# 题目

给定一个整数数组nums ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

**示例:**

输入：[-2,1,-3,4,-1,2,1,-5,4],

输出：6

解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。

**进阶:**

如果你已经实现复杂度为O(n) 的解法，尝试使用更为精妙的分治法求解。

# 分析

## 方法一：暴力破解

## 方法二：分治

## 方法三：动态规划

**思路：初始值+状态定义+状态转移方程+结束条件**

**动态规划解析：**

**状态定义：**

设动态规划列表dp，dp[i]代表以元素nums[i]为结尾的连续子数组最大和。

为何定义最大和dp[i]中必须包含元素nums[i]：保证dp[i]递推到dp[i+1]的正确性；如果不包含nums[i]，递推时则不满足题目的连续子数组要求。

**转移方程：**

若dp[i−1]≤0，说明dp[i−1]对dp[i]产生负贡献，即dp[i-1] + nums[i]还不如nums[i]本身大。

当dp[i - 1] > 0时：执行dp[i]=dp[i−1]+nums[i]；

当dp[i−1]≤0时：执行dp[i]=nums[i] ；

**初始状态：**

dp[0]=nums[0]，即以nums[0]结尾的连续子数组最大和为nums[0]。

**返回值：**

返回dp列表中的最大值，代表全局最大值。

**空间复杂度降低：**

由于dp[i]只与dp[i−1]和nums[i]有关系，因此可以将原数组nums用作dp列表，即直接在nums上修改即可。

由于省去dp列表使用的额外空间，因此空间复杂度从O(N)降至O(1)。

**代码：**

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int sumMax = nums[0];

for(int i=1;i<nums.size();i++)

{

if(nums[i-1] > 0)

nums[i] += nums[i-1];

sumMax = std::max(sumMax,nums[i]);

}

return sumMax;

}

};

**复杂度分析：**

时间复杂度O(N) ：线性遍历数组nums 即可获得结果，使用O(N)时间。

空间复杂度O(1) ：使用常数大小的额外空间。