**Mason Shepherd**

**Homework 9**

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? [5 points]

Ans:

O(*n*log*n*)

R-12.11 Suppose the method quickSortInPlace is executed on a sequence with duplicate elements. Prove that the algorithm still correctly sorts the input sequence. What happens in the partition step when there are elements equal to the pivot?

Ans:

They remain in the parent node along with the pivot element.

What is the running time of the algorithm if all the input elements are equal? [5 points]

Ans:

O(*n*)

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 (*k*1,*k*2,...,*kd*), where each *ki* is an integer in the range [0, *N* − 1]? [10 points]

Ans:

Given a sequence S:

S*kd* = {S sorted via the *d*thelement of each *d*-tuple}

S*kd-1*,*kd* = {S*kd* sorted via the (*d* – 1)th element of each *d*-tuple}

…

S*ki*,…,*kd* = {S*ki+1,…,kd* sorted via the *i*th element of each *d*-tuple}

…

S1,…,*kd* = (S*2,3,…,kd* sorted via the 1st element of each *d*-tuple}

R-12.23 Give an example input that requires merge-sort and heap-sort to take *O*(*n*log*n*) time to sort, but insertion-sort runs in *O*(*n*) time.

Ans:

((1,4), (2, 5), (3, 1), (6, 2))

What if you reverse this list? [10 points]

Ans:

Insertion takes O(*n*2) time, merge & heap take O(*n*log*n*) time still.

C-12.26 Describe and analyze an efficient comparison-based method for removing all duplicates from a collection *A* of *n* elements. [10 points]

Ans:

Use heap-sort. The book says its good for comparison-based methods.

P-12.59 Perform benchmarking tests on a version of merge-sort and quick-sort to determine which one is faster. Your tests should include >1000 sequences that are “random” as well as “almost” sorted. Implement your algorithm in the language of your choice. Provide your code, sample data and output of your algorithm on the data. [10 points]

Ans:

I don’t have the necessary amount of patience to even attempt this.