Assignment 4
Computer Science 231
Fall 2016
Due: Start of Class on Tuesday, February 28, 2017

Reading. CLRS Chapters 7, pages 170 -- 190

CLRS Problem 2-2bcd. Bubblesort is a popular sorting algorithm. It works by repeatedly swapping adjacent elements that are out of order.

BUBBLESORT(A)
1 for i = 1 to A.length
2 for j = A.length downto i + 1

b. State precisely a loop invariant for the for loop in lines 2 - 4, and prove that this loop invariant holds. Your proof should use the structure of the loop invariant proof presented in chapter 2.

c. Using the termination condition of the loop invariant proved in part (b), state a loop invariant for the for loop in lines 1- 4 that will allow you to prove that the array is sorted upon termination of the algorithm. Your proof should use the structure of the loop invariant proof presented in chapter 2.

d. What is the worst-case running time of Bubblesort? How does it compare to the running time of insertion sort?

Exercise 4-2. Thus far we have studied six comparison-based algorithms for sorting an array A[1..n]: insertion sort, bubble sort, selection sort (an iterative version is included below), merge sort, heap sort, and quicksort. For each of these six algorithms, answer the following questions.

a. What is the worst-case running time of the algorithm? Justify your answer. As part of your answer, describe the structure of an input array on which the algorithm exhibits its worst-case running time.

b. What is the best-case running time of the algorithm? Justify your answer. As part of your answer, describe the structure of an input array on which the algorithm exhibits its best-case running time.

c. What is the average-case running time of the algorithm? Explain. (Note: In most cases, you should be able to either figure this out or quote a result from the book. However, the average case analyses of insertion sort are rather complex and I certainly do not expect you to derive a formal average-cases analysis from scratch. Instead, you may make use of the following assumption: In Insertion-Sort, assume
that Insert(A, i) requires i/2 comparisons in the average case. That is, on average, the element being inserted will fall in the middle of the segment A[1..i].)

d. It is often important when sorting to model the temporary space (i.e., memory) required (space in addition to the given array). A sorting algorithm is in-place if the amount of temporary space is constant. Is the algorithm in-place? Explain. (Note: you must consider not only the temporary variables/arrays required by the algorithm, but also the longest path on the recursive invocation tree. In other words, if z(n) recursive method invocations are open/executing at a given time, then there are z(n) frames on the stack, which require \(\Omega(z(n))\) space.)

e. A sorting algorithm is stable if it preserves the relative positions of equal elements. In other words, numbers with the same value appear in the output array in the same order as they do in the input array. Ties between two numbers are broken by the rule that whichever number appears first in the input array appears first in the output array. For example - If an alphabetized class list is sorted by grade, then a stable method will produce a list in which students with the same grade are still in alphabetical order. Is the algorithm stable? Explain.
Algorithms for use in Exercise 4-2.

Iterative Selection Sort.

Selection-Sort(A)
\[
\text{for } i = 1 \text{ to } A.\text{length}[A] - 1 \\
\text{exchange } A[i] \text{ with } A[\text{Min-Index}(A, i, A.\text{length})]
\]

Min-Index(A, lo, hi)
{Assume lo and hi are legal subscripts, and hi >= lo.}
\[
\text{min_index} = \lo \\
\text{for } i = \lo + 1 \text{ to } \hi \\
\quad \text{if less}(A[i], A[\text{min_index}]) \\
\quad \text{min-index} = i \\
\text{return min_index}
\]

Merge Algorithm for MergeSort.

Merge(A, p, q, r)
\[
\text{n} = q - p + 1 \\
\text{n} = r - q \\
\text{create arrays } L[1..n_1 + 1] \text{ and } R[1..n_2 + 1] \\
\text{for } i = 1 \text{ to } n_1 \\
\quad L[i] = A[p + i - 1] \\
\text{for } j = 1 \text{ to } n_2 \\
\quad R[j] = A[q + j] \\
L[n_1 + 1] = \infty \\
L[n_2 + 1] = \infty \\
i = 1 \\
j = 1 \\
\text{for } k = p \text{ to } r \\
\quad \text{if } L[i] \leq R[j] \\
\quad \quad A[k] = L[i] \\
\quad \quad i = i + 1 \\
\quad \text{else } A[k] = R[j] \\
\quad \quad j = j + 1
\]