Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

题目： 给你一个array，找一串连续的子array，这个array有最大sum

**Example:**

**Input:** [-2,1,-3,4,-1,2,1,-5,4],

**Output:** 6

**Explanation:** [4,-1,2,1] has the largest sum = 6.

最基础的方法，暴力解

两个for遍历，一个for求和

例如以i=0,也就是-2，把所有可能的加一遍一直加到4

然后再i=1也就是1，

一直求出最大值

问题时间复杂度太高

三个for就是n^3

优化方法

前缀和

注意和前一种方法相比 多了一个sum

原来方法每一次都要从头到尾相加

这一次设计了一个前缀sum， 每次在sum的基础上加，然后比较maxSum与sum

复杂度n^2

class MaximumSubarrayPrefixSum {

public int maxSubArray(int[] nums) {

int len = nums.length;

int maxSum = Integer.MIN\_VALUE;

int sum = 0;

for (int i = 0; i < len; i++) {

sum = 0;

for (int j = i; j < len; j++) {

sum += nums[j];

maxSum = Math.max(maxSum, sum);

}

}

return maxSum;

}

}

super前缀和

复杂度n

加了一个minSum

我们要注意到找到这样一个array，前面的和必然是最小的（负数），如果是正的那就可以一直加了，然后再用后面的array尽量取最大的

sum是一直到i的的二和

maxSum是当前array减去前面最小值（最小值随着sum的改动而更新）

minSum是当前array与之前最小值取最小值， minSum更新就代表着出现了一个新的负数，把前面的所有正数都抵消掉了还有负，

假设是 +-+++-++，每次-都能盖住

那么第一次，minSum被更新，从此都一第一个负后面为开头，所以Sum不用减掉那部分，因此可以减去MinSum（减去这个负数实际上是加）

所以每次sum都要减，因此每次都更新，直到我们发现这个SUM比minSum还低，也就是说又要更新

class Solution {

public int maxSubArray(int[] nums) {

int maxSum = nums[0];

int sum = 0;

int minSum = 0;

for (int num : nums) {

// prefix Sum

sum += num;

// update maxSum

maxSum = Math.max(maxSum, sum - minSum);

// update minSum

minSum = Math.min(minSum, sum);

}

return maxSum;

}

}

分治法

class Solution {

public int maxSubArray(int[] nums) {

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

return helper(nums, 0, nums.length - 1);

}

private int helper(int[] nums, int l, int r) {

if (l > r) return nums[0];

int mid = (l + r) /2;

int left = helper(nums, l, mid - 1);

int right = helper(nums, mid + 1, r);

int leftMaxSum = 0;

int sum = 0;

// left surfix maxSum start from index mid - 1 to l

for (int i = mid - 1; i >= l; i--) {

sum += nums[i];

leftMaxSum = Math.max(leftMaxSum, sum);

}

int rightMaxSum = 0;

sum = 0;

// right prefix maxSum start from index mid + 1 to r

for (int i = mid + 1; i <= r; i++) {

sum += nums[i];

rightMaxSum = Math.max(sum, rightMaxSum);

}

// max(left, right, crossSum)

return Math.max(leftMaxSum + rightMaxSum + nums[mid], Math.max(left, right));

}

}

动态规划

class Solution {

public int maxSubArray(int[] nums) {

int currMaxSum = nums[0];

int maxSum = nums[0];

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

currMaxSum = Math.max(currMaxSum + nums[i], nums[i]);

maxSum = Math.max(maxSum, currMaxSum);

}

return maxSum;

}

}