UC Davis CS 122 Fall 2010, Gusfield: Counting Inversions in a Permutation

In the class discussion of the method to count the number of inversions in a permutation (discussed in Section 5.3 of the Kleinberg and Tardos book), we left out two parts of the proof of correctness: proving that every inversion gets counted at least once, and proving that no inversion gets counted twice. The proof in the book is also somewhat deficient on these points, so here is a complete proof for those claims.

Let Z denote the original list of all the numbers.

I. In algorithm Merge-and-Count, when the algorithm compares a_i, the top element of list A, to b_j, the top element of list B, and finds that $b_j < a_i$, it adds $|A|$ inversions to its count of inversions. We claim that at that point every element in the current A is indeed involved in an inversion pair with b_j. To see this, note that every element in the current A is larger or equal to element a_i (because A is in sorted order, smaller to larger). Since a_i is larger than b_j, every element in the current A is larger than b_j. Note also that every element in A is before (to the left of) b in Z. This follows inductively by the fact that when a list L is divided into two sublists for the recursive calls in Sort-and-Count(L), the two sublists are the left and right parts of L respectively. So, in Merge-and-Count(A,B) all the elements in A are to the left of all the elements in B, in the original list Z. So, when $a_i > b_j$, and the Merge-and-Count algorithm adds $|A|$ to the count of inversions that cross from A to B, it is because there are that many inversion pairs involving b_j as the second (i.e, smaller) element. This is essentially the part of the proof of correctness that we did in class.

II. Merge-and-Count counts every inversion at least once. Suppose $Z(i) > Z(j)$, i.e., there is an inversion in the original list Z between the elements in positions i and j. For concreteness, suppose $Z(i) = a$ and $Z(j) = b$. The calls to Sort-and-Count(L) successively divide Z into half-size lists A and B, where all elements in A are to the left of all elements in B, and elements in $A \cup B$ come from a consecutive interval in Z. So, at some point in algorithm Merge-and-Count, element a will be in the current A and element b will be in the current B. Merge-and-Count cannot exhaust all of A before b becomes the top of B, because $a > b$. So consider the actions of Merge-and-Count when b is at the top of B. At some point when b is the top of B and a is still in A, b will be compared to an element $a' \in A$, where $a' > b$ and $a' \leq a$. At that point, b is removed, and $|A|$ is added to the count accumulated in

Merge-and-Count. So, at that point, the inversion pair a, b contributes one to the count accumulated in the call of Merge-and-Count for A, B. Note that any element a'' to the left of a' in A is smaller than b (and as argued above, a'' is to the left of b in Z), so a'', b is not an inversion. Hence, every inversion pair gets counted at least once in the algorithm.

III. No inversion pair gets counted more than once. An inversion pair a, b can only contribute to a count when $a \in A$ and $b \in B$ in some invocation of Merge-and-Count. Since A and B become merged during that invocation, and stay together in every future invocations, the pair a, b can contribute to the count in exactly one invocation of Merge-and-Count. During that invocation when an inversion pair a, b does contribute to the count (as explained above), b is immediately removed from B, and so the pair a, b does not get counted again in that invocation of Merge-and-Count. Hence, no inversion gets counted more than once.