COSCI 21a, Assignment W7

Directions: To receive full credit:

• Place your name at the top of each page.
• Start each problem on a new page.

1. Starting with an empty binary heap using a full-tree implementation:
   A. Insert the integers 17, 3, 4, 7, 2, 9, 11, 5, 19, 14, 20, 18 in that order, and after each step, show the heap array.
   B. Do DELETEMINs until the heap is empty, and show the array after each step.

2. Suppose that heap data structure using a full binary tree was represented with pointers rather than stored in an array as presented in class; that is, each vertex has a parent pointer, a left child pointer, and a right child pointer. And assume we are still using the same algorithm for INSERT and DELETEMIN, except now, at the low level, movement around the tree is via pointers rather than arithmetic on array indices (and we can no longer simply increment or decrement nextRB). Describe in English and give pseudo-code for how to maintain a pointer to the RB leaf; that is how to find the next or previous leaf in level order when an INSERT or DELETEMIN operation requires insertion or deletion of the RB leaf. Analyze the time used.

3. Recall the recursive Merge Sort algorithm presented in class, where the presentation was at a relatively high level that did not address how the two lists were represented. Suppose that the list of n elements to be sorted was in an array A[0] ... A[n−1], and we have available an additional array B[0] ... B[n−1] that can be used for temporary storage. Describe in English and give pseudo-code to implement Merge Sort so that the input is in A, the algorithm concludes with the sorted data in A, and only O(1) space is used in addition to the space for A and B and the space used “behind the scenes” to implement the recursion. Include an explanation of why the additional space used to implement the recursion is only O(log(n)), and hence, at least for large n, the only significant space used in addition to A is for B.

4. Consider the following simpler version of PARTITION for quick sort:

   function simplePARTITION(i, j)
   \[
x := i + 1
   y := j
   \]
   while \(x \leq y\) do
   \[
   \text{if } A[x] \leq A[i] \text{ then } x := x + 1
   \]
   \[
   \text{else if } A[y] > A[i] \text{ then } y := y – 1
   \]
   \[
   \text{else exchange } A[x] \text{ and } A[y]
   \]
   \[
   \text{end}
   \]
   \[
   \text{if } i < y \text{ then exchange } A[i] \text{ and } A[y]
   \]
   return \((y–1, y+1)\)
   end

   A. Explain why it correctly partitions.
   B. Explain why using this version of partitioning does not change the asymptotic running time of partitioning, and discuss how performance would compare in practice.