

# Problem 410: Split Array Largest Sum

## Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

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

0 <= nums[i] <= 10

6

1 <= k <= min(50, nums.length)

## Code Snippets

### C++:

```
class Solution {
public:
    int splitArray(vector<int>& nums, int k) {

    }
};
```

### Java:

```
class Solution {
    public int splitArray(int[] nums, int k) {

    }
}
```

### Python3:

```
class Solution:
    def splitArray(self, nums: List[int], k: int) -> int:
```

### Python:

```
class Solution(object):
    def splitArray(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
```

```
"""
```

### JavaScript:

```
/**
 * @param {number[]} nums
 * @param {number} k
 * @return {number}
 */
var splitArray = function(nums, k) {

};
```

### TypeScript:

```
function splitArray(nums: number[], k: number): number {

};
```

### C#:

```
public class Solution {
    public int SplitArray(int[] nums, int k) {

    }
}
```

### C:

```
int splitArray(int* nums, int numsSize, int k) {

}
```

### Go:

```
func splitArray(nums []int, k int) int {

}
```

### Kotlin:

```

class Solution {
    fun splitArray(nums: IntArray, k: Int): Int {

    }
}

```

### Swift:

```

class Solution {
    func splitArray(_ nums: [Int], _ k: Int) -> Int {

    }
}

```

### Rust:

```

impl Solution {
    pub fn split_array(nums: Vec<i32>, k: i32) -> i32 {

    }
}

```

### Ruby:

```

# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def split_array(nums, k)

end

```

### PHP:

```

class Solution {

    /**
     * @param Integer[] $nums
     * @param Integer $k
     * @return Integer
     */
    function splitArray($nums, $k) {

    }
}

```

```
}
```

### Dart:

```
class Solution {  
  int splitArray(List<int> nums, int k) {  
  
  }  
}
```

### Scala:

```
object Solution {  
  def splitArray(nums: Array[Int], k: Int): Int = {  
  
  }  
}
```

### Elixir:

```
defmodule Solution do  
  @spec split_array(nums :: [integer], k :: integer) :: integer  
  def split_array(nums, k) do  
  
  end  
end
```

### Erlang:

```
-spec split_array(Nums :: [integer()], K :: integer()) -> integer().  
split_array(Nums, K) ->  
.
```

### Racket:

```
(define/contract (split-array nums k)  
  (-> (listof exact-integer?) exact-integer? exact-integer?)  
  )
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Split Array Largest Sum
 * Difficulty: Hard
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

class Solution {
public:
    int splitArray(vector<int>& nums, int k) {

    }
};
```

### Java Solution:

```
/**
 * Problem: Split Array Largest Sum
 * Difficulty: Hard
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

    }
}
```

### Python3 Solution:

```
"""
Problem: Split Array Largest Sum
Difficulty: Hard
Tags: array, dp, greedy, search
```

```

Approach: Use two pointers or sliding window technique
Time Complexity:  $O(n)$  or  $O(n \log n)$ 
Space Complexity:  $O(n)$  or  $O(n * m)$  for DP table
"""

class Solution:
    def splitArray(self, nums: List[int], k: int) -> int:
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

class Solution(object):
    def splitArray(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """

```

### JavaScript Solution:

```

/**
 * Problem: Split Array Largest Sum
 * Difficulty: Hard
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity:  $O(n)$  or  $O(n \log n)$ 
 * Space Complexity:  $O(n)$  or  $O(n * m)$  for DP table
 */

/**
 * @param {number[]} nums
 * @param {number} k
 * @return {number}
 */
var splitArray = function(nums, k) {

};

```

## TypeScript Solution:

```
/**
 * Problem: Split Array Largest Sum
 * Difficulty: Hard
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

function splitArray(nums: number[], k: number): number {

};
```

## C# Solution:

```
/*
 * Problem: Split Array Largest Sum
 * Difficulty: Hard
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

public class Solution {
    public int SplitArray(int[] nums, int k) {

    }
}
```

## C Solution:

```
/*
 * Problem: Split Array Largest Sum
 * Difficulty: Hard
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
```

```

* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int splitArray(int* nums, int numsSize, int k) {

}

```

### Go Solution:

```

// Problem: Split Array Largest Sum
// Difficulty: Hard
// Tags: array, dp, greedy, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}

```

### Kotlin Solution:

```

class Solution {
    fun splitArray(nums: IntArray, k: Int): Int {

    }
}

```

### Swift Solution:

```

class Solution {
    func splitArray(_ nums: [Int], _ k: Int) -> Int {

    }
}

```

### Rust Solution:

```

// Problem: Split Array Largest Sum
// Difficulty: Hard
// Tags: array, dp, greedy, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
    pub fn split_array(nums: Vec<i32>, k: i32) -> i32 {

    }
}

```

### Ruby Solution:

```

# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def split_array(nums, k)

end

```

### PHP Solution:

```

class Solution {

    /**
     * @param Integer[] $nums
     * @param Integer $k
     * @return Integer
     */
    function splitArray($nums, $k) {

    }

}

```

### Dart Solution:

```

class Solution {
    int splitArray(List<int> nums, int k) {

```

```
}  
}
```

### Scala Solution:

```
object Solution {  
  def splitArray(nums: Array[Int], k: Int): Int = {  
  
  }  
}
```

### Elixir Solution:

```
defmodule Solution do  
  @spec split_array(nums :: [integer], k :: integer) :: integer  
  def split_array(nums, k) do  
  
  end  
end
```

### Erlang Solution:

```
-spec split_array(Nums :: [integer()], K :: integer()) -> integer().  
split_array(Nums, K) ->  
.
```

### Racket Solution:

```
(define/contract (split-array nums k)  
  (-> (listof exact-integer?) exact-integer? exact-integer?)  
  )
```