Tutorial 5: AVL-trees, Heaps & the Master Theorem

(1) Draw the AVL-tree that is created when the following sequence of operations is performed on the empty tree:

- `insertItem` with keys `44, 17, 32, 78, 50, 88, 48, 62`
- `removeItem` with keys `32, 88`

(2) Suppose we have two heaps $A, B$ held as trees using pointers. We want to make one heap from them and will consider two possible approaches. We assume that $A$ and $B$ have $n$ elements each and assume further that they are complete (i.e., there are no missing vertices at the lowest level of the trees).

(a) One possibility is to put $A$ and $B$ into a single array and then use `buildHeap` (see the notes on Priority Queues and Heaps). What is the running time of this approach?

(b) The preceding approach ignores the fact that $A$ and $B$ are already heaps. Can you think of a way to take advantage of this fact and join them together faster? We want to beat the bound obtained above by a convincing margin.

(3) Devise an $O(n \log n)$ sorting algorithm based on AVL trees. To be precise the algorithm takes an array $A$ of $n$ integers and outputs it in sorted order, using an AVL tree as an auxiliary structure. As usual you can specify your algorithm either in pseudocode or in English or indeed a mixture (my solution gives one part in English and a subsidiary part in pseudocode). For full credit, you must include a runtime analysis. You are not required to provide a proof of correctness but should make a brief comment as to why your algorithm is indeed correct.

(4) Solve, as a $\Theta$-expression, the following recurrences, i.e., the recurrences give us a function in each case and we want its growth rate.

(a) $s(n) = \begin{cases} \Theta(1), & \text{if } n \leq 3; \\ 2s(n/2) + \Theta(1), & \text{if } n > 3. \end{cases}$

(b) $T(n) = \begin{cases} \Theta(1), & \text{if } n \leq 2; \\ 2T([n/3]) + T([n/3]) + \Theta(n), & \text{if } n > 2. \end{cases}$

(5) Consider the pseudocode in Figure 4.1 where $A$ is an array of $n$ integers.

Let $T_A$ and $T_B$ be the worst case runtimes of `algA` and `algB` respectively.

(a) Deduce the recurrence for $T_A$ (this will of course involve $T_B$); you can make the reasonable assumption that $T_B(1) = \Theta(1)$.

\begin{verbatim}
algA(A)
1. algB(A)
2. if n > 1 then
3. new array B of length [n/2]
4. new array C of length [n/2]
5. for i ← 0 to [n/2] − 1 do
7.     for i ← [n/2] to n − 1 do
8.         C[i − [n/2]] ← A[i]
9.     algA(B)
10.    algA(C)
11. for i ← 0 to n − 1 do
12.    if i < [n/2] then
14.    else
15.        A[i] ← C[i − [n/2]]
16.    algB(A)

Figure 4.1.
\end{verbatim}

(b) Assume now that $T_B(n) = \Theta(n^\alpha)$ for some $\alpha \geq 0$. Find $T_A(n)$ as a $\Theta$ expression when $\alpha \leq 1$ and when $\alpha > 1$. 