18 (minimum possible max)

## Optimized Java Solution (Binary Search on Answer)

We **binary search** on the possible range of answers:

* **Lower bound** = max element in nums
* **Upper bound** = total sum of nums

class Solution {

public int splitArray(int[] nums, int k) {

int left = getMax(nums);

int right = getSum(nums);

int result = right;

while (left <= right) {

int mid = left + (right - left) / 2;

if (canSplit(nums, k, mid)) {

result = mid;

right = mid - 1; // try smaller max sum

} else {

left = mid + 1; // need larger max sum

}

}

return result;

}

private boolean canSplit(int[] nums, int k, int maxSum) {

int count = 1;

int currentSum = 0;

for (int num : nums) {

if (currentSum + num > maxSum) {

count++;

currentSum = num;

if (count > k) return false;

} else {

currentSum += num;

}

}

return true;

}

private int getMax(int[] nums) {

int max = 0;

for (int n : nums) max = Math.max(max, n);

return max;

}

private int getSum(int[] nums) {

int total = 0;

for (int n : nums) total += n;

return total;

}

}

## Dry Run Using Table

### Input:

nums = [7,2,5,10,8], k = 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Iteration | Left | Right | Mid | Can Split? | Result | Action |
| 1 | 10 | 32 | 21 | Yes | 21 | Try smaller sum |
| 2 | 10 | 20 | 15 | No | 21 | Try larger sum |
| 3 | 16 | 20 | 18 | Yes | 18 | Try smaller sum |
| 4 | 16 | 17 | 16 | No | 18 | Try larger sum |
| 5 | 17 | 17 | 17 | No | 18 | Try larger sum |
| 6 | 18 | 17 | — | — | 18 | Stop |

✅ Final Answer: **18**

## Time / Space Complexity

|  |  |
| --- | --- |
| Metric | Value |
| Time | O(n × log(sum)) |
| Space | O(1) |

Where:

* n = nums.length
* sum = total sum of array (upper bound for search)

## Alternate Approaches

|  |  |  |
| --- | --- | --- |
| Approach | Time Complexity | Notes |
| ✅ Binary Search | O(n log sum) | Optimal, recommended |
| ❌ DP (memoized) | O(n² × k) | Too slow for large inputs |