1 Balanced Search Trees

(a) Convert the red-black tree into a 2-4 tree. (Solid nodes are black.)

```
          27
         /  \
        12   33
       /     /  \
      8     19   46
   /     /     /     /  \
  14     15      25      31
```

(b) Insert 13 into the resulting 2-4 tree. Assume that, if a node has 4 keys, we choose to push up the right of the 2 middle keys (so the 2nd key from the right).

(c) Convert the resulting 2-4 tree into a valid left-leaning red-black tree.
(d) Given a (2, 4) tree containing N keys, how would you obtain the keys in sorted order in worst case O(N) time? We don’t need actual code—pseudo code or an unambiguous description will do.

(e) If a (2,4) tree has depth h (that is, the leaves are at distance h from the root), what is the maximum number of comparisons done in the corresponding red-black tree to find whether a certain key is present in the tree?

2 Tries

(a) List the words encoded by the following trie.

(b) Draw the trie after inserting the words *indent*, *inches*, and *trie*.
3 Skip Lists

Draw the resulting skip list after adding the following numbers at the specified random height. Then highlight the links used to find 148.

<table>
<thead>
<tr>
<th>Number</th>
<th>41</th>
<th>48</th>
<th>59</th>
<th>77</th>
<th>40</th>
<th>131</th>
<th>148</th>
<th>54</th>
<th>139</th>
<th>179</th>
<th>43</th>
<th>128</th>
<th>161</th>
<th>189</th>
<th>170</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>