Inversion Counting

(needed only for coding)

Arnab Ganguly, Assistant Professor
Department of Computer Science, University of Wisconsin – Whitewater
Theory of Algorithms (CS 433)

1 Problem Definition and Naive Approach

Problem. Given an (unsorted) array $A[\,]$ of n integers, find out how many ordered pair of indices $\langle i, j\rangle$ exist such that $i < j$ and $A[i] > A[j]$. You do not need to find the indexes (just the count).

For example, if the array is $[100, 150, 50, 300, 15, 25]$, then there are 10 pairs of indexes:

- $\langle 0, 2\rangle; \langle 0, 4\rangle; \langle 0, 5\rangle$
- $\langle 1, 2\rangle; \langle 1, 4\rangle; \langle 1, 5\rangle$
- $\langle 2, 4\rangle; \langle 2, 5\rangle$
- $\langle 3, 4\rangle; \langle 3, 5\rangle$

The brute-force approach to this problem is just to use a nested for-loop on i and j. The outer loop i runs from 0 to $n-2$, and the inner loop j runs from $i+1$ to $n-1$. Within the inner loop, increment a counter if $A[i] > A[j]$. Finally, return the counter value. The complexity is $\Theta(n^2)$.

2 A Merge-Sort Kind of Approach

A faster approach is using merge-sort. Based on merge-sort division of arrays, note that the total number of inversions is the sum of the following:

- number of inversions in the left half of the array (i.e., $left \le i < j \le mid$ and $A[i] > A[j]$)
- number of inversions in the right half of the array (i.e., $mid < i < j \le right$ and $A[i] > A[j]$)
- the number of crossover inversions (i.e., $i \le mid$, $j > mid$, and $A[i] > A[j]$)

The first two can be easily computed by the recursion calls. To compute the third part, note that whenever you are merging the two sorted halves of the array, if you copy a number from the right half (i.e., index j), then all the remaining indexes in the left half (i.e., $mid - i + 1$) contribute to the inversion count. Thus, in each recursion level, you first (recursively) compute the number of inversions from left and right halves, then the number of crossover inversions, and finally return the sum. Obviously, the base-case is when you have one element, which does not contribute to the inversion count (and hence you return zero).

Since this is essentially merge-sort, with a couple of minor changes (return type of the function is an integer and a few extra statements that are needed to compute the inversion count in each level), the complexity is $\Theta(n \log n)$.