ECE608 CHAPTER 12 PROBLEMS

1) 12.2-1
2) 12.2-3
3) 12.2-4
4) 12.2-8
5) 12.3-3
6) 12.3-4
7) 12.4-2
8) 12.4-3
9) 12-2
10) 12-3

(1) CLR 12.2-1

Based on the structure of the binary tree, and the procedure of TREE-SEARCH, any node that is examined by the algorithm must either be $\geq$ all subsequent nodes examined, or $\leq$ all subsequent nodes. This occurs because the search moves down the binary tree, either to the right or the left.

(a) 2, 252, 401, 398, 330, 344, 397, 363

This is a valid search :

$$\begin{aligned}
2 &\leq \text{all subsequent nodes} \\
252 &\leq \text{all subsequent nodes} \\
401 &\geq \text{all subsequent nodes} \\
398 &\geq \text{all subsequent nodes} \\
330 &\leq \text{all subsequent nodes} \\
344 &\leq \text{all subsequent nodes} \\
397 &\geq \text{all subsequent nodes} \\
363 &= \text{search value}
\end{aligned}$$

(b) 924, 220, 911, 244, 898, 258, 362, 363

This is a valid search :

$$\begin{aligned}
924 &\geq \text{all subsequent nodes} \\
220 &\leq \text{all subsequent nodes} \\
911 &\geq \text{all subsequent nodes}
\end{aligned}$$

$$\begin{aligned}
244 &\leq \text{all subsequent nodes}\\
898 &\geq \text{all subsequent nodes}\\
258 &\leq \text{all subsequent nodes}\\
362 &\leq \text{all subsequent nodes}\\
363 &= \text{search value}
\end{aligned}$$

(c) 925, 202, 911, 240, 912, 245, 363

This is an invalid search :

$$\begin{aligned}
925 &\geq \text{all subsequent nodes}\\
202 &\leq \text{all subsequent nodes}\\
911 &\ngeq \text{all subsequent nodes}\\
911 &\nleq \text{all subsequent nodes}
\end{aligned}$$

(d) 2, 399, 387, 219, 266, 382, 381, 278, 363

This is a valid search :

$$\begin{aligned}
2 &\leq \text{all subsequent nodes}\\
399 &\geq \text{all subsequent nodes}\\
387 &\geq \text{all subsequent nodes}\\
219 &\leq \text{all subsequent nodes}\\
266 &\leq \text{all subsequent nodes}\\
382 &\geq \text{all subsequent nodes}\\
381 &\geq \text{all subsequent nodes}\\
278 &\leq \text{all subsequent nodes}\\
363 &= \text{search value}
\end{aligned}$$

(e) 935, 278, 347, 621, 299, 392, 358, 363

This is an invalid search :

$$\begin{aligned} 935 &\geq \text{all subsequent nodes} \\ 278 &\leq \text{all subsequent nodes} \\ 347 &\ngeq \text{all subsequent nodes} \\ 347 &\nleq \text{all subsequent nodes} \end{aligned}$$

(2) CLR 12.2-3

```
TREE-PREDECESSOR($x$)
1. **if** $left[x] \neq NIL$
2.     **then return** TREE-MAXIMUM($left[x]$)
3. $y \leftarrow p[x]$
4. **while** $y \neq NIL$ **and** $x = left[y]$
5.         **do** $x \leftarrow y$
6.            $y \leftarrow p[y]$
7. **return** $y$
```

(3) CLR 12.2-4

Consider the tree in Figure 1, with a search path beginning at the root and searching for the key 9, a leaf. So, $A = \{7\}, B = \{6, 8, 9\}$, and $C = \{\}$. Choose $7 \in A, 6 \in B$, but $7 > 6$. Clearly the professor's claim is false.

(4) CLR 12.2-8

Assume that the node n has k successors. If it does, its successors are comprised of elements in the subtree rooted by n's right child, ancestors that have node n in its

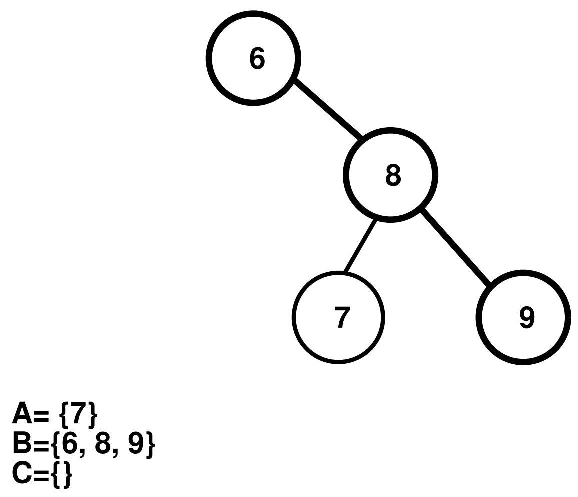

Figure 1: The Search Tree for CLR 12.2-4

left tree, and nodes that fall into the right trees of those ancestors that are one of n's k successors.

If node n has a right subtree with $0 < m \leq k$ nodes, then we can obtain those m successors by performing exactly the same inorder walk as does INORDER-TREE-WALK on n's right tree. Because the routine traverses each of the tree edges at most twice, the running time is $2m$ to obtain the successors. To see that an edge is traversed at most twice, notice that the only place in the code that goes up is in the while loop (or just before entering it) in TREE-SUCCESSOR. The only place it goes down is in the call to TREE-MINIMUM. Any time the code moves down a particular edge, it will end up returning up the same edge. Furthermore, no edge is traversed more than once in the same direction.

If node n with height $i \leq h$ has k nodes in its right subtree, then the time to traverse the k successors is $2k$. Note that to get to the minimum in the right subtree requires at most the traversal of i links. Hence, the time to traverse the successors is $O(h+k)$.

If node n has height $i < h$ with $m < k$ nodes in its right subtree, then the time to traverse the m successors contained in the subtree is $2m$. To obtain the remaining $k-m$ successor nodes, we must traverse the parent links to find ancestors that contain n in their left subtrees. Each successor ancestor accounts for one of the k successors at minimum and up to 1 + the number of elements in its right subtree. Assume that p successor ancestors must examined to obtain the k successors. Since an ancestor of n cannot have a height that is greater than the root, the time to traverse the links between n and the ancestors is $O(h)$ time. Suppose that $A_1, A_2, \ldots A_p$ are successor ancestors of n with right subtrees containing $m_1, m_2, \ldots m_p$ nodes, respectively, and $m+p+m_1+\ldots+m_p \geq k$, $m+p+m_1+\ldots+m_{p-1} \leq k$, with $r = k-(m+p+m_1+\ldots+m_{p-1})$ representing the successors of n in A_p's right tree that must be explored to obtain the k successors. Then the time needed to obtain the successors is less than or equal to $2(m+m_1+\ldots+m_{p-1}+r)+h$. Since $m+m_1+\ldots+m_{p-1}+r < k$, the time to traverse the successors is $O(h+k)$.

(5) CLR 12.3-3

Consider the following pseudocode for sorting an array A of length n:

```
SORT($A$) **for** $i \leftarrow 1$ to $n$
          **do** TREE-INSERT($A[i]$)
INORDER-TREE-WALK(*root*)
```

The worst-case running time occurs when the list of elements to be inserted in the tree are already sorted in increasing (or decreasing) order. In this case, each element will be inserted into a linear list after being compared to all of the elements that have already been inserted. The first element can be inserted with no comparisons, the second with 1, the third with 2, …, the ith with $i-1$, …, and the nth with $n-1$ comparisons; hence, the time to insert the ith element is $i-1+1=i$, giving $\sum_{i=1}^{n} i = \frac{(n+1)n}{2} = \Theta(n^2)$ worst-case time. Note that because the time to perform the inorder tree walk is $\Theta(n)$, the overall worst-case time is $\Theta(n^2)$.

The best case occurs when the order of the numbers in the set results in a balanced binary search tree. In this case, the first element will be compared to no others, the left and right child will have one comparison to the root, their children will have two comparisons, etc. For $i \geq 1$, the ith element inserted will be compared to $\lfloor \lg i \rfloor$ elements for its insertion, giving a time to insert of $\sum_{i=1}^{n}(\lfloor \lg i \rfloor + 1) \leq \sum_{i=1}^{n}(\lg i + 1) \leq 2\sum_{i=1}^{n} \lg i = 2\lg(\prod_{i=1}^{n} i) = 2\lg(n!) = O(n \lg n)$. Also, $\sum_{i=1}^{n}(\lfloor \lg i \rfloor + 1) \geq \sum_{i=1}^{n}((\lg i - 1) + 1) = \sum_{i=1}^{n} \lg i = \lg(n!) = \Omega(n \lg n)$. Note that because of this and the time to perform the inorder tree walk is $\Theta(n)$, the overall best-case time is $\Theta(n \lg n)$.

(6) CLR 12.3-5

It is false. Consider the counterexample in Figure 2. It relies on the fact that we replace a node with its successor if it has two children but with its child if it is a singleton.

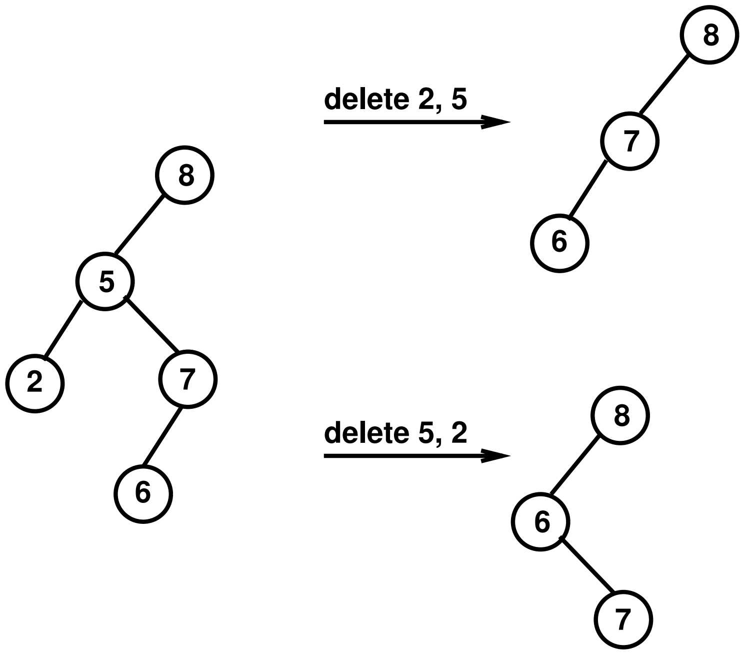

Figure 2: The counterexample for CLR 12.3-5

(7) CLR 12.4-2

We need to construct a binary tree such that the height is $\omega(\lg n)$ i.e., it should be asymptotically strictly greater than $\lg n$. At the same time we need to guarantee that the average height of the nodes in the tree is $\Theta(n)$:

$$h_{avg} = \frac{1}{n}\sum_{i=1}^{n} h_i$$

We can visualize an instance of such a tree by first starting with a balanced binary tree. In this tree each internal node has two children and the height of the tree is $\lg n$. We remove a number of leaf nodes and then attach them as a subtree to one of the leaves, concatenating them into a chain such that each of the nodes has only one child. This new tree is almost a balanced tree except for a long chain of nodes which makes the height of the tree $\lg n + x$, where x is the number of leaves popped off and chained. We need to choose x such that $x = \omega(\lg n)$, while maintaining the expression for h_{avg} given above. The nodes in the balanced portion of the tree have depth $\lg i$ where i is the index counting the nodes starting at the root. There are $n - x$ nodes in the balanced portion of the tree. The internal node to which the chain of x concatenated nodes is attached is at the last level of the balanced tree and so it's depth is $\lg(n - x)$. The j'th node in the concatenated chain has a depth of $\lg(n - x) + j$. Now we can express the average height of the nodes as:

$$h_{avg} = \frac{1}{n}\sum_{i=1}^{n} h_i$$

$$h_{avg} = \frac{1}{n}(\sum_{i=1}^{n-x} \lg i + \sum_{i=1}^{x}(\lg(n-x) + j))$$

$$h_{avg} = \frac{1}{n}(\lg(\Pi_{i=1}^{n-x} i) + \sum_{j=1}^{x} \lg(n-x) + \sum_{j=1}^{x} j)$$

$$h_{avg} = \frac{1}{n}(\lg(n-x)! + \lg(\Pi_{j=1}^{x}(n-x)) + \frac{x(x-1)}{2})$$

$$h_{avg} = \frac{1}{n}((n-x)\lg(n-x) + \lg(n-x)^x + \frac{x(x-1)}{2})$$

$$h_{avg} = \frac{1}{n}(n\lg(n-x) - x\lg(n-x) + x\lg(n-x) + \frac{x(x-1)}{2})$$

$$h_{avg} = \lg(n-x) + \frac{x^2 - x}{2n})$$

Choosing $x = \sqrt{n} = \omega(\lg n)$ gives us :

$$h_{avg} = \lg{(n-x)} + \frac{\sqrt{n}^2 - \sqrt{n})}{2n})$$

$$h_{avg} = \lg{(n-x)} + O(1))$$

$$h_{avg} = \Theta(\lg n)$$

(8) CLR 12.4-3

Lets show this difference by looking at the example of the binary trees that can be built using three keys $1, 2, 3$, ranked in increasing order of key value.

There are 5 possible binary trees that can be built on 3 keys. Each of these are equally likely to be chosen with a probability of $1/5$:

```
.        1                1                2                3                3
.         2                3             1    3           2                1
.          3              2                            1                    2
```

However there are 6 possible permutations given 3 keys, all being equally likely when randomly constructing a tree:

1) $1, 2, 3$ corresponds to building tree #1 from the left shown above.

2) $1, 3, 2$ corresponds to building tree #2 from the left shown above.

3) $2, 1, 3$ corresponds to building tree #3 from the left shown above.

4) $2, 3, 1$ corresponds to building tree #3 from the left shown above.

5) $3, 2, 1$ corresponds to building tree #4 from the left shown above.

6) $3, 1, 2$ corresponds to building tree #5 from the left shown above.

Thus we can see that when the tree is randomly constructed, tree #3 occurs with probability $2/6$ whereas all other trees have probability of $1/6$ to occur. In contrast, when randomly choosing from the set of all possible trees, each tree would have a probability of $1/5$ to be chosen.

(9) CLR 12-2

To sort the strings of S, we first insert them into a radix tree and then use a preorder tree walk to extract them in lexicographically sorted order. The tree walk outputs strings only for "white" nodes (nodes that indicate the existence of a string).

Correctness: The preorder ordering is the correct order because any node's string is a prefix of all its descendants' strings, hence belongs before them in the sorted order (rule 2). Also, a node's left descendants belong before its right descendants because the corresponding strings are identical up to that parent node, and in the next position the left tree's strings have 0 while the right tree's strings have 1 (rule 1).

Run time: It is $\Theta(n)$ because insertion takes $\Theta(n)$ time, since the insertion of each string takes time proportional to its length (traversing a path through the tree whose length is the length of the string), and the sum of all the string lengths is n. Also, the tree walk takes $O(n)$ time. It is just like PREORDER-TREE-WALK (it prints the current node and calls itself recursively on the left and right subtrees), so it takes time proportional to the number of nodes in the tree. The number of nodes is $\leq 1+$ the sum of the lengths of the binary strings in the tree (i.e., n), because a length-i string corresponds to a path through the root and i other nodes, but a string node may be shared among many string paths.

(10) CLR 12-3

(a) To obtain the average depth of a node in a tree, it is sufficient to sum all depths of all nodes and divide by the total number of nodes; hence, since $P(T) = \sum_{x \in T} d(x,T)$, the average depth of a node in the tree is $\frac{1}{n} \sum_{x \in T} d(x,T) = \frac{1}{n} P(T)$.

(b) T_L and T_R represent the left- and right-subtrees of T respectively, and T has n nodes.

The number $P(T_R)$ represents the internal path length of the subtree T_R with respect to the root of T_R. Similarly, $P(T_L)$ represents the internal path length of

T_L with respect to its root. However, since T_L and T_R are subtrees of the tree T, every node in T_R and T_L is actually at a depth that is one greater than the value used to calculate $P(T_R)$ and $P(T_L)$. Since the two subtrees contain $n-1$ nodes total, the value $P(T)$ can thus be calculated as $P(T) = P(T_R) + P(T_L) + n - 1$.

(c) $P(n)$ is the average internal path length of a randomly built binary search tree T with n nodes.

The tree T will have one node as a root. Thus, there will be $n-1$ nodes distributed among the left and right subtrees. This distribution is random, and follows the pattern:

Left	Right
0	$n-1$
1	$n-2$
$\vdots$	$\vdots$
$n-2$	1
$n-1$	0

Since there are n total possible distributions, and they are all equally likely, we get:

$$P(n) = \frac{1}{n}\sum_{i=0}^{n-1}(P(i) + P(n-i-1) + n - 1)$$

(d)

$$\begin{aligned}
P(n) &= \frac{1}{n}\sum_{i=0}^{n-1}(P(i) + P(n-i-1) + n - 1) \\
&= \frac{1}{n}(\sum_{i=0}^{n-1} P(i) + \sum_{i=0}^{n-1} P(n-i-1) + \sum_{i=0}^{n-1}(n-1)) \\
&= \frac{1}{n}(\sum_{i=0}^{n-1} P(i) + \sum_{j=0}^{n-1} P(j) + n(n-1)) \\
&= \frac{2}{n}\sum_{i=0}^{n-1} P(i) + n - 1 \\
&= \frac{2}{n}\sum_{i=0}^{n-1} P(i) + \Theta(n)
\end{aligned}$$

(e) Solve the recurrence by the substitution method:

$$P(n) = \frac{2}{n}\sum_{q=1}^{n-1} P(q) + \Theta(n)$$

Guess: $P(n) \le a\,n\,\lg n + b$

$$\begin{aligned}
P(n) &= \frac{2}{n}\sum_{q=1}^{n-1} P(q) + \Theta(n) \\
&\le \frac{2}{n}\sum_{q=1}^{n-1}(aq\lg q + b) + \Theta(n) \\
&= \frac{2a}{n}\sum_{q=1}^{n-1} q\lg q + \frac{2b}{n}(n-1) + \Theta(n)
\end{aligned}$$

Show: $\sum_{q=1}^{n-1} q\lg q \le \frac{1}{2}n^2\lg n - \frac{1}{8}n^2$ by splitting the sum into two parts:

$$\begin{aligned}
\sum_{q=1}^{n-1} q\lg q &= \sum_{q=1}^{\lceil \frac{n}{2}\rceil - 1} q\lg q + \sum_{q=\lceil \frac{n}{2}\rceil}^{n-1} q\lg q \\
&\le \sum_{q=1}^{\lceil \frac{n}{2}\rceil - 1} q\lg \frac{n}{2} + \sum_{q=\lceil \frac{n}{2}\rceil}^{n-1} q\lg n \\
&= \sum_{q=1}^{\lceil \frac{n}{2}\rceil - 1} q(\lg n - 1) + \sum_{q=\lceil \frac{n}{2}\rceil}^{n-1} q\lg n \\
&= (\lg n - 1)\sum_{q=1}^{\lceil \frac{n}{2}\rceil - 1} q + \lg n \sum_{q=\lceil \frac{n}{2}\rceil}^{n-1} q \\
&= \lg n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\lceil \frac{n}{2}\rceil - 1} q \\
&\le \lg n \frac{n(n-1)}{2} - \frac{(n/2-1)n/2}{2} \\
&\le \frac{1}{2}n^2\lg n - \frac{1}{8}n^2, \text{ if } n \ge 2
\end{aligned}$$

Using the bound $\sum_{q=1}^{n-1} q\lg q \le \frac{1}{2}n^2\lg n - \frac{1}{8}n^2$:

$$\begin{aligned}
P(n) &\leq \frac{2a}{n}(\frac{1}{2}n^2 \lg n - \frac{1}{8}n^2) + \frac{2b}{n}(n-1) + \Theta(n) \\
&\leq an \lg n - \frac{a}{4}n + 2b + \Theta(n) \\
&= an \lg n + b + (\Theta(n) + b - \frac{a}{4}n) \\
&\leq an \lg n + b
\end{aligned}$$

because we can choose a large enough that $a/4\,n$ dominates $\Theta(n) + b$.

(f) In RANDOMIZED-QUICKSORT where the pivot is removed from further comparisons, comparisons to the first pivot correspond to comparisons to the root in RANDOMIZED-TREE-INSERT. Comparisons to the left partition's pivot correspond to comparisons to the left child of the root. Comparisons to the right partition's pivot correspond to comparisons to the right child of the root. Hence, we have modified PARTITION to place the pivot element in its correct position in the array and call QUICKSORT without that pivot element because the additional comparisons with that element would not be made if it were the root of a subtree.

```
QUICKSORT($A, p, r$)
1. if $p < r$
2.     then $q \leftarrow$ PARTITION($A, p, r$)
3.              QUICKSORT($A, p, q-1$)
4.              QUICKSORT($A, q+1, r$)
5. ▷ Notice that the pivot is not included in the recursive calls.
```

```
PARTITION($A, p, r$)
1. $x \leftarrow A[r]$
2. $i \leftarrow p - 1$
3. for $j \leftarrow p$ to $r - 1$
```

```
4.     do if $A[j] \leq x$
5.           then $i \leftarrow i + 1$
6.                exchange $A[i] \leftrightarrow A[j]$
7. exchange $A[i+1] \leftrightarrow A[r]$
8. return $i + 1$
```