CS 577: Introduction to Algorithms 10/29/15

Midterm Grading Comments

Instructor: Dieter van Melkebeek TAs: Kevin Kowalski, Andrew Morgan, Bryce Sandlund

Problem 1 [10 points]

Grader: Kevin

Rubric:

- [6 points] for part (a)
  - [3 points] for stating the following invariants implicitly or explicitly
    - [0.5 points] for something that forces $\ell = r + 1$ when the loop ends, when combined with the loop condition
    - [1.5 points] for stating that $A[1, \ell - 1]$ only contains elements $< m$ and $A[r + 1, n]$ only contains elements $\geq m$
    - [1 point] for stating that w_L contains the total weight of all elements in $A[1, \ell - 1]$
  - [2 points] for proofs of invariants
    - [+1 point] if the proofs are especially detailed
  - [1 point] for using invariants to prove partial correctness
- [2 points] for a counterexample to part (b), with explanation
- [2 points] for a counterexample to part (c), with explanation

Comments: In part (a), some students did not realize that in order to show that $A[1, r]$ contains all the elements that are less than m, it is necessary to show both that $A[1, r]$ contains no elements greater than or equal to m and that $A[r + 1, n]$ contains no elements less than m.

A number of students stated that the procedure is both partially correct and terminates in parts (b) and (c), even though the problem description states that the procedure is incorrect. In retrospect, we probably should have emphasized the fact that the procedure is incorrect more heavily.

One common mistake for part (b) was reusing the counterexample for part (c), with the claim that because the procedure doesn't terminate on this particular input, it must not be partially correct. Partial correctness can still hold even if a procedure does not terminate on any inputs.

Problem 2 [8 points]

Grader: Bryce

Rubric:
- [1 point] $O(n \log n)$ algorithm
- [1 point] divide and conquer
- [4 points] correctness
  - [1 point] recurse on children (children have the same problem)
  - [1 point] return value correct (count inversions left + right + cross, or count inversions afterwards)
  - [2 points] swap condition
- [1 point] runtime argument
- [1 point] correctness argument

Comments: The most common mistake was incorrectly determining when to make a swap at an internal node. The two most prevalent incorrect ideas were to base it off a min or max of the left or right subtree, or to base it off a sum of the leaves in the left or right subtree. All three strategies can actually be broken by the following leaf nodes, in order:

$$1, 5, 6, 7,\ \ 2, 3, 4, 20$$

A sum, comparison of the minimums, or comparison of the maximums would leave the root's children as is, for an inversion count of 9, while the optimal choice is to swap the root's children, for an inversion count of 7. There were also a few solutions that looked at the maximum in the left and minimum in the right, which is also broken with this example.

Additionally, people who employed any of these strategies sometimes said their solution was $O(n \log n)$, but in fact $O(1)$ work is done at each node, and there are n nodes, so the complexity is actually $O(n)$. I was lenient on this, however, since it is true that at most $O(n)$ work is being done at every level.

A number of solutions got very close to the correct approach, but missed that you need to merge the lists of leaves into sorted order as you return up the tree, and either called $O(n \log n)$ counting inversions at every level or asserted incorrectly that counting could be done in $O(n)$ at every level. These solutions got somewhere in between 6 and 8 points, depending on the writeup and whether the fact the overall solution was $O(n \log^2 n)$ was caught.

Problem 3 [12 points]

Grader: Drew

Rubric:

- [6 points] for part (a)
  - [1 point] for implicitly finding the correct characterization of the optimal solution, *i.e.*, pairing the largest n with the smallest n.
  - [3 point] for using median-finding to get a linear-time solution. You didn't get much credit here if you used selection but didn't get a linear-time algorithm in the end. If you used bucket sort, or a $\Theta(n\log(n))$ solution that was essentially sorting, you got 1 point here. If you did anything worse, it was difficult to justify giving any points.
  - [2 points] for correctness and efficiency. There was some subjectivity here, but mostly you got some credit for trying, and some credit for succeeding (if you did succeed). If you accurately computed your running time (even if it was slow), you got at least one point for that.
- [6 points] for part (b)
  - [1 point] for pairing elements from the larger n with elements from the smaller n.
  - [1 point] for construction a maximal prefix/suffix-style solution. You didn't have to find the maximum length, but if you tried to find a maximal length one, then you got this point.
  - [1 point] if your solution recognized the need to "interleave pairs" in the optimal solution, *i.e.*, you gave a solution with pairs $(a, b), (c, d)$ with $a < c$ and $b < d$.
  - [1 point] if your algorithm ran in $O(n\log(n))$ time. If you were copying from part (a) and bucket sorted here, you did not get this point.
  - [2 points] for correctness and efficiency. Again, some subjectivity here, but you again got credit for trying (if you did), and credit for succeeding (if you did).

Comments: The most common problem was inadequate proofs in both parts. The grading was generous in this regard, but a large number of people didn't succed in arguing the key idea as to *why* the optimal solutions look the way they do. In part (b) this is largely because of inadequate characterizations of the solution (and subsequent incorrect algorithms). For part (a), most people characterized the solution correctly, but the argument was often missing or unclear or incomplete, for example because of only considering pairs (i, j) where $a_i < a_j$. For the most part, this didn't cost that many points, but it did have a decent impact on those who left proofs out of both parts entirely.

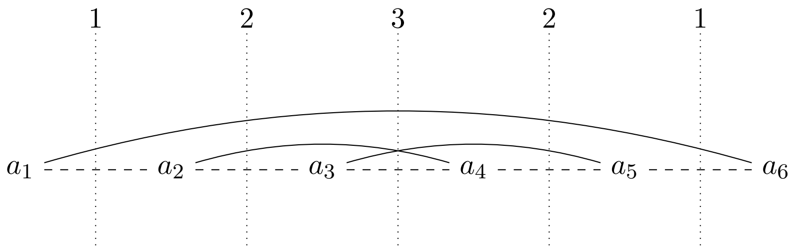
Solutions to (a) were mostly in two camps: bucket sorting and median-finding. Median-finding was the intended solution. Bucket sorters lost points for forgetting that the running time of bucket sort depends on the range of the integers, not just how many there are. In this problem, the range could have been arbitrarily big, and hence bucket sort could have been arbitrarily slow. An common wrong solution formed pairs only out of adjacent elements in the input sequence. Another common solution was sorting and admitting to an $O(n\log(n))$ solution, or just repeatedly matching-then-removing the maximum and minimum elements, which results in a $\Theta(n^2)$ solution.

As for (b), many people gave (variations on) one of two general ideas: repeatedly matching high/low until the constraint breaks (this breaks on $1, 2, 3, 4$), or matching the highest element with the largest element less than half the high element, and continuing until this isn't defined (this breaks on $3, 4, 5, 8$). There were some other incorrect solutions, too, but those two were the most common ones.

One student gave a nice pictorial proof of the characterization in (a). The basic idea was to notice that, in any optimal solution (and assuming $a_1 < a_2 < \ldots < a_{2n}$), the value can be written

$$1 \cdot [a_2 - a_1] + 2 \cdot [a_3 - a_2] + \cdots + n \cdot [a_{n+1} - a_n] + \cdots + 2 \cdot [a_{2n-1} - a_{2n-2}] + 1 \cdot [a_{2n} - a_{2n-1}]$$

corresponding to the number of arcs over the intervals between the points in the example diagram here:



The student then showed that, when you write out any other pair as a sum of the form

$$\sum_{i=1}^{2n-1} \alpha_i (a_{i+1} - a_i)$$

the coefficients α_i are at most the coefficients appearing above. Intuitively this is obvious: just look at how many lines could possibly be drawn over each interval in the diagram while preserving the "disjoint pairs" property.