Let $T$ be a binary tree such that all the external nodes have the same depth.

$D_e =$ sum of depths of all external nodes

$D_i =$ sum of depths of all internal nodes

Find $a$ and $b$ such that

$$De + 1 = a D_i + bn$$

where $n =$ number of nodes in $T$. 

Example: $h = 2$

$$De = h \cdot 2^h$$

$$No. \ of \ external \ nodes = 2^h - 2^d$$

$$No. \ of \ internal \ nodes = 2^d - 1 = 2^{h} - 1$$

$$De = 2 \cdot 2^2 = 8$$

$$D_i = 2$$

$$n = 7$$

$$De + 1 = a D_i + bn$$

$$8 + 1 = 2a + 7b$$

$$a = 1, \ b = 1$$
R-2.8  T is a binary tree with n nodes.

p is the level numbering of nodes of T such that p(n) = 1, a node v has left child numbered 2p(v) and right child numbered 2p(v + 1), if they exist.

(a) Show that for every node v of T, 

\[ p(v) \leq 2^{(v+1)/2} - 1 \]
(b) Consider the following binary tree that attains the maximum upper bound with 5 nodes:

```
     1
   /   \
  3     2
 /     /  \
2  6     7
```

\[ n = \frac{5}{2^{(5+1)/2} - 1} = \frac{2^{6/2} - 1}{2^3 - 1} = 7 \]

C-24 - Queues work in FIFO and stacks in LIFO.
- Use stack \( S_1 \) for enqueue operations.
- Use stack \( S_2 \) for dequeue operations.
- Push into \( S_1 \) until it's full for enqueue.
  Once it's full, pop and push all elements into \( S_2 \).
- If \( S_2 \) is empty, continuously pop and push from \( S_1 \).
- Otherwise, pop and return for dequeue.

```
Enqueue (c)
if (S1.full())
  while (S1.not.Empty())
    S2.push(S1.pop())
S1.push(c)

Dequeue
if (S2.empty())
  while (S1.not.Empty())
    S2.push(S1.pop())
out(c) S2.pop()
```
In the worst case, Enqueue requires $\Theta(n)$, and Dequeue requires the same. Thus, worst-case running time is $O(n)$.

C-2.6 Algorithm enumerate(A)
Input: Arrary A of $n$ elements
Output: ALL permutations of A
if $A.length() = 2$ then
    print (A)
    swap (A[0], A[1])
else
    for $i = 1$ to $(n-1)$ do
        swap (A[0], A[i])
        enumerate (A[1:] + A[0])
        swap (A[0], A[i])
    end for
end if
Using the same global array of size $n$ takes $O(n)$ space.
However, running time is $O(n!)$

C-2.11 Procedure Next(v): output the node visited after $v$ in a preorder traversal of $T$
Algorithm: preorder Next(Node v):
if $v$ is internal then
    output $v$'s left child
else
    Node p = parent of $v$
    if $v$ is the left child of $p$ then
        output right child of $p$
else
    end if
end if
while \( v \) is not left child of \( p \) do

\[ \begin{align*}
\text{\( v = p \)} \\
\text{\( p = p \), parent} \\
\text{return \( v \), right child of \( p \).}
\end{align*} \]

Algorithm \text{inorderNext} (\text{Node} \ v):

\[ \begin{align*}
\text{if \( v \) is internal () then} \\
\text{return \( v \)'s right child} \\
\text{else} \\
\text{Node} \ p = \text{parent of} \ v \\
\text{if \( v \) is left child of \( p \) then} \\
\text{return} \ p \\
\text{else} \\
\text{while \( v \) is not left child of \( p \) do} \\
\text{\( v = p \)} \\
\text{\( p = p \), parent} \\
\text{return} \ p
\end{align*} \]

Algorithm \text{postorderNext} (\text{Node} \ v):

\[ \begin{align*}
\text{if \( v \) is internal () then} \\
\text{\( p = \text{parent of} \ v \) } \\
\text{if \( v \) is left child of \( p \) then} \\
\text{return} \ p \\
\text{else} \\
\text{while \( v \) is not right child of \( p \) do} \\
\text{\( v = \text{right child of} \ p \)} \\
\text{while \( v \) is not internal do} \\
\text{return} \ v \\
\text{else} \\
\text{\( p = \text{parent of} \ v \) }
\end{align*} \]
if v is left child of p then
    parent of v is parent of p
else
    parent of v is p

Worst case running time for all algorithms is \( O(\log n) \) where \( n \) is the height of T.

C-3.1 Given an array \( A \) of \( n \) distinct integers in the range 1 to \( n + 1 \), find the missing integer.

\( \epsilon(x) \) Algorithm BinMissing \( (A, \text{lows, high}) \)
Input: Array \( A \) with \( n \) elements from 1 to \( n+1 \), \( \text{lows} \) and \( \text{high} \)
Output: Missing element in \( A \)

\( \text{mid} = \lfloor (\text{high} + \text{lows}) / 2 \rfloor \)
if \( (A[\text{mid}] = \text{mid}) \)
    \( \text{lows} = \text{mid} + 1 \)
else if \( (A[\text{mid}] > \text{mid}) \)
    if \( A[\text{mid} - 1] = \text{mid} - 1 \)
        return \( \text{mid} \)
    else
        \( \text{high} = \text{mid} - 1 \)
        BinMissing \( (A, \text{lows, high}) \)
else
    BinMissing \( (A, \text{lows, high}) \)
C-3.4 Algorithm `findAllElements(k, v, c)`:

Input: The search key `k`, a node of the binary search tree `v`, and a container `c`

Output: A set containing the found elements

if `v` is a internal node then
    output `c.elements`
if `k = key(v)` then
    `c.addElements(v)`
    output `findAllElements(k, T.rightChild(v), c)`
else if `k < key(v)` then
    output `findAllElements(k, T.leftChild(v))`
else
    output `findAllElements(k, T.rightChild(v))`

C-3.7 Solutions manual

C-3.10 Solutions manual