When describing an algorithm or data structure, there is always a balance: You need enough details to avoid ambiguity, but too many details obscure understanding. A few specific comments stemming from the homework —

- *Overview before details.* Give an overview — what is the central idea of the algorithm or behind the design? — before launching into a more detailed description of the specifics, especially if you are expressing those details at the level of pseudocode.

- *Describe algorithms, not code.* A good description explains the idea and steps; describing code is a more cumbersome restatement of the program logic that only obscures the algorithm even more.

  For example, consider inserting an element into a sorted array.

  Describing the algorithm:

  > To insert x into a sorted array:
  >
  > 1. Starting from the end of the array, find the position where x belongs in the sorted order.
  > 2. Shift all elements that are larger than x one position to the right.
  > 3. Insert x into the empty position created.

  Describing the code:

  > Start with $i = n - 1$. While i is greater than or equal to 0 and the element at $A[i]$ is larger than x, copy the element at $A[i]$ into position $A[i + 1]$ and decrement i. After the loop finishes, store x in position $A[i + 1]$ and increase n by 1.

  If you need that level of detail, use pseudocode instead:

```
i = n - 1
while i >= 0 and A[i] > x do
  A[i+1] = A[i]
  i--
A[i+1] = x
n++
```

  (Though be sure to include an explanation of the idea behind the pseudocode to provide context — overview before details.)

When a problem asks for an algorithm or data structure to achieve a certain big-Oh (running time or space), be sure to explicitly address why your solution achieves that big-Oh.

For #2, what about duplicate elements? Duplicates are a good typical special case to keep in mind when dealing with collections of things.

#3 also asked you to assess the amortized time of the "bad sequence" under both the shrink-when-half-full scheme and the better scheme from (b) and a sequence of n deletions under the better scheme from (b). The intent here was to show the math — write out the cost (including resizing) for each operation in the sequence of n steps, write that as a sum (with Σ), determine the big-Oh for that sum (using the sums table if needed), and divide by n to determine the average cost per operation. Saying that there were $O(1)$ or $O(n)$ operations resize or non-resize operations and assessing the total work and amortized cost from there was fine, but claiming that the better scheme was constant amortized time because there were a lot fewer resizes wasn't sufficient.

For #4, the whole process needs to be $\Theta(1)$, even special cases like deleting the tail. The tail is problematic because there isn't a node after to actually remove, but the footnote on page 79 that the problem refers to addresses this case. The comment "OK if sentinel" reflects handling that is appropriate using the permanent sentinel implementation mentioned in the footnote, but where there wasn't any direct mention of that being the case in the solution.