

# Split Array Largest Sum:-

**410. Split Array Largest Sum**

Given an integer array `nums` and an integer `k`, split `nums` into `k` non-empty subarrays such that the largest sum of any subarray is **minimized**.

Return the **minimized largest sum** of the split.

A **subarray** is a contiguous part of the array.

**Example 1:**

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

Output: `18`

Explanation: There are four ways to split `nums` into two subarrays. The best way is to split it into `[7,2,5]` and `[10,8]`, where the largest sum among the two subarrays is only `18`.

**Example 2:**

Input: `nums = [1,2,3,4,5]`, `k = 2`

Output: `9`

Explanation: There are four ways to split `nums` into two subarrays. The best way is to split it into `[1,2,3]` and `[4,5]`, where the largest sum among the two subarrays is only `9`.

**Constraints:**

- `1 <= nums.length <= 1000`
- `1 <= nums[i] <= 10^4`
- `1 <= k <= min(10, nums.length)`

Typical minimum of maximum / maximum of minimum problem.

Binary Search.

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

① `[7]` `[2, 5, 10, 8]` → 28

② `[7, 2]` `[5, 10, 8]` → 23

③ `[7, 2, 5]` `[10, 8]` → 18 ← minimum ✓

④ `[7, 2, 5, 10]` `[8]` → 24

Let's try to build intuition:-

range:- `max(nums)` → `sum(nums)`

Let `S` be the maximum sum of any subarray, check if total subarrays = `k`.

eg in `[7, 2, 5, 10, 8]`

`lo = 10` `hi = 7 + 2 + 5 + 10 + 8 = 32`

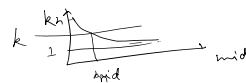
`mid = 21` `18 → (18 - 31)`

21 `7, 2, 8` `(7, 2, 5)(10, 8)`  
`hi = mid`

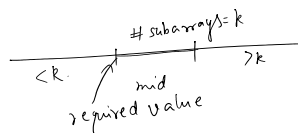
23 `24`  
21 `25`

14 `10, 8`  
18 `21` `19` `18, 11`  
15, `21` `18, 11`  
`18`

`mid ↑ k ↓`  
`mid ↓ k ↑`  
`mid:`



There will be a range of `mid` where # subarrays = `k`



`while (lo < hi)`  
`if (test(mid) > k) hi = mid`

`else lo = mid + 1;`  
`;`

`return lo`

// debugging

eg: `a = [1, 1]`

`k = 1`

① `mid = 1`

`[1, 1]` `1`

↑  
`[1, 1]` `1+1 = 2`  
↑ `ans = 1`

`return ans = 2;`

`test > mid`

`lo = 2`

## Solution Code & Result :-

The screenshot displays a LeetCode submission interface. On the left, the 'Accepted' status is confirmed with a green checkmark. Below this, a graph shows the runtime performance, with a single data point at 0 ms. The memory usage is 10.38 MB, and the solution is 86.42% efficient. The C++ code is shown in the center, featuring a recursive function to split the array into k subarrays. The code is as follows:

```
class Solution {
public:
    int splitArray(vector<int>& nums, int k) {
        int n = nums.size();
        int lo = 0, hi = 0;
        for (int i = 0; i < n; i++) {
            lo = max(lo, nums[i]);
            hi += nums[i];
        }
        return splitArray(nums, k, lo, hi);
    }
};
```

At the bottom, there are links to 'More challenges' and 'Test Result'.