CMSC 420: Spring 2023

CMSC 420 (0201) - Midterm Exam 1

Problem 1. (10 points)

(a) (5 points) Show the final tree after performing `insert(25)` into the AVL tree shown below. Show both the **final tree** and the **balance factors** for all the nodes. (Intermediate results can be given for partial credit.)

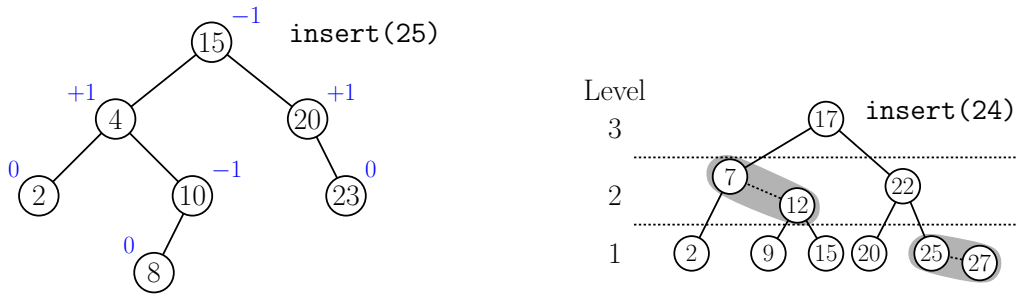

Figure 1: AVL and AA tree operations.

(b) (5 points) Consider the AA tree in Fig. 1 (right). (Note that nodes 12 and 27 are both red.) Show the final tree after performing `insert(24)` on this tree. Indicate the **levels** as well. (Intermediate results can be given for partial credit.)

Problem 2. (35 points) Short answer questions. Explanations are not required, but may always be given to help with partial credit.

(a) (4 points) You have a binary tree nodes with inorder threads. There are n nodes. How many threads are there? (If the answer cannot be determined from n alone, answer "It depends").

(b) (3 points) You have a union-find data structure for a set of n objects. After initialization, you perform k union operations, where $1 \leq k \leq n-1$. As a function of n and/or k, what is the maximum possible height of any tree in the data structure?

(c) (4 points) We showed that in any leftist heap containing n nodes the NPL (null path length) of the root is $O(\log n)$. What is the best that can be said about the NPL value of the root node in *any* binary tree having n nodes? (Select one.)

(1) $O(\log n)$
(2) $O(\log^2 n)$
(3) $O(\sqrt{n})$
(4) $O(n)$

(d) (4 points) You perform the operation `update-key` in a leftist heap containing n entries. Assuming you do this by sifting the entry up or down the tree, what is the worst-case time for this operation? (Select one)

(1) $O(1)$ time
(2) $O(\log n)$ time
(3) $O(\sqrt{n})$ time
(4) $O(n)$ time
(5) $O(n \log n)$ time

(e) (4 points) You perform a postorder traversal of a binary tree with $n \geq 1$ nodes. True or false: The first node in the traversal must be a leaf.

(f) (6 points) You want to maintain an ordered dictionary using a simple linear list. List (very briefly) one advantage and one disadvantage of an array-based approach versus a linked-list approach.

(g) (6 points) You have an AA tree whose root resides at level 13. What is the minimum and maximum number of red nodes that can be encountered along any path from the root to the leaf level (that is, level 1)?

(h) (4 points) You have a data structure in which the ith operation takes time $O(i)$. Given a string of n operations on this data structure, what is the best that can be said about the *amortized time* of each operation? **Hint:** $\sum_{i=1}^{m} i = m(m+1)/2$.

(1) $O(1)$
(2) $O(\sqrt{n})$
(3) $O(n)$
(4) $O(n^{3/2})$
(5) $O(n^2)$

Problem 3. (15 points) Throughout this problem assume that you have a multi-way tree represented in the first-child/next-sibling representation. Recall the node structure.

```
class Node {              // a node of a multi-way tree
  Key data                // data
  Node firstChild         // leftmost child (null if leaf)
  Node nextSibling        // next sibling to right (null if rightmost child)
}
```

Given any node `p` of the tree, define its *leftmost leaf* to be the leftmost leaf in the subtree rooted at `p`. If `p` is a leaf, then it is its own leftmost leaf. Otherwise, it is the leftmost leaf of its leftmost child. The *rightmost leaf* of `p` is defined symmetrically (see Fig. 2).

(a) (5 points) Present pseudocode for a function `Node leftLeaf(Node p)` that returns `p`'s leftmost leaf. For full credit, express your answer using just recursion (no loops). (You may create additional helper functions.) For half credit, you may use loops.

(b) (10 points) Same as (a), but for the rightmost leaf of `p`, `Node rightLeaf(Node p)`. Again, for full credit use recursion only, no loops. Loops are okay for partial credit.

Problem 4. (25 points) Alice and Bob continue to test their study of the performance of standard (unbalanced) binary search trees. They test a new insertion pattern. First, they pick a positive integer k, they fill the entries of an upper triangular $k \times k$ matrix row-by-row with

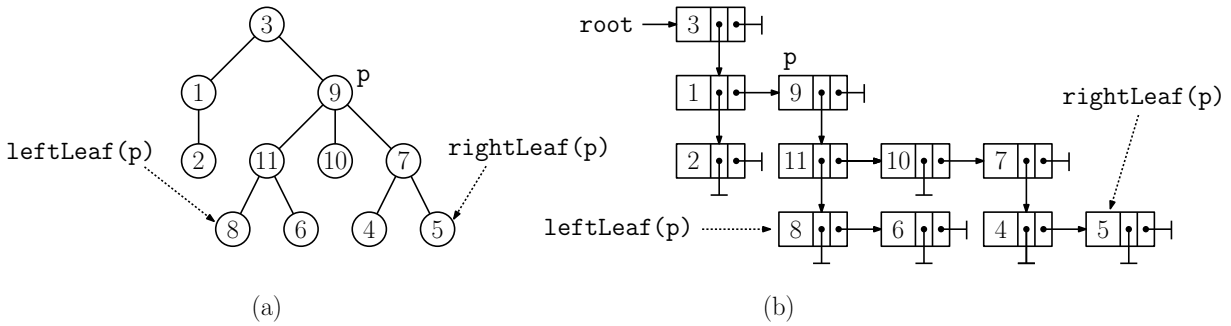

Figure 2: Leftmost and rightmost leaves in a multi-way tree.

numbers $\{1, 2, 3, \ldots\}$, and then they insert the keys into an unbalanced binary search tree by reading down the columns, working from right to left. An example for $k = 5$ is shown below.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ & 6 & 7 & 8 & 9 \\ & & 10 & 11 & 12 \\ & & & 13 & 14 \\ & & & & 15 \end{bmatrix}$$

They would insert $n = 15$ keys in the order $\langle 5, 9, 12, 14, 15, 4, 8, 11, 13, 3, 7, 10, 2, 6, 1 \rangle$.

Answer the following questions based on Alice and Bob's insertion process.

(a) (5 points) Given $k \geq 1$, what is the total number of keys n inserted? (Express your answer as a function of k.) **Hint:** For any $m \geq 0$, $\sum_{i=1}^{m} i = m(m+1)/2$.

(b) (5 points) Draw the final binary search tree that results for the above example when $k = 5$. **Hint:** It has a very regular structure.

(c) (5 points) What is the height of the resulting tree? (Express your answer as a function of k and/or n.)

(d) (5 points) Letting h denote the height of the final tree. Give a formula $d(i)$, which for $0 \leq i \leq h$ is the number of nodes at depth i in the resulting tree. Express your answer as a function of k, n, and/or i.

(e) (5 points) Assuming that the time needed to insert a node at depth i is $i+1$, what is the total time needed to insert all n keys in the tree? For full credit, express your answer in big-O notation as an tight asymptotic function of n in closed form—no summations or recurrences. **Hint:** For any $m \geq 0$, $\sum_{i=1}^{m} i^2 = (2m^3 + 3m^2 + m)/6$.

Problem 5. (15 points) In this problem we will analyze the amortized complexity of a new variant of the expanding stack. Rather than doubling, the size of the array m grows as a **perfect square**, that is, $m = k^2$ for some $k \geq 1$. We start with an array of size 1, and then subsequent overflows result in arrays of sizes 4, 9, 16, 25, and so on. Each operation costs +1 work unit, unless it causes the array to overflow. If so, we allocate a new larger array, copy the old contents into it. *The actual cost of the expansion is the number of elements copied.*

See Fig. 3 for an example. A run starts when we overflow an array of size $(k-1)^2$, resulting in an array of size $m = k^2$. When this array overflows, we allocate an array of size $(k+1)^2$ and copy all m elements to it.

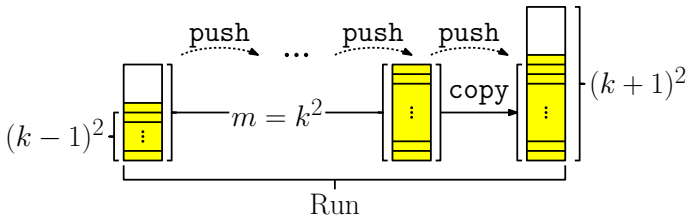

Figure 3: Perfect-square expanding array.

(a) (10 points) Consider a single run involving an array of size $m = k^2$ for some large k Fig. 3. Derive the worst-case amortized cost of this single run, including costs of the operations to fill the array and the expansion cost. Express your answer as a function of m and/or k. Since k is large, you may ignore $O(1)$ additive terms.

(b) (5 points) Suppose that we start with an empty stack and an array of size $m = 1$ and perform n operations, for some very large number n. What is the worst-case asymptotic amortized cost of this data structure as a function of n? Express your answer as a function of n using big-O notation. (A brief explanation suffices. We don't need a formal proof.)