Analysis of Algorithms - Scrimmage II (Solutions)

L. Kovalchick  
LCSEE,  
West Virginia University,  
Morgantown, WV  
{lynn@csee.wvu.edu}

1. Write down the order in which the nodes of tree T in Figure (1), will be visited, assuming an inorder traversal.

![Binary Tree T](image)

Figure 1: Binary Tree T

Observe that in an inorder traversal, the left children of a node are visited before it is visited and the right children of the node are visited after it is visited. Applying this recursively, we conclude that the nodes in T would be visited in the following order: 1, 2, 3, 4, 5, 6, 7.

2. A fair coin is flipped three times. What is the probability that you see more heads than tails?

**Proof:** The sample space for this experiment is:

\[ S = \{HHH, HTH, HHT, THH, THT, HTT, TTH, TTT, THT\} \]

Since the coin is fair, each of these outcomes will be equally likely to occur. Therefore, we assign a probability of \(\frac{1}{8}\) to each outcome (remember, the sum of the outcomes must equal 1). Let \(A\) represent the event of seeing more heads than tails, then:

\[ A = \{HHH, HTH, HHT, THH\} \]
So the probability of event $A$ occurring (i.e., you seeing more heads than tails) is:

\[
Pr(A) = \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = \frac{4}{8} = \frac{1}{2}
\]

3. Solve the recurrence:

\[
\begin{align*}
T(1) &= 1 \\
T(n) &= T(n-1) + n^2, \ n > 1
\end{align*}
\]

**Proof:** Using expansion:

\[
T(n) = T(n-1) + n^2
\]

\[
= [T(n-2) + (n-1)^2] + n^2
\]

\[
= [T(n-3) + (n-2)^2] + (n-1)^2 + n^2
\]

\[
\vdots
\]

\[
= [T(n-(n-1)) + (n-(n-2))^2] + (n-(n-3))^2 + \ldots + (n-1)^2 + n^2
\]

\[
= T(1) + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2
\]

\[
= 1 + \sum_{i=2}^{n} i^2
\]

\[
= 1^2 + \sum_{i=2}^{n} i^2
\]

\[
= \sum_{i=1}^{n} i^2
\]

\[
= \frac{n \cdot (n+1) \cdot (2n+1)}{6}
\]

4. Write an algorithm for finding the second smallest element in a binary search tree? What is its worst-case running time?

Let $v_{min}$ denote the node in the binary search tree $T$, whose key is minimum and $v_{min}^p$ denote its parent, if $v_{min}$ is not the root of $T$. Without loss of generality, we assume that $T$ has at least 2 elements (otherwise, the second smallest element is undefined!). We now make the following observations:

(a) $v_{min}$ can never be the right child of its parent. If $v_{min}$ is the root of $T$, then it has no parent; otherwise it does have a parent (viz. $v_{min}^p$) and if it is the right child of $v_{min}^p$, then $v_{min}^p$ has a key which is smaller than it, contradicting the minimality of $v_{min}$!
(b) If both the children of \( v_{\text{min}} \) are external, then the node with the second smallest key is clearly \( v_{\text{min}}^{r} \).

(Think carefully!)

(c) If \( v_{\text{min}} \) has a right child, then the node with the second smallest key is the minimum key node of \( v_{\text{min}}.rightchild() \).

Algorithms (0.1) and (0.2) put all these ideas together.

**Function** \( \text{FIND-SECOND-MIN}(T) \)

1. Let \( \text{root} = T.root \)
2. Let \( v = \text{FIND-MIN}(T, \text{root}) \)
3. if \( (T.\text{isRoot}(v)) \) then
4. \( \text{return}(\text{FIND-MIN}(T, v.rightchild())) \)
5. end if
6. if \( (\text{ISEXTERNAL}(T, v.rightchild())) \) then
7. \( \text{return}(v.\text{parent}) \)
8. else
9. \( \text{return}(\text{FIND-MIN}(T, v.rightchild())) \)
10. end if

**Algorithm 0.1:** Finding the smallest element in a binary search tree

**Function** \( \text{FIND-MIN}(T, v) \)

1: if \( (\text{ISEXTERNAL}(T, v.leftchild())) \) then
2: \( \text{return}(v) \)
3: else
4: \( \text{return}(\text{FIND-MIN}(T, v.leftchild())) \)
5: end if

**Algorithm 0.2:** Finding the smallest element in a binary search tree

In the worst-case, there could be 2 calls to \( \text{FIND-MIN}() \), each taking time \( \Omega(n) \). Thus the running time of \( \text{FIND-SECOND-MIN}() \) is \( O(n) \) in the worst case.

5. Show that the worst-case running time of \( \text{MERGE-SORT()} \) is \( O(n \log n) \), assuming that

\[
T(1) = O(1) \\
T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)
\]

**Proof:** Without loss of generality, assume that \( n = 2^{k} \) and hence \( k = \log_{2} n \).

Using expansion:

\[
T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n) \\
= 2[2 \cdot T\left(\frac{n}{4}\right) + O\left(\frac{n}{2}\right)] + O(n) \\
= 4 \cdot T\left(\frac{n}{4}\right) + 2 \cdot O(n) \\
= 4[2 \cdot T\left(\frac{n}{8}\right) + O\left(\frac{n}{4}\right)] + 2 \cdot O(n) \\
= 8 \cdot T\left(\frac{n}{8}\right) + 3 \cdot O(n) \\
\]

3
\[ = 2^k \cdot T\left(\frac{n}{2^k}\right) + k \cdot O(n)\]
\[ = 2^\log n \cdot T\left(\frac{n}{2^\log n}\right) + \log n \cdot O(n)\]
\[ = n \cdot T(1) + \log n \cdot O(n)\]
\[ = n \cdot O(1) + \log n \cdot O(n)\]
\[ = O(n) + O(n \log n)\]
\[ = O(n \log n)\]

\[\square\]

6. Show that if \( f(n) = \Theta(g(n)) \) and \( g(n) = \Theta(h(n)) \), then \( f(n) = \Theta(h(n)) \).

**Proof:** We want to show that \( f(n) = \Theta(h(n)) \), which implies that:

\[
\begin{align*}
    f(n) &= O(h(n)), \text{ and} \\
    h(n) &= O(f(n))
\end{align*}
\]

By definition of \( \Theta \), \( f(n) = \Theta(g(n)) \) implies that:

\[
\begin{align*}
    f(n) &= O(g(n)), \text{ and} \\
    g(n) &= O(f(n))
\end{align*}
\]

Also, by definition of \( \Theta \), \( g(n) = \Theta(h(n)) \) implies that:

\[
\begin{align*}
    g(n) &= O(h(n)), \text{ and} \\
    h(n) &= O(g(n))
\end{align*}
\]

Observe that:

\[
\begin{align*}
    f(n) &= O(g(n)) \\
    \Rightarrow f(n) &\leq c \cdot g(n) \\
    &= c \cdot O(h(n)) \\
    &\leq c \cdot c' \cdot h(n) \\
    &= c'' \cdot h(n)
\end{align*}
\]

Then, by definition of \( 'O' \), \( f(n) = O(h(n)) \).

Next, observe that:

\[
\begin{align*}
    h(n) &= O(g(n)) \\
    \Rightarrow h(n) &\leq c \cdot g(n) \\
    &= c \cdot O(f(n)) \\
    &\leq c \cdot c' \cdot f(n) \\
    &= c'' \cdot f(n)
\end{align*}
\]
Then, by definition of \( O \), \( h(n) = O(f(n)) \).

Then, by definition of \( \Theta \), \( f(n) = \Theta(h(n)) \). \( \square \)