Problem 1 (25 pts).

Start with an empty Binary Search Tree.

(a) Insert the keys 20, 36, 24, 30, 10, 44, 26 into the tree, in the order given. Show the resulting Binary Search Tree.

(b) Using this tree, which comparisons would be made when searching for the key 24? Which comparisons would be made to search for 28?

(c) Label all nodes with their leftSize values. Use the tree you drew in part (a).

(d) Show the comparisons and rank values as you search for the element with index 3 in this Indexed Binary Search Tree.

(e) The element with value 20 is now to be removed. This can be done in two ways. Redraw the tree after the removal, including LeftSize values, for each of these two strategies.

\[
\begin{align*}
\text{(a)} & \quad 20 & \quad 36 & \quad 30 & \quad 10 & \quad 44 & \quad 26 \\
& \quad 0 & \quad 0 & \quad 1 & \quad 0 & \quad 0 & \quad 0 \\
& \quad 0 & \quad 0 & \quad 1 & \quad 0 & \quad 0 & \quad 0 \\
\end{align*}
\]

\[
\begin{align*}
(24) & \quad 24 > 20 \quad \text{go right} \\
& \quad 24 < 36 \quad \text{go left} \\
& \quad 24 = 24 \\
\end{align*}
\]

\[
\begin{align*}
(3) & \quad 3 > 1 \ (\text{at 20}), \text{so search for } 3 - 1 - 1 = 1 \ \text{in } 36 \\
& \quad 1 < 3 \ (\text{at 31}), \text{so search for } 1 \ \text{in } 24 \\
& \quad 1 > 0 \ (\text{at 24}), \text{so search for } 1 - 0 - 1 = 0 \ \text{in } 30 \\
& \quad 0 < 1 \ (\text{at 30}), \text{so search for } 0 \ \text{in } 26 \\
& \quad 0 = 0 \ (\text{at 26}) \quad \text{found it!}
\end{align*}
\]
WORKSPACE

1. Replace with inorder predecessor.
   ![Tree Diagram]

2. Replace with inorder successor.
   ![Tree Diagram]
Problem 2 (25 pts).

Draw the height biased max leftist tree that results when the two height biased max leftist trees below are melded by following their rightmost paths. Use the recursive strategy used in the meld algorithm of the text, and discussed in class. Show the steps used to arrive at the melded tree.

```
42
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>47</td>
</tr>
</tbody>
</table>

54
/ | |
| 49 | 51 |

32
---

69
/ |
| 54 |

60
/ |
| 49 |

47
/ |
| 51 |

52
---

57
---

63
---

58
---

44
---

45
---

24
---

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap

Swap
Problem 3 (25 pts).

Start with an empty AVL search tree and insert the following keys in the given order: 24, 34, 44, 19, 21, 26. Draw the trees following each insertion, and also after each rotation. Specify the rotation types, and the balancing factors.
Problem 4 (25 pts.)

Reverse-level-ordering is similar to "level ordering", except the levels are visited in reverse order, i.e. starting from the bottom. However, on each level, the nodes are still visited in the usual left-to-right order.

```
        A
       / \
      /   \
     B     C
    / \   / \
   D  E  F
   / \
  G
  / \
 H I
```

For this tree, the reverse-level-order traversal is H, I, G, D, E, F, B, C, A.

Below is the original algorithm for performing a level-order traversal of a binary tree. Re-write it to make it reverse-level-order.

```
t = root;
while (t != null) {
    visit(t);
    if (t->left != null) q.enqueue(t->left);
    if (t->right != null) q.enqueue(t->right);
    if (q.isempty()) t = null;
    else t = q.dequeue();
}
```

Hint: In addition to the queue, use a stack.

```
while (!s.isempty()) {
    visit(s.top());
    s.pop();
}
```

Alternative:

```
while (t != null) {
    s.push(t);
    if (t->right != null) q.enqueue(t->right);
    if (t->left != null) q.enqueue(t->left);
    }
```