12.1-2
Recall the following:

- **Binary-search-tree property:** Let $x$ be a node in a binary search tree. If $y$ is a node in the left subtree of $x$, then $\text{key}(y) \leq \text{key}(x)$. If $y$ is a node in the right subtree of $x$, then $\text{key}(x) \leq \text{key}(y)$.
- **Min-heap property:** For every node $i$ other than the root, $A[\text{Parent}(i)] \leq A[i]$.

a) The difference

Let $x$ be a node. A heap requires that $\text{key}(x)$ must be less than or equal to both $\text{key}[\text{left}[x]]$ and $\text{key}[\text{right}[x]]$. But a binary tree requires that $\text{key}[x] \geq \text{key}[\text{left}[x]]$, i.e. $\text{key}[x]$ cannot be less than $\text{key}[\text{left}[x]]$.

b) The min-heap property cannot be used to print out the keys in sorted order in $O(n)$ time.

We only know that a node must be less than or equal to its children. Assume that we can compare in constant time two children $y, z$, and decide which subtree to traverse first. But we still cannot know if children of $y$ are greater than children of $z$, or vice versa. So there is no rule stating the relationship between children. Thus, we cannot use an inorder tree walk to print the keys in sorted order in $O(n)$ time. Linear sorts such as radix sort only work on integers. Therefore, in general we cannot use heap property to print out the keys in sorted order in $O(n)$ time.

12.2-4
A counter-example: If we search 3 from the following tree, then $A = \{\}$, $B=\{3,6,15\}$, $C=\{7\}$. Let $b=15$ and $c=7$. But $b > c$ (contradictory).

12.2-7
[The key is to show that each node will be visited at most 3 times]

Let $x$ be a node. Consider the following case:

Case 1: $x$ must be on the simple path from the root of a subtree to the local minimum, because $x$ will be visited by the first TREE-MINIMUM or the TREE-MINIMUM in TREE-SUCCESSOR.

Case 2:
If $x$ has a left subtree $L$, then the minimum $m$ of the subtree rooted at $x$ is in $L$. The algorithm will return $m$, then the successor of $m$, and so on until the predecessor $p$ of $x$ is returned. When TREE-SUCCESSOR($p$) is called, $x$ will be visited and returned.

Case 3:
If $x$ has a right subtree, then the successor of $x$ is in the right subtree. Then the inorder walk will return all nodes in the right subtree. After that, TREE-SUCCESSOR will visit $x$ again in order to find the next successor (which may be parent($x$)).

As a result, node $x$ will be visited at least once and most 3 times by the algorithm.

13.1-5
By lemma 13.1, a red-black tree with $n$ internal nodes has height at most $2\log(n+1)$. Thus, the longest simple path from a node $x$ to a descendant leaf is at most $2\log(n+1)$.

Let $h$ be the height of the tree. According to red-black tree property 3 – if a node is red, then both its children are black, at least half of the nodes on any simple path from the root to a leaf, not including the root, must be black.
Thus, $h/2 \leq bh(x)$, where $h \leq 2\lg(n+1) \Rightarrow bh(x) \geq \lg(n+1)$ = the shortest path is the path with only black nodes. Therefore, the longest simple path is at most twice that of the shortest simple path.

13.3-2

(Sentinels will not be shown)

#1

A square represents a black node

#2

A circle represents a red node

#3

(Case 3)

#4

(Case 1)

#5

(Case 2, then Case 3)

#6

(Case 1)
The resulting red-black tree is not always the same. Consider the following case: