Exercise 0. When implementing heaps in C, using zero-based indexing for the underlying arrays, it’s necessary to adjust the definitions of Parent, Left, and Right to reflect the fact that the children of the node at array index \( i \) are at indices \( 2i + 1 \) and \( 2i + 2 \) (rather than \( 2i \) and \( 2i + 1 \)). In C, write macros `parent(i)`, `left_child(i)`, and `right_child(i)` to implement the adjusted definitions for these operations.

Exercise 1. Using the macros from the preceding exercise, write the Max-Heapify procedure in C, using (as recommended in exercise 6.2–5 of the text) an iterative control structure rather than recursion to “float” the value that violates the max-heap invariant “down” the branch on which it is located until the invariant is restored.

Exercise 2. Implement the Build-Max-Heap and Heapsort procedures in C. Use the latter to sort the array \( \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle \).

Exercise 3. The file `/home/stone/courses/algorithms/code/quicksort.c` contains an implementation of the quicksort algorithm in C, specialized to sorting arrays of `int` values. Determine whether the method that it uses is Lomuto’s Partition (page 171, Figure 7.1) or Hoare’s Hoare-Partition (page 185). Then develop an alternative implementation that uses the other partitioning method.

Exercise 4. Instrument both versions of quicksort so that each returns a count of the number of assignments to positions within the array it performs in the course of the sort. Then run both versions on a large array of random integers. Which version performs more data movements? Why?