HW 7

- give a complete statement of the algorithm
  - identify the base case (e.g. n=1) *and* state what the answer is in that case
  - for easy split, explain how to do the combine step with enough detail to understand...
    - ...how to do it (if there's not an obvious brute force algorithm)
      - e.g. "count the inversions involving an element from the first half and an element from the second half" → since the definition of inversion is a pair (*a*,*b*) where *a*>*b*, the obvious brute force algorithm would be to check all pairs of elements (*a*,*b*) where *a* is in the first half and *b* is in the second half
    - ...how to achieve the claimed/desired running time (if better than brute force)
      - e.g. checking all pairs of elements (*a*,*b*) where *a* is in the first half and *b* is in the second half involves n/2 x n/2 pairs → $O(n^2)$ … so if you claim O(n) for this step, you need to explain how
  - (for easy merge, do the same for the split step)
  - it's OK to expect the friends to do more, but this needs to be accounted for –
    - include what they do in the definition of the subproblem (task, output)
    - include how they do it in the main case steps

HW 7

- account for special cases – watch for implicit assumptions
  - e.g. if a task involves finding *the* min or *the* max … what if there's not a unique value? (i.e. duplicate values)
    - is that possible in a legal input? if so, need to address it

HW 7

- give the steps of the process from class, not just a statement of the algorithm or a narrative account of your reasoning process
  - establish the problem
    - specifications
    - examples
  - identify avenues of attack
    - targets
    - patterns
  - define the algorithm
    - size
    - generalize / define subproblems
    - base case(s)
    - main case
    - top level: initial subproblem, setup, wrapup
    - special cases
    - algorithm
  - show termination and correctness
    - termination: making progress, the end is reached
    - correctness: establish the base case(s), show the main case, final answer
  - determine efficiency
    - implementation
    - time and space
    - room for improvement
- for process issues that occurred in more than one of the problems, the issue was generally only pointed out once
  - the same comments were not repeated for all three problems

HW 7

- specifications
  - be sure to actually state the problem!
  - explicitly identify the input and output – don't just leave them stated as part of the task

HW 7

- avenues of attack – targets
  - give the obvious brute force algorithm and its running time
    - based directly on the definition of the task – you're not trying to be efficient, just have a strategy that is clearly correct
      - e.g. an inversion is a pair of values (*a*,*b*) where *a*>*b* → obvious brute force is to enumerate all the pairs and check each
    - the point of many divide-and-conquer algorithms is to improve on the obvious brute force, so identifying that running time lets you know if you've succeeded...or if you need to improve the efficiency of your divide-and-conquer
- avenues of attack – patterns
  - consider both easy split and easy merge patterns
    - for each, is it applicable the this problem?
  - goal is to outline the framework for the algorithm, not to figure out the whole solution
    - easy split: split input in half, friends solve, then we figure out how to combine those results
    - easy merge: friends return the two halves of the solution, so we have to figure out how to split the input to make that possible

HW 7

- defining the subproblems
  - generalizing the problem means going from solving the problem for the whole input to solving the problem for a portion of the input
    - should be reflected in the statement of the subproblem's task and in its input (include how the portion is specified)
  - it's OK for the friends to do/return more than in the original problem, but this needs to be included in the subproblem definition (task, output)
- "setup" and "wrapup" refer to anything done before and after the first recursive call (for the initial subproblem)
  - one time things – *not* what happens in the main case before and after the recursive calls

HW 7

- *priority #2* –
  - the score from the resubmit for #2 will be taken as the score for all problems originally submitted (assuming improvement)
  - encouraged to resubmit other problems – for practice, and to get the +1 for a resubmit with substantive improvements for those problems
  - submit any missing the first time for credit