2. Not My Cup of Tree

(a) Base Case: U = empty

values(makeBst(nil))	
= values(empty)	Definition of makeBst
= nil	Definition of values

(b) Inductive Step: $U = \text{cons}(a, L)$

values(makeBst(cons(a, L)))

= values(node(b, makeBst(P), makeBst(R)))	Def makeBST
. where $(P, S) = \text{split}(m, \text{cons}(a, L))$, $m = \lfloor \text{len}(\text{cons}(a, L))\ /\ 2 \rfloor, S = \text{cons}(b, R)$	
= concat(values(makeBst(P), cons(b, values(makeBst(R)))))	Definition of value
= concat(P, cons(b, values(makeBst(R)))	Inductive Hypothesis*
= concat(P, cons(b, R))	Inductive Hypothesis*
= concat(P, S)	since $S = \text{cons}(b, R)$
= cons(a, L)	Lemma 1

* Justification that the Inductive Hypothesis applies to to the lists P and R: Note that $m = \lfloor \text{len}(\text{cons}((a, L)/2 \rfloor$ (according to the values defined above), and $n = \text{len}(U) = \text{len}(\text{cons}(a, L))$, so $m \leq \text{len}(U)$, so Lemma 1 applies.

$\text{len}(P) = m$	Lemma 1
$= \lfloor \text{len}(\text{cons}(a, L)/2 \rfloor$	Substitution for m
$\leq \text{len}(\text{cons}(a, L))$	Arithmetic

$\text{len}(R) = \text{len}(S) - 1$	Def of len (since $S = \text{cons}(b, R)$)
$= \text{len}(\text{cons}(a, L)) - m - 1$	Lemma 1)
$\leq \text{len}(\text{cons}(a, L))$	Arithmetic ($m \geq 0$

So since len(P) ≤ len(cons(a, L)) and len(R) ≤ len(cons(a, L)), we can apply the inductive hypothesis to both these lists.

(c) Prove len(S) $\geq$ 1

$$\begin{aligned}
\text{len}(S) &= \text{len}(L) - m && m \leq \text{len}(L), \text{ Property of split} \\
&= \text{len}(L) - \lfloor \text{len}(L)/2 \rfloor && m = \lfloor \text{len}(L) \ / \ 2 \rfloor \\
&\geq 1 - \lfloor \text{len}(L)/2 \rfloor && \text{len}(L) \geq 1 \\
&\geq 1 - 0 && \lfloor \text{len}(L)/2 \rfloor \geq 0 \\
&\geq 1 && \text{Arithmetic}
\end{aligned}$$

4. Many More Fish in the Tree

(a) Claim: Define $P(L)$ to be that contains(a, concat(L, S)) = contains(a, L) or contains(a, S) where a is any integer and S is any List. I will prove the claim by structural induction on L.

Base Case: nil

$$\begin{aligned} \text{contains}(a, \text{concat}(\text{nil}, S)) &= \text{contains}(a, S) && \text{Definition of concat} \\ &= \text{false or contains}(a, S) && \text{Identity} \\ &= \text{contains}(a, \text{nil}) \text{ or contains}(a, S) && \text{Definition of contains} \end{aligned}$$

Inductive Hypothesis: Suppose that $P(B)$ holds for an arbitrary list B. I.e., suppose that contains(a, concat(B, S)) = contains(a, B) or contains(a, S)

Inductive Step: We need to show that $P(\text{cons}(r, B))$ holds for any integer r.

$$\begin{aligned} &\text{contains}(a, \text{concat}(\text{cons}(r, B), S)) \\ &= \text{contains}(a,\ \text{cons}(r,\ \text{concat}(B, S))) && \text{Def of concat} \\ &= (a = r) \text{ or contains}(a,\ \text{concat}(B, S)) && \text{Def of contains} \\ &= (a = r) \text{ or contains}(a, B) \text{ or contains}(a, S) && \text{Inductive Hypothesis} \\ &= \text{contains}(a,\ \text{cons}(r, B)) \text{ or contains}(a, S) && \text{Def of contains} \end{aligned}$$

Conclusion: $P(L)$ holds for any list L by structural induction

(b) Claim: Define $P(U)$ to be the claim that contains(a, values(U)) = lookup(a, U) where a is any integer and U is any BST. I will prove the claim by induction.

Base Case: empty

$$\begin{aligned}
\text{contains}(a,\ \text{values}(\text{empty})) &= \text{contains}(a,\ \text{nil}) && \text{Definition of values} \\
&= \text{false} && \text{Definition of contains} \\
&= \text{lookup}(a,\ \text{empty}) && \text{Definition of lookup}
\end{aligned}$$

Inductive Hypothesis: Suppose that $P(K)$ holds for an arbitrary BST K. I.e., contains(a, values(K)) = lookup(a, K) where a is any integer.

Inductive Step: We need to show $P(\text{node}(b, S, T))$ where b is an arbitrary integer and S and T are arbitrary BSTs.

$$\begin{aligned}
&\text{contains}(a,\ \text{values}(\text{node}(b, S, T))) \\
&= \text{contains}(a,\ \text{concat}(\text{values}(S),\ \text{cons}(b,\ \text{values}(T)))) && \text{Def of values} \\
&= \text{contains}(a,\ \text{values}(S)) \text{ or contains}(a,\ \text{cons}(b,\ \text{values}(T))) && \text{Part A} \\
&= \text{contains}(a,\ \text{values}(S)) \text{ or } (a = b) \text{ or contains}(a,\ \text{values}(T)) && \text{Def of contains}
\end{aligned}$$

a can either be less than, greater than, or equal to b, so there are three cases to check.

Case 1: $a < b$

$$\begin{aligned}
&\text{contains}(a,\ \text{values}(S)) \text{ or } (a = b) \text{ or contains}(a,\ \text{values}(T)) && \text{Def of contains} \\
&= \text{contains}(a,\ \text{values}(S)) \text{ or False or contains}(a,\ \text{values}(T)) && \text{Since } a < b \text{, so } a \neq b \\
&= \text{contains}(a,\ \text{values}(S)) \text{ or False or False} && \text{BST Invariant} \\
&= \text{contains}(a,\ \text{values}(S)) && \text{Identity} \\
&= \text{lookup}(a, S) && \text{Inductive Hypothesis} \\
&= \text{lookup}(a,\ \text{node}(b, S, T)) && \text{Definition of lookup, since } a < b
\end{aligned}$$

Case 2: $a > b$

$$\begin{aligned}
&\text{contains}(a,\ \text{values}(S)) \text{ or } (a = b) \text{ or contains}(a,\ \text{values}(T)) && \text{Def of contains} \\
&= \text{contains}(a,\ \text{values}(S)) \text{ or False or contains}(a,\ \text{values}(T)) && \text{Since } a > b \text{, so } a \neq b \\
&= \text{False or contains}(a,\ \text{values}(T)) \text{ or False} && \text{BST Invariant} \\
&= \text{contains}(a,\ \text{values}(T)) && \text{Identity} \\
&= \text{lookup}(a, T) && \text{Inductive Hypothesis} \\
&= \text{lookup}(a,\ \text{node}(b, S, T)) && \text{Definition of lookup, since } a > b
\end{aligned}$$

Case 3: $a = b$

contains(a, values(S)) or ($a = b$) or contains(a, values(T))	Def of contains
= contains(a, values(S)) or True or contains(a, values(T))	Since $a = b$
= True	Domination
= lookup(a, node(b, S, T))	Definition of lookup, since $a = b$

Conclusion: Since these three cases in the inductive step are exhaustive, we have shown that the claim holds for an arbitrary BST $K = \text{node}(b, S, T)$. Therefore, we have also proven that $P(U)$ holds for any BST U by structural induction.

5. Chomping at the Split