如果想用二分查找优化算法，可以将算法的暴力递归形式写出来，如果含有以下形式：

```
// func(int i)是一个单调函数(递增，递减均可)
int func(int i);

// 形如如下形式的for循环可以采用二分查找优化效率
for (int i =0;i<target;i++) {
  if (func(i) == target) {
    return i;
  }
}
```

`func(int i)`单调函数，表示其随着i增加而增加，或随着`i`增加而减小。

> 给你输入一个数组nums和数字m，你要把nums分割成m个子数组。
> 肯定有不止一种分割方法，每种分割方法都会把nums分成m个子数组，这m个子数组中肯定有一个和最大的子数组对吧。

分析： 固定了m的值，让我们确定一个最大子数组和；所谓反向思考就是说，我们可以反过来，限制一个最大子数组和max，来反推最大子数组和为max时，至少可以将nums分割成几个子数组。

```
class Solution {
public:
  int splitArray(std::vector<int>& nums, int m) {
    int left = getMax(nums), right = getSum(nums);

    for (int i = left; i <= right; i++) {
      int n = split(nums, i);
      if (n <= m) {
        return i;
      }
    }
    return -1;
  }

private:
  int getMax(std::vector<int>& nums) {
    int max = INT_MIN;
    for (auto item : nums) {
      max = (max > item ? max : item);
    }

    return max;
  }

  int getSum(std::vector<int>& nums) {
    int sum = 0;
```

```
    for (int i = 0; i < nums.size(); i++) {
      sum += nums[i];
    }

    return sum;
  }

  // 当最大子数组和为sum,  最多可以分为多少组
  int split(std::vector<int>& nums, int sum) {
    int count = 1;
    int s     = 0;
    for (int i = 0; i < nums.size(); i++) {
      if (s + nums[i] > sum) {
        count++;
        s = nums[i];
      } else {
        s += nums[i];
      }
    }

    return count;
  }
};
```

对结果集选择遍历的形式容易超时。 从二分搜索，看查找结果的左右边界：

```
class Solution {
public:
  int splitArray(std::vector<int>& nums, int m) {
    int left = getMax(nums), right = getSum(nums);

    while (left <= right) {
      int mid = left + (right - left) / 2;
      int n   = split(nums, mid);
      if (n == m) {
        // 收缩右边界达到搜索左边界的目的
        right = mid - 1;
      } else if (n < m) {
        // 最大上限和较大收缩右边界
        right = mid - 1;
      } else {
        // 最大下限和较小，收缩左边界
        left = mid + 1;
      }
    }
    return left;
  }

private:
  int getMax(std::vector<int>& nums) {
    int max = INT_MIN;
    for (auto item : nums) {
```

```
      max = (max > item ? max : item);
    }

    return max;
  }

  int getSum(std::vector<int>& nums) {
    int sum = 0;
    for (int i = 0; i < nums.size(); i++) {
      sum += nums[i];
    }

    return sum;
  }

  // 当最大子数组和为sum,  最多可以分为多少组
  int split(std::vector<int>& nums, int sum) {
    int count = 1;
    int s     = 0;
    for (int i = 0; i < nums.size(); i++) {
      if (s + nums[i] > sum) {
        count++;
        s = nums[i];
      } else {
        s += nums[i];
      }
    }
    return count;
  }
};
```