Jonathan Simonin

Professor Wei

CSE 2100

6 December 2016

Homework #3

R-12.2 In the merge-sort tree shown in Figures 12.2 through12.4, some edges are drawn as arrows. What is the meaning of a downward arrow? How about an upward arrow?

The downward arrows represent the node that the branch is going down. When it shows the downward arrow, it goes down further in the tree to the next smallest node, and then with and arrow up it shows that whatever is held in the node at the base of the arrow is getting sent to the tip of the arrow (the next closest node to the top of the tree). Essentially, it shows the progression of the sort method going up and down levels of the tree.

R-12.4 Is our array-based implementation of merge-sort given in Section 12.1.2 stable? Explain why or why not.

No it is not stable. If the if statement didn't just return, then it would properly sort the array and send it back, so when the first set of n from S1 gets to less than 2, the method returns and ends execution without returning the entire sorted array. The last line that says merge(…); merges the unsorted arrays, since the recursive calls never makes a new array that is the new sorted array list.

R-12.5 Is our linked-list-based implementation of merge-sort (Code Fragment 12.3) stable? Explain why or why not.

As with the previous question, the code will run but is not stable due to the way the return statement works.

R-12.9 Suppose we modify the deterministic version of the quick-sort algorithm so that, instead of selecting the last element in an n-element sequence as the pivot, we choose the element at index ⌊n/2⌋. What is the running time of this version of quick-sort on a sequence that is already sorted?

The middle element of a sorted array will result in an already balanced split, which is optimal. The running time of this would be O(nlogn).

R-12.13 If the outermost while loop of our implementation of quickSortInPlace (line 9 of Code Fragment 12.6) were changed to use condition left < right, instead of condition left <= right,

there would be a flaw. Explain the flaw and give a specific input sequence on which such an implementation fails.

R-12.14 If the conditional at line 14 of our quickSortInPlace implementation of Code Fragment 12.6 were changed to use condition left < right, instead of condition left <= right, there would be a flaw. Explain the flaw and give a specific input sequence on which such an implementation fails.

The flaw would be that it would not scan all elements, as it would not include any values equal to right. 5 6 4 8 5 9 8 7 2 4 8 5, in this sequence, right = 8,and that means that any value that the scanner finds that is equal to 8, does not get caught in the sequence and it will not be added to the right.

R-12.18 Is the bucket-sort algorithm in-place? Why or why not?

The bucket-sort algorithm is not in place since it creates a new pointer B that grows as the same size as S, but not with S, so you need extra space to compensate for this. Had the pointer B been a temporary list that would be used for a recursive method, it would be in-place since it would not be holding up and more memory, but only temporarily. Another way to make it in-place is to get rid of B and make the algorithm only use S.

R-12.19 Describe a radix-sort method for lexicographically sorting a sequence S of triplets (k,l,m), where k, l, and m are integers in the range [0,N−1], for N ≥2. How could this scheme be extended to sequences of d-tuples (k1,k2,...,kd), where each ki is an integer in the range [0,N−1]?

You can iterate through the index I of all the tuples, and reverse traverse through the entire list. At each index you get to, sort the tuples using bucket sorting by using their i0th entry with N buckets.

R-12.20 Suppose S is a sequence of n values, each equal to 0 or 1. How long will it take to sort S with the merge-sort algorithm? What about quick-sort?

Merge-sort will take O(nlogn) time, and with quick-sort it will take O(n) time.

R-12.21 Suppose S is a sequence of n values, each equal to 0 or 1. How long will it take to sort S stably with the bucket-sort algorithm?

Depending on the size of S, it will take O(n) or O(nlogn)

R-12.25 Show that the worst-case running time of quick-select on an n-element sequence is Ω(n2).

If the pivot point is chosen to be at the largest element in the subsequence of n, then it will have to go through the worst case run time of n^2.