连续子数组的最大和\_剑指Offer\_48

# 连续子数组的最大和\_剑指Offer\_48

## 题目介绍：

\* HZ偶尔会拿些专业问题来忽悠那些非计算机专业的同学。今天测试组开完会后,

\* 他又发话了:在古老的一维模式识别中,常常需要计算连续子向量的最大和,

\* 当向量全为正数的时候,问题很好解决。但是,如果向量中包含负数,

\* 否应该包含某个负数,并期望旁边的正数会弥补它呢？

\* 例如:{6,-3,-2,7,-15,1,2,2},连续子向量的最大和为8(从第0个开始,到第3个为止)。

\* 给一个数组，返回它的最大连续子序列的和，你会不会被他忽悠住？(子向量的长度至少是1)

## 思路分析：

### 分析数组规律

当前元素索引为n，则定义preSum保存之前递进的和；maxSum保存之前出现的和的最大值；

preSum初始化为Integer的最小值，若preSum小于等于0，则对以后的和累加是负作用，故舍弃之前的preSum，直接令preSum=nums[n];而若preSum大于0，则继续累加preSum=preSum+nums[n];

每次当前和之后，都要和maxSum作比较，左preSum大于maxSum，则maxSum需要更新。

### 动态规划解题

\* 动态规划 **g(n) = max(f(n));**

\* 当n=0或f(n-1)<=0时，f(n)=nums[n];

\* 当n!=0且f(n-1)>0时，f(n)=nums[n]+f(n-1);

\* 注意: maxSum初始化为Integer的最小值；

## Java代码

public int **FindGreatestSumOfSubArray**(int[] nums) {

if(nums == null|| nums.length == 0) return 0;

int preSum = 0,**maxSum = Integer.MIN\_VALUE**;

for(int i = 0;i < nums.length;i++){

if(preSum <= 0) preSum = nums[i]; //i==0时，preSum=0

else preSum += nums[i];

if(preSum > maxSum) **maxSum = preSum;** //相当于g(n)=max[f(n)]

}

return maxSum;

}