CPSC 331 — Solutions for Tutorial Exercise #10

Additional Operations on Binary Search Trees

1. This question concerned the `addToArray` method included in the tutorial exercise. This algorithm is given again, in these solutions, in Figure 1 on page 2.

   (a) You were first asked to *sketch a proof* of the correctness of this algorithm. Information about the required information was included in the tutorial exercise.

   Solution: We are trying to prove the following

   > **Claim:** If the "Binary Search Tree" invariant is initially satisfied, and the `addToArray` algorithm is executed with
   >
   > - a (possibly `null` node `x` in this binary search tree, and
   > - an `ArrayList A`, storing values with the same type as are stored in this binary search tree,
   >
   > then the execution of this algorithm eventually halts. Furthermore, when the execution halts, all the elements stored in the subtree with root `x` have been added to `A`, *in increasing order*.

   This claim can be proved by *mathematical induction*. One can use induction on either the ***size*** or ***depth*** of the subtree with root `x`. In either case, the *strong* form of mathematical induction should be used.

   In the *basis* you should consider the case that the subtree with root `x` has size 0 (when inducting on size) or depth -1 (when inducting on depth). In either case, the node `x` must be `null` so that the "subtree with root `x`" is the empty tree and the `ArrayList A` should not be changed.

   One can now see by inspection of the code that this is the case: The test at line 1 fails and the execution ends immediately after that without any changes having been made at all.

   If you are inducting on size then you should introduce an integer k such that $k \geq 0$. The Inductive Hypothesis will assert that the conditions in the claim hold if the subtree of `x` has size h for any integer h such that $0 \leq h \leq k$. The Inductive Claim, to be proved, will be that these conditions are also satisfied if the subtree of `x` has size $k + 1$.

```
private void addToArray(BSTNode x, ArrayList<E> A) {
1.  if (x != null) {
2.    addToArray(x.left, A);
3.    A.add(x.element);
4.    addToArray(x.right, A);
    }
}
```

Figure 1: Implementation of `addToArray`

If you are inducting on depth, instead, then you should introduce an integer k such that $k \geq -1$. The Inductive Hypothesis will concern the execution of the algorithm when the *depth* of the subtree with root x has *depth* h, for any integer h such that $-1 \leq h \leq k$, and the Inductive Claim, to be proved, will be that these conditions also hold if the subtree of x has *depth* $k + 1$.

The proof will continue by considering the execution of the algorithm when the subtree with root x has size (respectively, depth) $k + 1$. In each case x cannot be `null`, so that the subtree with root x is nonempty and the `ArrayList A` should be changed. One can see by inspection of the code that the test at line 1 is now passed, so that the steps at lines 2, 3 and 4 are executed.

Parts 2 and 3 of the "Binary Search Tree Invariant" are now useful: It follows by Property #3 that the subtree with root x is also a binary search tree, so that it satisfies Property #2: All of the nodes in the *left* subtree of the subtree with root x store values that are *less than* the value stored at x, and all the nodes of the *right* subtree of the subtree with root x store values that are *greater than* x. It follows that all the elements stored in the subtree with root x have been added in increasing order if they have been added as follows:

a) First, all the elements stored in the left subtree were added, in increasing order.

b) Then, the element stored at the root (that is, at x) was added.

c) Finally, all the elements stored in the right subtree were added in increasing order.

Note that the left and right subtrees of the subtree with root of x both have sizes (respectively, depths) that are strictly ***less than*** the size (respectively, depth) of the subtree with root x.

- It now follows by the Inductive Hypothesis that the execution of the step at line 2 eventually ends, and all the elements stored in the left subtree of the subtree with root x have been added to `A`: Part (a) of the above has been carried out.
- One can see by inspection of the code that the execution of the step at line 3

also eventually ends, and the element stored at `x` has been added to `A`: Part (b) of the above has been carried out.

- It also follows by the Inductive Hypothesis that the execution of the step at line 4 eventually ends, and all the elements stored in the right subtree of the subtree at root `x` have been added to `A`: Part (c) of the above has been carried out.

The computation now ends — and all the elements stored in the subtree with root `x` have been added to `A` as needed to complete the Inductive Step and complete the proof.

(b) $T(n)$ was then defined to be the number of steps used by this algorithm — defined using the "uniform cost criterion" and ***not*** including steps used by `A`'s `add` method — when this algorithm is executed and the subtree with root `x` has size n.

It was claimed that, for every non-negative integer n,

$$T(n) \leq \begin{cases} 1 & \text{if } n = 0, \\ \max\limits_{0 \leq k \leq n-1} (T(k) + T(n-k-1) + 4) & \text{if } n \geq 1. \end{cases} \tag{1}$$

You were asked to use this to prove that $T(n) \leq 5n + 1$ for every non-negative integer n.

Solution: We wish to prove the following

> **Claim:** If $T : \mathbb{N} \to \mathbb{N}$ is a function satisfying the recurrence shown at line (1) then $T(n) \leq 5n + 1$ for every non-negative integer n.

Proof: By induction on n. The strong form of mathematical induction will be used and the case that $n = 0$ will be considered in the basis.

Basis: Suppose that $n = 0$. Then

$$\begin{aligned} T(n) &= T(0) && \text{(since } n = 0\text{)} \\ &\leq 1 && \text{(by the above recurrence)} \\ &= 5n + 1 && \text{(since } n = 0\text{)} \end{aligned}$$

establishing that $T(n) \leq 5n + 1$ in this case.

Inductive Step: Let h be an integer such that $h \geq 0$. It is necessary and sufficient to use the following

> Inductive Hypothesis: If the function $T : \mathbb{N} \to \mathbb{N}$ satisfies the above recurrence then $T(i) \leq 5i + 1$ for every integer i such that $0 \leq i \leq h$.

to prove the following

> Inductive Claim: If the function $T : \mathbb{N} \to \mathbb{N}$ satisfies the above recurrence $T(h + 1) \leq 5(h + 1) + 1$.

If the function $T : \mathbb{N} \to \mathbb{N}$ *does not* satisfy the above recurrence then there is nothing to prove (because this is an empty claim). Suppose, therefore, that it does. Then, since $h \geq 0$, $h + 1 \geq 1$, and it follows by this recurrence that

$$T(h+1) \leq \max_{0 \leq k \leq h} (T(k) + T((h+1) - k - 1) + 4). \tag{2}$$

Consider any integer $0 \leq k \leq h$. It now follows, by the Inductive Hypothesis, that

$$T(k) \leq 5k + 1.$$

Since $0 \leq k \leq h$, $(h+1) - k - 1 = h - k$ is an integer between 0 and k (inclusive) as well, and it also follows by the Inductive Hypothesis that

$$T((h+1) - k - 1) = T(h - k) \leq 5(h - k) + 1.$$

Thus

$$\begin{aligned} T(k) + T((h+1) - k - 1) + 4 &= (5k + 1) + 5(h - k) + 1 + 4 \\ &= 5h + 6 \\ &= 5(h+1) + 1. \end{aligned}$$

Since this is true for every integer k such that $0 \leq k \leq h$,

$$\begin{aligned} T(h+1) &\leq \max_{0 \leq k \leq h} (T(k) + T((h+1) - k - 1) + 4 && \text{(by the equation at line 2)} \\ &\leq \max_{0 \leq k \leq h} (5(h+1) + 1) && \text{(by the above bound)} \\ &= 5(h+1) + 1 \end{aligned}$$

as required to establish the Inductive Claim and complete the proof. □

(c) It follows that — excluding steps used by `A`'s `add` method — the total number of steps used by the `makeArray` method is at most linear in the size of this binary search tree (which is the size of the finite set it represents).

You were next asked to apply a result established earlier in this course to establish that the total number of steps used by `A`'s `add` method, during this application of `makeArray`, is at most linear in the size of this binary search tree, as well.

Solution: Note that the `ArrayList A` is initially empty. If the binary search tree has size N then (because no other `ArrayList` operations are being applied, and its final size is N), a sequence of N `add` operations have been applied — and, again, no other `ArrayList` operations have been applied, at all.

It follows by results discussed during Lecture #7 — and proved in the supplemental document concerning the ***amortized cost*** of a sequence of `ArrayList` operations

— that the total number of steps used by the `ArrayList` `add` operations is in $O(N)$, as claimed.

Note: It ***does not*** follow that the total number of steps used by `A`'s `ArrayList` operation is always at most linear in the size of the subtree whose root is the input node `x`: This node might be a leaf, so that the size of this subtree is one. However, the single `add` operation that is used as part of this might require the `capacity` of the `ArrayList` to be expanded — so that material must be copied from one underlying array to another — and the number of steps used by this `add` operation might be as high as linear in the entire binary search tree.

(d) It follows that the total number of steps used by the `MakeArray` operation is always at most linear in the size of the binary search tree.

You were asked to explain why this must always be *at least* this large — indeed, greater than or equal to the size of this tree — as well.

Solution: If the binary search tree has size N then N of `A`'s `add` operations are needed to increase its size from zero to N, and each of these certainly has cost at least one when the "uniform cost criterion" is used to define this.