CSE 332 Summer 18
Exercise 07

Building Sorts

Due Date: Friday July 27, 11:59 PM
Submit as a pdf to gradescope.

In this exercise we'll see how sorting relates to building data structures that maintain some sort of ordering property (here, heaps and AVL trees).

1. We saw Floyd's BuildHeap could improve the running time of HeapSort by a constant factor. In this problem, we'll use Floyd's BuildHeap to design sorting-like algorithms where the BuildHeap method gives us a better asymptotic running time.

   Consider the k-sorting problem.

   > k-sorting
   > Given an array of unsorted elements, find the k smallest elements in the array.

   Consider three algorithms for this problem:

Algorithm A	Algorithm B	Algorithm C

The $O(n \log k)$ algorithm you implemented Project 2.	`Heap H = empty heap`
	`for(i from 1 to n)`
	`  H.insert(A[i])`
	`for (i from 1 to k)`
	`  print H.removeMin()`
		`Heap H = BuildHeap(A)`
		`for (i from 1 to k)`
		`  print H.removeMin()`

   (a) What are the running times of Algorithms B and C? (give O() bounds in terms of n and k)? [2 points]

   (b) If k is a constant, what do the big-O running times of A,B, and C become? (Your answers for this part should not have k anywhere – big-O notation suppresses constants) [2 points]

   (c) If k is $\sqrt{n}$, what do the big-O running times of A,B, and C become? [2 points]

   (d) Based on your answers for the previous parts, explain when algorithm C is better than B by more than a constant factor. [2 points]

   (e) Give a scenario in which you would use Algorithm A over B and C, and explain why it is better. [2 points]

2. We spent half of a lecture (lecture 4) designing BuildHeap. We did not do the same for BuildAVL:

`BuildAVL`
Given an unsorted array, create an AVL tree containing the elements of the array.

In this problem, we'll figure out why we haven't gone over it.

(a) First, we'll use a **reduction**. Design an algorithm to sort an array that uses `BuildAVL` as a black box. Your algorithm should do only $O(n)$ work beyond the `BuildAVL` call, and no extra comparisons. [4 points]

(b) Argue that any comparison-based algorithm for `BuildAVL` must take $\Omega(n \log n)$ time. [4 points]

(c) Explain why an AVL Tree you implemented in Project 2 has to use comparisons (i.e. why you couldn't use a fancy $O(n)$ sort as part of some hypothetical `BuildAVL` method) [2 points]

Now it's clear why we didn't design a fancy `BuildAVL` algorithm – it doesn't exist!