Problem 1: Master Method Problem

(a) T(n) = $3T(\frac{n}{9}) + n$
Let a = 3, b = 9, and f(n) = n. Then $\log_9 3 = \frac{1}{\log_3 9} = \frac{1}{2}$.
Then, notice that the following inequality is:

$$f(n) = n > n^{\frac{1}{2}}$$

Therefore,

$$f(n) = n = \Omega(n^{\frac{1}{2}+\epsilon})$$

where $\epsilon = \frac{1}{2}$
Furthermore, check for regularity then $af(\frac{n}{b}) \leq cf(n)$ then

$$3f(\frac{n}{9}) \leq cf(n)$$

implies that $\frac{1}{3} \leq c, \forall n$.
Then by case 3 of Master Method the run-time is T(n) = $\Theta(n)$.

(b) T(n) = $7T(\frac{n}{3}) + n$
Let a = 7, b = 3 and f(n) = n.
Then for $\log_b a$ notice how

$$\log_b a = \log_3 7 > \log_3 3 = 1$$

and that

$$\log_b a = \log_3 7 < \log_3 9 = 2$$

then the following implication can be made

$$f(n) = n < n^{\log_3 7}$$

Then from Case 1 of Master Method it follows that for and $\epsilon \approx 0.7712$ then

$$f(n) = n = O(n^{\log_3 7 - \epsilon})$$

then T(n) = $\Theta(n^{\log_3 7})$ from Case 1 of Master Method.

(c) Let a = 2, b = 4, and $f(n) = \sqrt{n} = n^{\frac{1}{2}}$.
Then for $\log_b a$ notice how

$$\log_b a = \log_4 2 = \frac{1}{\log_2 4} = \frac{1}{2}$$

Then from master method case 2, since

$$f(n) = n^{\frac{1}{2}} = \Theta(n^{\frac{1}{2}})$$

then T(n) = $\Theta(n^{\frac{1}{2}} \log n)$

Problem 2 Solving a Recurrence

To solve the given, master method cannot be applied, therefore, we use guess and substitute method.
Let us take the change of variables to do our work. Let that

$$T(2^m) = S(m)$$

then with the given

$$T(n) = T(\sqrt{n}) + c$$

then

$$T(n) = T(\sqrt{2^m}) + C$$

which can be rewritten as

$$S(m) = S(\frac{m}{2}) + c$$

then let a = 1 and b = 2 then from using master method it is case two on in terms of S(m) then

$$S(m) = \Theta(\log(m))$$

And from the change of variables then

$$T(n) = \Theta(\log(\log(n))$$

Then let our guess be T(n) = c(log(log(n)) for a constant c. Then use induction to prove whether our guess is correct.
This fails the base step as

$$T(2) = c(\log(\log(2)) = 0 \neq c$$

Let us make a new guess of $c\log(\log(n)) + c$
Base Case
T(2) = $c\log(\log(2)) + c = c$, therefore, the base case is met.
Induction

$$T(\sqrt{(}n)) + c = c(\log(\log(\sqrt{n}))) + 2c$$

then using the exponent rules of logs then it can be further rewritten as

$$= c(\log(\frac{1}{2}\log(n))) + 2c$$

Then using the product rule of logs then rewrite it as

$$= c(\log(\frac{1}{2}) + \log(\log(n)) + 2c$$

Since $\log(\frac{1}{2}) = \log(0.5) = -1$ then

$$-c + c\log(\log(n)) + 2c = c\log(\log(n) + c$$

From induction, our guess was proved to be correct.

Problem 3 Binary Search Tree Proof

(a) **Prove by contradiction that the successor S cannot be an ancestor or cousin of X, so S must be in a subtree rooted at X.**
Suppose that S is a successor of X and that X has two children. Since S is a successor then by definition it implies that S > X. To prove the statements let us break it into case statements.

- **Case 1**
  Assume that S has an ancestor of X. Then by assumption, since S is an ancestor of X then S must be in the subtree of S.left by the definition of BST.
  From the initial assumption, X has two children so then X < X.right from definition of BST. The this implies that $X <$ X.right < S which creates a contradiction since there should not be any between numbers in the BST of X and S.
- **Case 2**
  Assume that S is a cousin of X. Then by that assumption, there exists a grandparent **G** since they are in different subtrees. Then that implies that one is greater and the other is less than G. From S being a successor of X, then $S > G > X$ which causes a contradiction of S being a successor.

Therefore, from our two case statements then X cannot be an ancestor nor cousin.

(b) **Identify and prove the subtree of X that successor S must be in**
From part (a), S is a subtree rooted at X and from the initial statement of (a), S is a successor of X which implies that $S > X$.
For the purpose of contradiction let us e assume that S is located at the left sub-tree of X. Then by the definition of a BST that implies $X > S$ which creates a contradiction as the initial assumption was that $S > X$.
Therefore, from the definition of BST then S must be located on the right sub-tree of X.

(c) **Show by contradiction that successor S cannot have a left child.**
Suppose that the assumption was that S had a left child Y. In part (b) it was proven that S is in the right child of X. Then by definition of BST it implies that $X < Y < S$ then this causes a contradiction because this implies that S is not a successor of X.
Hence, S cannot have a left child.

(d) **Indicate how this proof would be changed for the predecessor.**
Suppose that S is a predecessor of X such that X has two children. Then this implies that $X > S$.
Proof that S is not an ancestor of X
For the sake of contradiction assume that S is an ancestor of X. Then this implies that X is in the sub-tree of S.right.
From the initial assumption, X has two children then $X > X.left$ from the definition of BST. Then this implies that S $< X.left < X$. However, this causes a contradiction to the definition of a predecessor since there should not exists anything between X and

S.

Proof that S is not a cousin of X

Suppose that S is a cousin of X. Then by the assumption, ∃ grandparent **G** since they are located in different sub-trees. Then this implies that one is greater and one is less compared to G. Then $S < G < X$, by the inital assumption of $S < X$. However, this causes a contradiction of being a predecessor of a BST as there should not exists between S and X.

Hence, X and S are not cousins.

Proof that subtree of X that predecessor S must be in

For the sake of contradiction let us assume that S is located in the right sub-tree of X. Then from the definition of BST, then it implies that $S > X$. However, this causes a contradiction with the definition of a predecessor as $S < X$.

Therefore, if S is not located in the right sub-tree then it must be in the left sub-tree.

Proof that predecessor S cannot have a right child

For the sake of contradiction, let us assume that S has a right child Z. Previously, it was proven that S is located in the left of X. Then from the definition of BST then it implies that $S < Z < X$. This causes a contradiction because this implies that S is not a predecessor of X and rather Z is.

Thus, S cannot have a right child.

Problem 4: Deletion in BST

(a) **How does this code rely on the lemma you just proved in problem 3? Be specific, referring to line numbers. Be careful that you are using the actual lemma above, and not a similar fact proven elsewhere.**
At line 11, y.left is replaced with the sub-trees under z.left. From the lemma proved in problem (3), it tells that it is not overwritting any subtree y.left is empty.

(b) **When node z has two children, we arbitrarily decide to replace it with its successor. We could just as well replace it with its predecessor. (Some have argued that if we choose randomly between the two options we will get more balanced trees.) Rewrite Tree-Delete to use the predecessor rather than the successor Modify this code just as you need to and underline or boldface the changed portions.**

Algorithm 1 TREE-DELETE(T,z)

```
1: if z. left == NIL
2:     TRANSPLANT(T,z,z.right)
3: else if z.right == NIL
4:     TRANSPLANT(T, z, z.left)
5: else y = TREE-MAXIUMUM(z.left)
6: if y.p != z
7:     TRANSPLANT(T,y,y.left)
8:         y.left.p = y
9:         y.left.p = y
10:    TRANSPLANT(T,z,y)
11:    y.right = z.right
12:    y.right.p = y
```

Problem 5 Constructing a Balanced BST

(a) Write pseudocode (or Java if you wish) for an algorithm that constructs the tree and returns the root node. (We won't worry about making the enclosing BinaryTree class instance.) You will need to use methods for making a new TreeNode, and for setting its left and right children.

Algorithm 2 BINARY-BALANCED-BST(A, low, high)

```
1: if low > high // the sub-tree is empty
2:       return NIL
3: mid = ⌊(high + low)/2⌋ // take the average of high and low
4: leftTree = BINARY-BALANCED-BST(A, low, mid - 1)
5: rightTree = BINARY-BALANCED-BST(A, mid + 1, high)
6: return new TreeNode (A[mid], leftTree, rightTree)
```

(b) **What is the Θ time cost to construct the tree, assuming that the array has already been sorted? Justify your answer.**
Based on the algorithm removes an element in constant time and recursively calls the remaining. Furthermore, there a n elements to remove and from a mathematical standpoint n is greater than a constant so the run time in terms of Θ is $\Theta(n)$.

(c) **Compare the expected runtime of BinarySearch on the array to the expected runtime of BST TreeSearch in the tree you just constructed. Have we saved time?**
From ICS 211 and our textbook, recall that the expected run time for Binary Search is $O(\log n)$ and the run-time for a BST is also $O(\log n)$. Therefore, from the reasons we do not save any time.