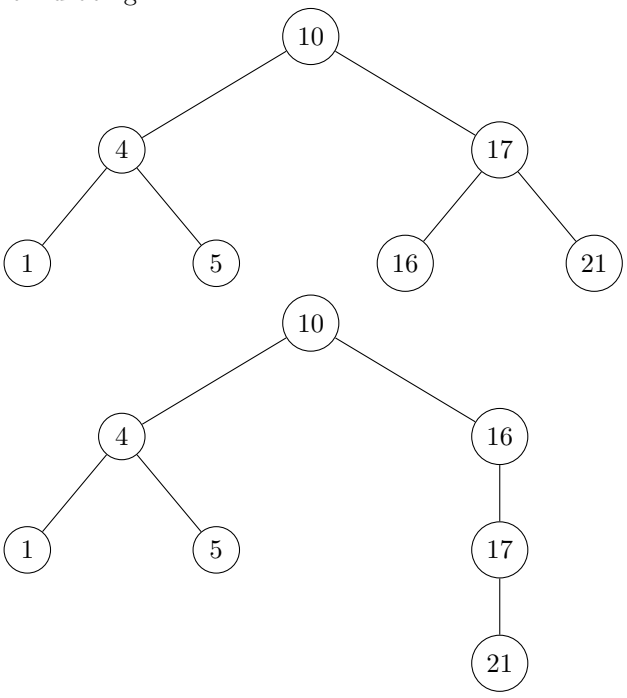
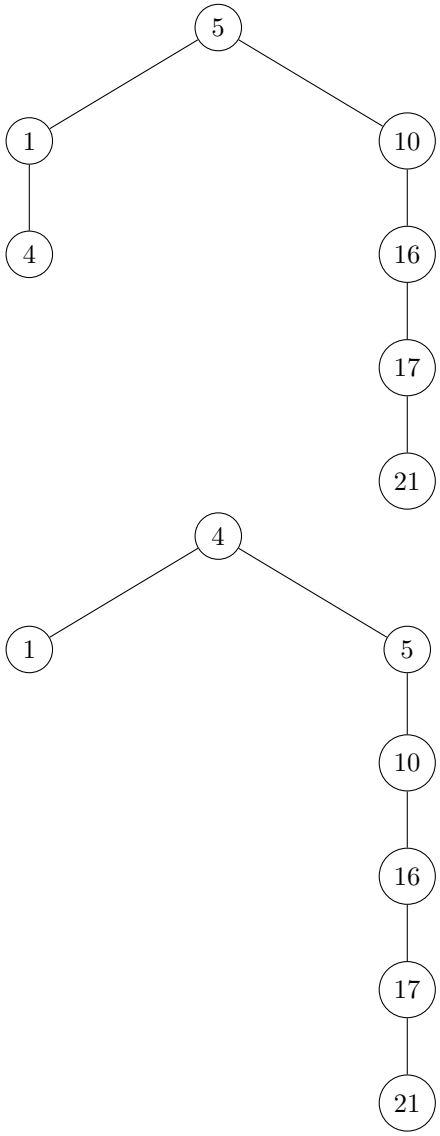
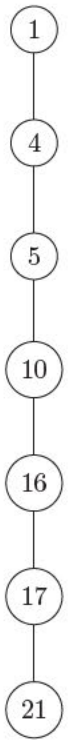
Exercise 12.1-1

Anytime that a node has a single child, treat it as the right child, with the left child being NIL




5
1
10
4
16
17
21
4
1
5
10
16
17
21



Exercise 12.1-2

In a heap, a node's key is $\geq$ both of its children's keys. In a binary search tree, a node's key is $\geq$ its left child's key, but $\leq$ its right child's key.

The heap property, unlike the binary-searth-tree property, doesn't help print the nodes in sorted order because it doesn't tell which subtree of a node contains the element to print before that node. In a heap, the largest element smaller than the node could be in either subtree.

Note that if the heap property could be used to print the keys in sorted order in $O(n)$ time, we would have an $O(n)$-time algorithm for sorting, because building the heap takes only $O(n)$ time. But we know (Chapter 8) that a comparison sort must take $\Omega(n \lg n)$ time.

Exercise 12.1-4

We call each algorithm on $T.root$.

Exercise 12.2-1

option c could not be the sequence of nodes explored because we take the left child from the 911 node, and yet somehow manage to get to the 912 node which cannot belong the left subtree of 911 because it is greater. Option e is also impossible because we take the right subtree on the 347 node and yet later come across the 299 node.

Exercise 12.2-2

Algorithm 3 TREE-MINIMUM(x)

if $x.left \neq NIL$ **then**
  return $TREE - MINIMUM(x.left)$
else
  return x
end if

Algorithm 4 TREE-MAXIMUM(x)

```
if x.right ≠ NIL then
    return TREE − MAXIMUM(x.right)
else
    return x
end if
```

Exercise 12.2-3

Algorithm 5 TREE-PREDECESSOR(x)

```
if x.left ≠ NIL then
    return TREE-MAXIMUM(x.left)
end if
y = x.p
while y ≠ NIL and x == y.left do
    x = y
    y = y.p
end while
return y
```

Exercise 12.2-6

First we establish that y must be an ancestor of x. If y weren't an ancestor of x, then let z denote the first common ancestor of x and y. By the binary-search-tree property, $x < z < y$, so y cannot be the successor of x.

Next observe that $y.left$ must be an ancestor of x because if it weren't, then $y.right$ would be an ancestor of x, implying that $x > y$. Finally, suppose that y is not the lowest ancestor of x whose left child is also an ancestor of x. Let z denote this lowest ancestor. Then z must be in the left subtree of y, which implies $z < y$, contradicting the fact that y is the successor if x.

Exercise 12.3-1

The initial call to TREE-INSERT-REC should be NIL,T.root,z

Exercise 12.3-3

Here's the algorithm:

```
TREE-SORT(A)
  let T be an empty binary search tree
  for i = 1 to n
      TREE-INSERT(T, A[i])
  INORDER-TREE-WALK(T.root)
```

Worst case: $\Theta(n^2)$—occurs when a linear chain of nodes results from the repeated TREE-INSERT operations.

Best case: $\Theta(n \lg n)$—occurs when a binary tree of height $\Theta(\lg n)$ results from the repeated TREE-INSERT operations.