Andrew Clayton

CSCI330 Programming Language Concepts

Homework 3 Written Answers

Question 2, parts 1-4:

2.1: Define a recursive function to do the partitioning. The basic partition strategy is to maintain the processed items as two lists, and at each stage take the next two unprocessed items and put one in each list (what if there are zero or one item in the unprocessed list?).

If there are only one or zero items in the unprocessed list, then this is considered a sorted list (the items, or lack of them, will be in the right order).

2.2 The result of partitioning is two halves, As per mergesort, what should we do to each half?

After we partition and get each half, then we call mergesort on each half. This sorts each half. Once the halves are sorted, we merge the sorted halves back together.

2.3 Define a merge function that merges two sorted lists. Now the processed part is one sorted list, and the unprocessed part is two sorted lists. We need to pick the first item from one of the two unprocessed lists and append it to processed part. What are the end conditions?

Overall, the end conditions are that we have iterated over all of the elements of the two sorted (unprocessed) lists.

In terms of while loops, we have 3: the end conditions for the first are once we have iterated through all of the elements of either of the two unprocessed sorted links. Then, we have two loops for each unprocessed links, ensuring that we have processed both of them all of the way. This is because the first while loop breaks once EITHER of the lists have been fully processed, meaning one of them (we don't know which) still has elements to be processed.

Question 4, parts a-c:

4.a). While insertion sort is in progress, we track 2 lists: the sorted items and the unsorted items. What should these look like when the process starts and when the process ends? Can we use that information to decide termination?

When the process starts, the list of sorted items should be empty (there are no items sorted yet). Consequently, the unsorted items list will contain all the items from the input.

When the process ends, this is flipped – the sorted list should contain all the elements, and the unsorted list none of them. We can indeed use this information to decide termination – once the unsorted items list is empty, this means our work is done (all elements are sorted).

4.b) In each pass we start with two lists. At the end of the pass, we would have moved one more item from the sorted to the unsorted list. When is this process trivially accomplished? What will the recursive call look like?

This process is trivially accomplished when the list of unsorted items is empty. This is our base case in recursion.

The recursion call then would have these criteria:

- Input: unsorted items list.
- Base case: once unsorted items list is empty
    - In which case, we return the then complete sorted items list.
- Operations in each recursive step:
    - Take first item from unsorted list
    - Recursively call insertion function to place this item in the correct position in the sorted list
    - Make a recursive call to the main insertion sort function with the rest of the items in the unsorted list (excluding the item we just sorted)
    - This will return the sorted version of the remaining unsorted items.

4.c) Moving requires an operation that can insert an item into a sorted list in ascending order. This can be done easily using the predefined LAST function in Common Lisp (look for this in the LISP text). Again, to represent this insertion process, we need to track the items that have been examined, the item to be inserted, and the items yet to be examined. There are two ways in which the process can terminate – what are they?

The two ways this process can terminate:

- We reach the end of the sorted items list (items that have been examined) – meaning that the current item is the largest item examined yet.
- We find an element in the "items yet to be examined" list that is greater than or equal to the item to be inserted – this is the correct position to insert the current item. The current item should be inserted before this element.