Homework 3

1. Mergesort and merging
   (Assume we don’t use sentinels when merging two sorted arrays.)
   (a) What is the worst-case number of key comparisons done by (standard) MERGE on two sorted arrays, one of size 3 and one of size 4? What is the information theory lower bound for this problem?
   (b) What is the worst-case number of key comparisons that MERGESORT does when sorting \( n = 7 \) numbers? What is the information theory lower bound for sorting \( n = 7 \) numbers?

2. Quicksort
   Assume we use the HOARE-PARTITION procedure below (also given in class):
   \[
   \text{partition}(a,n) \quad // \quad \text{variant due to Hoare} //
   \]
   \[
   // \quad \text{partitions } a[1..n] \text{ around } a[1] = \text{pivot} //
   \]
   \[
   // \quad \text{output } j; \quad A_1 = a[1..j] \text{ and } A_2 = a[j+1..n] //
   \]

   1. \( x <- a[1] \)
   2. \( i <- 0 \)
   3. \( j <- n+1 \)
   4. repeat \( j <- j-1 \) until \( a[j] <= x \)
   5. repeat \( i <- i+1 \) until \( a[i] >= x \)
   6. if \( i < j \) swap \( a[i] \) and \( a[j] \) and go to 4
      else output \( j \) and halt
      //indices have crossed each other//
   // \( A_1 = a[1..j] \) and \( A_2 = a[j+1..n] //

   (a) Demonstrate the operation of HOARE-PARTITION on the array \[13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21\], showing the values of the array and auxiliary values after each iteration of lines 4–6.
   (b) What is the running time of QUICKSORT when all elements of the array have the same value?

3. Binary search trees, heaps, and sorting
   Consider the following argument for proving that any comparison-based algorithm for constructing a binary search tree from an arbitrary list of \( n \) elements takes \( \Omega(n \log n) \) time in the worst case.

   Let \( B \) be any comparison-based algorithm for constructing a binary search from an arbitrary list of \( n \) elements. Let \( T(n) \) be its worst-case running time. Let \( S \) be the following comparison-based sorting algorithm (that uses \( B \)); input array is \( a[1..n] \).
Alg. S

Step 1. Call B to construct a BST T on the keys in a[1..n].

Step 2. Do an inorder traversal of T to print the keys in sorted order.

Note that the worst-case running time of S is \( T(n) + \Theta(n) \); By the ITLB for sorting we know that the worst-case running time of S must be \( \Omega(n \log n) \), that is, \( T(n) + \Theta(n) = \Omega(n \log n) \); consequently, \( T(n) = \Omega(n \log n) - \Theta(n) = \Omega(n \log n) \), as required.

(a) Is the above argument correct? why or why not? If you think the argument is incorrect, can you revise it to a valid proof?

(b) Consider the design of an algorithm for printing out the keys of an \( n \)-node binary min-heap in sorted order. Prove or disprove: this task can be accomplished in \( O(n) \) time.

Other suggested exercises (for practice; you do not need to turn in)

4. Insertion Sort

Explain and illustrate how INSERTION-SORT works on the following array of \( n = 10 \) numbers: \( A = [6, 0, 5, 7, 2, 3, 9, 0, 2, 6] \). Show the array after each iteration of the algorithm. What are the best-case running time and the worst-case running time of INSERTION-SORT on an array with \( n \) elements?

5. Binary search trees and sorting

We can sort a given set of \( n \) numbers by first building a binary search tree containing these numbers (using INSERT repeatedly to insert the numbers one by one; recall, the version of INSERT we have studied does not do any re-balancing) and then printing the numbers by an in-order tree walk. What is the worst-case running time of this sorting algorithm?

6. Binary search trees: insertion and deletion

Is the operation of insertion “commutative” in the sense that inserting \( x \) and then \( y \) into a binary search tree yields the same tree as inserting \( y \) and then \( x \)? Argue why it is or give a counterexample. Then answer the same question for deletion. (Hint: As usual, first try to find counterexamples.)