problem a08. b08:

Given an array of integers, find a consecutive sub-array, including empty sub-array, with maximized sum.

兩題題目相同但a08: n<4000; b08: n<=600000

Analysis: 在一個正整數的 array 中找連續的一段使得總和最大。
如果資料在 a[1..n]，a[0]=0.
是甚麼是所有可能得解? 對於所有的 0<=i<=j<n, (a[0]用來對付 empty subarray (sum=0)). If we try all possible solution
=>O(n^2) possible solution, each takes O(n) time.
=>O(n^3) time complexity =>TLE

How to save time?
Suppose we first compute the prefix-sum, i.e., letting pre[i]=a[0]+a[1]+…a[i]
⇨ a[i]+a[i+1]+…a[j]=pre[j]-pre[i-1]
⇨ 每一個連續 subarray 的和可以在 O(1)算出
⇨ time complexity=O(n^2)
但如何計算 pre[i]?
pre[0]=a[0]=0
pre[i]=pre[i-1]+a[i]
簡單的 DP 可以在 O(n)算出所有 prefix-sum, 但需要 O(n)的 space

這個技巧是”空間換取時間”, 利用 preprocessing 算出一些中間結果, 讓後續的計算加快

---

b08: n 可能達到 60000, O(n^2)無法在 3 sec 得到解。
如何加速?
方法不只一種。
以 DP 的概念，令 f(i)是以 i 為右端點的最大 subarray 總和，則
f(1)=a[1]，
For i>1, we have
f(i)=(f(i-1)>0)? f(i-1)+a[i]: a[i];
而最佳解是 f(i)中最大者或 0，
因此先設 opt=0，然後只要 i 由小往大算過去，每次檢查是否更新 opt

注意:上述算式，右邊是 f(i-1)左邊 f(i)，此種遞迴由小往大算不需 recursive call，此即是 DP

Preprocessing 解法

左端在 i 的最大區間和=

left(i)=

max{sum[j,i]: j<=i}

=prefix_sum(i)-min{prefix_sum(j-1): j<=i}

先求出所有點的 prefix sum

在求出所有 prefix_sum 的 prefix_min

```
m=0;  // opt
pre[0]=0;
for (i=1;i<=n;i++) pre[i]=pre[i-1]+a[i];
pm[0]=0;
for (i=1;i<=n;i++) pm[i]=(pm[i-1]<pre[i])? pm[i-1]: pre[i];
for (i=1;i<=n;i++) {
        k=pre[i]-pm[i-1]; // left(i)
        if (k>m) m=k;
}
```