CSCI3160: Regular Exercise Set 2

Prepared by Yufei Tao

Problem 1 (Faster Algorithm for Finding the Number of Crossing Inversions). Let S_1 and S_2 be two disjoint sets of n integers. Assume that S_1 is stored in an array A_1, and S_2 in an array A_2. Both A_1 and A_2 are sorted in ascending order. Design an algorithm to find the number of such pairs (a, b) satisfying all of the following conditions: (i) $a \in S_1$, (ii) $b \in S_2$, and (iii) $a > b$. Your algorithm must finish in $O(n)$ time (we gave an $O(n \log n)$-time algorithm in the class).

Solution. Merge A_1 and A_2 into one sorted list A, which takes $O(n)$ time. Scan the elements of A in ascending order. In the meantime, maintain the number n_2 of elements that (i) originate from A_2, and (ii) have already been scanned so far: this can be done by setting n_2 to 0 at the beginning, and increment it each time an element originating from A_2 is scanned. Furthermore, also maintain a counter c as follows: $c = 0$ at the beginning; every time an element a originating from A_1 is seen, c is increased by the current value of n_2. The final c at the end of the algorithm is the answer we are looking for.

Problem 2. Give an $O(n \log n)$-time algorithm to solve the dominance counting problem discussed in the class. (Hint: Require the $n/2$ points on each of side of the split line to be sorted after recursion.)

Solution. Let P be the input set of points. Recall that, as discussed in the class, our algorithm divides P into two halves P_1 and P_2 using a vertical line ℓ, and then recurse on P_1 and P_2 respectively. The first change we make to the algorithm is to ensure that, when the recursion on P_1 and P_2 ends, the points of P_1 and P_2 have been sorted by y-coordinate. Now it remains to find, for each point $p_2 \in P_2$, the number of points $p_1 \in P_1$ that are dominated by p_2. Next we show that this can be done in $O(n)$ time, which makes the total running time $O(n \log n)$.

In $O(n)$ time, merge P_1 and P_2 into one sorted list P, where the points are sorted in ascending order by y-coordinate. Scan P. In the meantime, maintain the number n_1 of points that (i) originate from P_1, and (ii) have already been scanned so far. Every time a point p_2 originating from P_2 is seen, the number of points $p_1 \in P_1$ dominated by p_2 is precisely the current value of n_1.

Problem 3 (Section 4.1 of the Textbook). Let A be an array of n integers (A is not necessarily sorted). Each integer in A may be positive or negative. Given i, j satisfying $1 \le i \le j \le n$, define *sub-array* $A[i : j]$ as the sequence $(A[i], A[i+1], ..., A[j])$, and the *weight* of $A[i : j]$ as $A[i] + A[i+1] + ... + A[j]$. For example, consider $A = (13, -3, -25, 20, -3, -16, -23, 18)$; $A[1 : 4]$ has weight 5, while $A[2 : 4]$ has weight -8.

1. Give an algorithm to find a sub-array of with the largest weight, among all sub-arrays $A[i : j]$ with $j = n$. Your algorithm must finish in $O(n)$ time.
2. Give an algorithm to find a sub-array with the largest weight in $O(n \log n)$ time (among *all* the possible sub-arrays).

Solution. Subproblem 1: Scan the elements of A from $A[n]$ to $A[1]$. At any time, maintain the sum s of the elements already scanned: at the beginning $s = 0$; after scanning an element $A[i]$, add

$A[i]$ to s. Every time we finish doing so for element $A[i]$, the current value s is precisely the weight of $A[i:n]$. In this way, we obtain the weights of all sub-arrays $A[n:n]$, $A[n-1:n]$, ..., $A[1:n]$ (in this order) in $O(n)$ time. The maximum weight can then be found easily.

Subproblem 2: Break A into two halves: array A_1 which contains the first $\lceil n/2 \rceil$ elements, and array A_2 which contains the rest. Recursively, find the sub-array of A_1 with the largest weight, and then the sub-array of A_2 with the largest weight. It remains to consider the "crossing" sub-arrays $A[i:j]$ where $i \le \lceil n/2 \rceil$ and $j > \lceil n/2 \rceil$. In particular, we want to find the "best" crossing sub-array, i.e., the one with the maximum weight. Then, the sub-array to output can be decided easily from the three sub-arrays aforementioned.

We say that a sub-array $A_1[i:j]$ is *right grounded* if $j = \lceil n/2 \rceil$, and a sub-array $A_2[i:j]$ is *left grounded* if $i = 1$. A crucial observation is that the "best" crossing sub-array must be the concatenation of

- the right grounded sub-array in A_1 with the maximum weight, and
- the left grounded sub-array in A_2 with the maximum weight.

From Subproblem 1, we know that each of the above two grounded sub-arrays can be found in $O(n)$ time.

Therefore, if $f(n)$ is the time of solving the problem on an array of length n, it holds that $f(n) \le 2 \cdot f(\lceil n/2 \rceil) + O(n)$, which gives $f(n) = O(n \log n)$.

Problem 4. In the class, we explained how to multiply two $n \times n$ matrices in $O(n^{2.81})$ time when n is a power of 2. Explain how to ensure the running time for any value of n.

Solution. If n is not a power of 2, let m be the smallest power of 2 that is larger than n. If A, B are the $n \times n$ input matrices, obtain an $m \times m$ matrix A' by padding $m - n$ dummy rows and columns to A containing only 0 values, and similarly, an $m \times m$ matrix B' from B. Calculate $A'B'$ in $O(m^{2.81}) = O((2n)^{2.81}) = O(n^{2.81})$ time. Then, the matrix AB can be obtained by discarding the last $m - n$ rows and columns from the matrix $A'B'$.