VE281 Writing Assignment Four
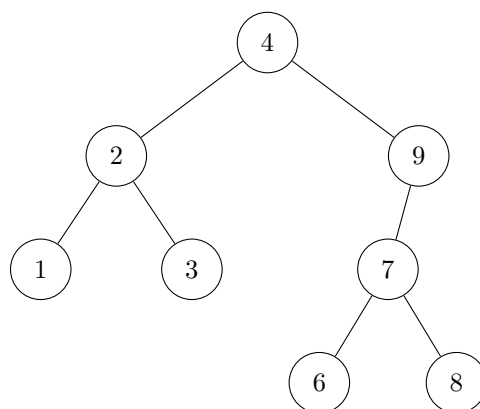
Liu Yihao 515370910207

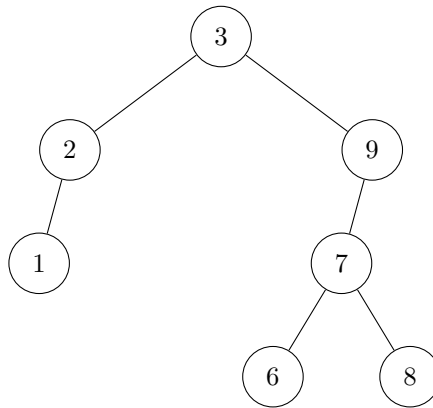## Ex. 1

(a)

```
         4
        / \
       2   9
      / \   \
     1   3   5
              \
               7
              / \
             6   8
```

(b)

```
         4
        / \
       2   9
      / \   \
     1   3   7
              / \
             6   8
```
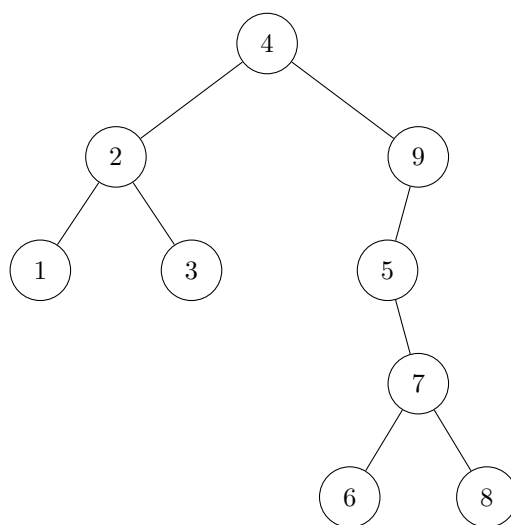
(c)

Ex. 2

Input: A non-empty binary tree with root node $root$
Output: Whether the tree is a binary search tree
  function EXAMINE($root$)
    if $root.left$ is empty **then**
        $flag \leftarrow true$
    else if $root.left.key > root.key$ **then**
        $flag \leftarrow false$
    else
        $flag \leftarrow$ EXAMINE($root.left$)
    end if
    if not $flag$ **then**
        return flag
    end if
    if $root.right$ is empty **then**
        $flag \leftarrow true$
    else if $root.right.key < root.key$ **then**
        $flag \leftarrow false$
    else
        $flag \leftarrow$ EXAMINE($root.right$)
    end if
    return $flag$
  end function

We know that if we want to ensure the tree is a binary search tree, at least every node should be verified once. In my algorithm, each node is verified exactly once, which should be a most runtime-efficient one. The runtime is $\Theta(n)$.

Ex. 3

```
node* getPredHelper(node* root, Key key, node* parent) {
    if (root == NULL) return NULL;
```

```
    if (root->key == key) {
        if (root->left) return findMax(root->left);
        return root;
    }
    node *temp = NULL;
    if (root->key < key) temp = getPredHelper(root->left, key, root);
    if (!temp) temp = getPredHelper(root->right, key, root);
    if (!temp) return NULL;
    if (temp->key == key) {
        if (!parent) return NULL;
        if (root->key < key) return root;
    }
    return temp;
}

node* getPred(node* root, Key key) {
    return getPredHelper(root, key, NULL);
}
```

Ex. 4