Fundamental Algorithms, Assignment 6
Solutions

1. Consider a Binary Search Tree $T$ with vertices $a,b,c,d,e,f,g,h$ and $ROOT[T] = a$ and with the following values ($N$ means NIL)

<table>
<thead>
<tr>
<th>vertex</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>N</td>
<td>e</td>
<td>e</td>
<td>a</td>
<td>d</td>
<td>g</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>left</td>
<td>h</td>
<td>N</td>
<td>N</td>
<td>e</td>
<td>c</td>
<td>N</td>
<td>f</td>
<td>N</td>
</tr>
<tr>
<td>right</td>
<td>d</td>
<td>N</td>
<td>g</td>
<td>N</td>
<td>b</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>key</td>
<td>80</td>
<td>170</td>
<td>140</td>
<td>200</td>
<td>150</td>
<td>148</td>
<td>70</td>
<td></td>
</tr>
</tbody>
</table>

Draw a nice picture of the tree. Illustrate $\text{INSERT}[i]$ where $\text{key}[i] = 100$.
Solution: Here is the picture, without the key values.

```
    a
   / \     \\
  h   d
    / \
   e   \\
  c   b
   \ \
    f
```

For $\text{INSERT}[i]$: We start at root $a$ with $\text{key}[a] = 80$. As $80 < 100$ we replace $a$ by its right child $d$ with $\text{key}[d] = 200$. As $100 < 200$ we replace $d$ by its left child $e$ with $\text{key}[e] = 150$. As $100 < 150$ we replace $e$ by its left child $c$ with $\text{key}[c] = 140$. As $100 < 140$ we replace $c$ by its left child. But its left child is NIL so we make the new vertex $i$ its left child by setting $p[i] = c$ and $\text{left}[c] = i$.

2. Continuing with the Binary Search Tree of the previous problem:

(a) Which is the successor of $c$. Illustrate how the program $\text{SUCCESSOR}$ will find it.
Solution: The successor of $c$ is $f$. As $c$ has a right child $g$, $\text{SUCCESSOR}$ will call $\text{MIN}[g]$ which will go to the left as long as possible, ending (in one step) at $f$.

(b) Which is the minimal element? Illustrate how the program $\text{MIN}$ will find it.
Solution: $h$. Start at root $a$. Go to left: $h$. Go to left: NIL. Return $h$. 
(c) Illustrate the program \texttt{DELETE[e]}

\textbf{Solution:} There are two approaches (equally correct) to \texttt{DELETE[x]} when \(x\) has two children. One can effectively replace \(x\) by the maximum of its left tree or the minimum of its right tree.

\textbf{Solution 1:} \(e\) has a left child \(c\). Applying \texttt{MAX[c]} gives \(g\). \(g\) has a left child \(f\). So we splice \(f\) into \(g\)'s place by resetting \(\text{right}[d] = g\), \(\text{left}[g] = c\), \(\text{right}[g] = b\), and \(p[g] = d\).

\textbf{Solution 2:} \(e\) has right child \(b\). Applying \texttt{MIN[b]} gives \(b\) itself. We splice \(b\) into \(e\)'s place by resetting \(p[c] = b\) and \(\text{left}[b] = c\) and \(p[b] = d\) and \(\text{left}[d] = e\).

3. Set \(N = 2^K\). We'll represent integers \(0 \leq x < N\) by \(A[0\ldots(K-1)]\) with \(x = \sum_{i=0}^{K-1} A[i]2^i\). (This is the standard binary representation of \(x\), read right to left.) Consider the following algorithms:

\texttt{Procedure JACK[A]}
\begin{verbatim}
I ← 0
A[0] ++
\end{verbatim}
\begin{verbatim}
WHILE (A[I] = 2 AND I < K - 1)
\begin{verbatim}
A[I] ← 0
I ++
A[I] ++
\end{verbatim}
\end{verbatim}
\begin{verbatim}
END WHILE
\end{verbatim}

and:

\texttt{ANYA[A]}
\begin{verbatim}
FOR J = 1 TO N - 1
\begin{verbatim}
DO JACK[A]
\end{verbatim}
\end{verbatim}
\begin{verbatim}
END FOR
\end{verbatim}

If the input to \texttt{JACK[A]} is the binary representation of \(x\) with \(0 \leq x \leq N - 2\) describe what the output will be.

\textbf{Solution:} \texttt{JACK} increments by one, the final value of \(A\) will be the binary representation of \(x+1\). For example, if \(A\) (reading right to left) is 1100111 then it becomes 1100112, 1100120, 1100200, 11001000 and then stops.

4. Let \(T\) be a binary search tree on nodes 1, \ldots, \(N\) (in no particular order in the tree) with height \(H\). For any vertex \(v\) define \texttt{depth[v]} as the distance from \(v\) to the root. (The root has depth zero, its children
have depth one, grandchildren two, etc.) Let $TD$ denote the sum of $depth[v]$ for all nodes $v$.

(a) Give an algorithm to find any particular $depth[i]$ in time $O(H)$ and $TD$ in time $O(HN)$.

Solution: For a particular $i$ climb to the top:

1. DEPTH[i]
2. $d \leftarrow 0$
3. WHILE $i \neq \text{root}[T]$
4. $d + +$
5. end WHILE
6. return $d$

For $TD$ go through all vertices and take their sum.

(b) Modifying In-Order-Tree-Walk give an algorithm that finds all $depth[i]$ and also $TD$ in total time $O(N)$ – regardless of the value of $H$.

Solution: Initialize $r = \text{root}(T)$, $depth[r] = 0$, $TD = 0$. Now run IOTW[r]. When you call $IOTW[s]$ ($s \neq r$) set $depth(s) = depth(\pi(s)) + 1$ (note the depth of the parent has already been calculated!) and $TD \leftarrow TD + depth[s]$. 