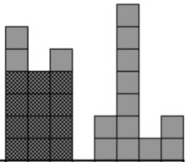
Problem 9. Architects 'R Us [15 points]

You are assisting Prof. Gehry with designing the shape of a new room in the Stata Center. The professor has given you n columns, each of the same unit thickness, but with different heights: $A[1], A[2], \ldots, A[n]$. He asks you to permute the columns in a line to define the shape of the room. To make matters difficult, MIT wants to be able to hang a large rectangular picture on the columns. If j consecutive columns in your order all have a height of at least k, then we can hang a rectangle of size $j \cdot k$.

The example below contains 3 consecutive columns with heights of at least 4, so we can hang a rectangle of area 12 on the first three columns.



i	1	2	3	4	5	6	7	8
A[i]	6	4	5	0	2	7	1	2

(a) Give an efficient algorithm to find the largest area of a hangable rectangle for the *initial* order $A[1], A[2], \ldots, A[n]$ of columns.

問題: 給定 $A[1,\ldots,n]$ 其中 $A[i]$ 代表的是位置 i 的高度 且 $A[i] = 0,\ldots,n-1$, $\forall i$

求 這 $A[1,\ldots,n]$ 構成之 structure 中 最大之 rectangle area 為多少　　$A[i] \neq A[j]$, $\forall i \neq j$

Algorithm: 共做 n 輪, 每一輪 做如下操作:

(1) 設為第 i 輪, 計算從 $A[i]$ 向左右擴展可得最大Area, 記錄在 $B[i]$ $\Rightarrow O(n)$

(2) 做完 n 輪後, $\max_{1 \le i \le n} \{B[i]\}$ 為所求

∴ 為 $O(n^2)$

DP Algorithm: 令 $B[i,j]$ 為 $A[i,\ldots,j]$ 之最小值

$\Rightarrow B[i,j] = \min\{B[i,j-1], A[j]\}$

∵ ① $A[j]$ 為最小值

② $A[j]$ 不為最小值

∴ 計算 $B[1,\ldots,n, 1,\ldots,n]$ 需 $O(n^2) \cdot O(1) = O(n^2)$

之後, 再計算: $\max_{1 \le i \le n, 1 \le j \le n} \{B[i,j] \cdot (j-i+1)\} \Rightarrow O(n^2)$

∴ 為 $O(n^2)$

Solution: The best algorithms we know run in $O(n^2)$ time.

Way 1

The simplest $O(n^2)$ algorithm is the following. The biggest rectangle is bounded on top by some column. Guess that column i. So the height of the rectangle is $A[i]$. Now walk left until reaching a column of height $< A[i]$, and similarly walk to the right. Count the number k of traversed columns, and multiply by $A[i]$.

Way 2

Another $O(n^2)$ algorithm is the following. Define $m[i, j]$ to be the minimum of the interval $A[i], \ldots, A[j]$. Then $m[i, j] = \min\{m[i, j-1], A[j]\}$, so by memoization, we can compute all $m[i, j]$'s in $O(n^2)$ time. Now the solution is $\max\{m[i, j] \cdot (j - i + 1) : i \leq j\}$, which takes $O(n^2)$ time to compute given the $m[i, j]$'s.

Way 3

The easy brute-force algorithm already runs in $O(n^3)$ time (and was worth a base value of 4–5 out of 8 points for this part). Just use the computation above, but without memoizing the $m[i, j]$'s, so each takes $O(n)$ to compute, and we use $O(n^2)$ of them.

Way 4

An $O(nh)$ algorithm, where $h = \max_i A[i]$, was worth a base value of 6 out of 8 points. We define one subproblem per (x, y) coordinate: $R(x, y)$ is the maximum possible area of a rectangle whose upper-right corner is at (x, y). There are $O(nh)$

透過左右調動column來maximize面積

(b) Devise an efficient algorithm to permute the columns into an order that maximizes the area of a hangable rectangle.

三種情況都可以
1. Increasing Order
2. Decreasing Order
3. Maximum In Middle

Algorithm: 只需排序 $A[1,\ldots,n]$ 即可，排序後的 $A[1,\ldots,n]$ 以可得到最大的 rectangular area

Correctness: 設 k 為該 maximum rectangular area 的高，則 $\exists A[i] = k$, for some i
又在 sorted array 中在 k 右邊的 element 都大於 k，以可和這些數建出 max area.

Step 1 :
把column順序放好
O(nlgn)

Solution: The intended algorithm is to sort the columns in decreasing order, e.g., using merge sort in $O(n \lg n)$ time. This works because, if the correct height of a rectangle is k, then at best it can involve all columns with height $\geq k$, and these are consecutive in the sorted order. In fact, increasing order works just as well, as does a strange order (suggested by several students) of putting the maximum in the middle, then repeatedly placing the next smaller column alternately between the left and right sides of the construction so far.

Step 2 :
計算面積
O(n)

We can compute the hangable rectangle in $O(n)$ additional time, though this was not necessary to receive full credit. For each prefix $B[1 \cdots i]$ of the sorted array, we'd like to compute $(\min B[1 \cdot i]) \cdot i$, and take the maximum over all i. But $\min B[1 \cdot i] = B[i]$, so this actually takes constant time per choice of i, for a total cost of $O(n)$ time.

2 out of 7 points were removed for lack of justification of sorting. 1 out of 7 points was removed for using counting (or radix) sort, which is not necessarily efficient given the setup of the problem.