1. For each of the parts below, give a short answer (a few words or symbols, or perhaps a short sentence).

(a) We have used rotations in several of our BST insertion algorithms, including AVL Trees, Splay Trees, and Red-Black Trees. Explain why rotations are useful. Do not explain how any of these systems use rotations; instead explain what rotations accomplish that is so commonly useful.

(b) Explain why no sorting algorithm can have performance better than $O(n)$.

(c) Explain what a “stable sort” is, and why it might be desirable.
2. Explain, for each of these types of induction, the overall structure of a proof. Give examples of common base cases, and then explain (in English) what the inductive step must prove. That is, what does the Inductive Hypothesis assume? I’ve given you the answer for one type of induction as an example.

- Induction over the non-negative integers
- Structural induction over a BST
- Proving a loop invariant

**Example Solution**

**Induction over non-negative integers**

In induction over non-negative integers, we prove that a conjecture is true about all positive integers - or perhaps, about some interesting subset of them. The base case can be any value, but often is 0, 1, and/or 2. In the inductive step, we generally assume that “the conjecture holds for all values less than $n$” and then try to prove it that it also holds for $n$. 
3. Examine the following Java code. (You may recall that `ArrayList` is a class which works like an array, but which allows you to add new items as often as you like; it expands automatically.)

```java
int[] input = ... ;  // assume that there are no duplicate values

int    pivot;
ArrayList less = new ArrayList();
ArrayList more = new ArrayList();

pivot = input[0];
for (int i=1; i<input.length; i++)
    if (input[i] < pivot)
        less.append(input[i]);
    else
        more.append(input[i]);
```

Prove the following loop invariant about the `for()` loop: “When any iteration begins, all of the values in `less` are less than `pivot`.”
4. Define what a binary tree is. Then define how a binary search tree is different than a simple binary tree.

5. In each of the following trees, rotate right at the node X. Draw the new tree.

(a) 
```
X
/ 
B
/ 
A
```

(b) 
```
Z
/ \ 
X C
/ \ 
Q E D
```

(c) 
```
X
/ \ 
D A
/ \ 
F G
/ \ 
P S
```
6. Consider the following blocks of pseudocode; each represents the `insert()` method (or some noteworthy part of it) for some type of balanced tree. Identify the type of tree, and explain your answer.

(a) 
`insert(node, key)`
- insert new leaf using ordinary BST insert
- update the child-height fields in the parent node
- if child-heights now differ by 2
  - perform a rotation to rebalance the tree
- else
  - recurse to parent’s parent, updating child heights and looking for rebalance opportunities.

(b) 
`insert(node, key)`
- insert new leaf using ordinary BST insert
- perform rotations until the new leaf is at the root

(c) 
`insert(node, key)`
- if (node has 3 keys)
  - split node into two pieces
  - push middle key up into the parent node
  - insert(parent)
- else if (node is a leaf)
  - add key to node
- else
  - compare 'key' to the key(s) stored in 'node'
  - recurse into the proper child of node
7. The following tree contains a violation of the AVL property. Draw the tree as it will be after the AVL rebalance is performed. (You are not required to label each tree with its height, but you may if it helps you.)

```
E
/ \  
/   \ /
/    C G
/     / \ /
A     D  F  J
\    /   /
B    H  K  
  \  
   I
```
8. (20 points) Implement the ‘merge’ step of Merge Sort in Java. Merge the two arrays into a newly-allocated array; do not merge in place. Also, do not assume that the two arrays are the same length.

I have provided the method declaration and the first few lines:

```java
int[] mergeArrays(int[] left, int[] right)
{
    int[] retval = new int[left.length + right.length];

    int i = 0;  // num elements used from left
    int j = 0;  // num elements used from right
```
9. (10 points) Give a recursive method for insertion into a BST. The recursive method should take two parameters: a reference to the root of some subtree, and a key to insert. Do not include any rebalancing; simply insert it. You may use Java or pseudocode.

NOTES:

- Don’t worry about handling the case where the tree might be empty; assume that the node reference is non-null.
- You may assume that the key does not already exist in the tree.
- If you use Java, you may assume that the BST nodes use a type named TreeNode, and that the type of the key is int.
10. Use induction to prove the following conjecture:

\[ 4 \mid (5^n - 1), \ n \geq 0 \]

(Remember: \( a \mid b \) means “\( b \) is divisible by \( a \).”)