Analysis of this problem:  
Apparently, this is a optimization problem, which can be usually solved by DP. So when it comes to DP, the first thing for us to figure out is the format of the sub problem(or the state of each sub problem). The format of the sub problem can be helpful when we are trying to come up with the recursive relation.

At first, I think the sub problem should look like: maxSubArray(int A[], int i, int j), which means the maxSubArray for A[i: j]. In this way, our goal is to figure out what maxSubArray(A, 0, A.length - 1) is. However, if we define the format of the sub problem in this way, it's hard to find the connection from the sub problem to the original problem(at least for me). In other words, I can't find a way to divided the original problem into the sub problems and use the solutions of the sub problems to somehow create the solution of the original one.

So I change the format of the sub problem into something like: maxSubArray(int A[], int i), which means the maxSubArray for A[0:i ] which must has A[i] as the end element. Note that now the sub problem's format is less flexible and less powerful than the previous one because there's a limitation that A[i] should be contained in that sequence and we have to keep track of each solution of the sub problem to update the global optimal value. However, now the connect between the sub problem & the original one becomes clearer:

maxSubArray(A, i) = maxSubArray(A, i - 1) > 0 ? maxSubArray(A, i - 1) : 0 + A[i];

And here's the code

public int maxSubArray(int[] A) {

int n = A.length;

int[] dp = new int[n];//dp[i] means the maximum subarray ending with A[i];

dp[0] = A[0];

int max = dp[0];

for(int i = 1; i < n; i++){

dp[i] = A[i] + (dp[i - 1] > 0 ? dp[i - 1] : 0);

max = Math.max(max, dp[i]);

}

return max;

}