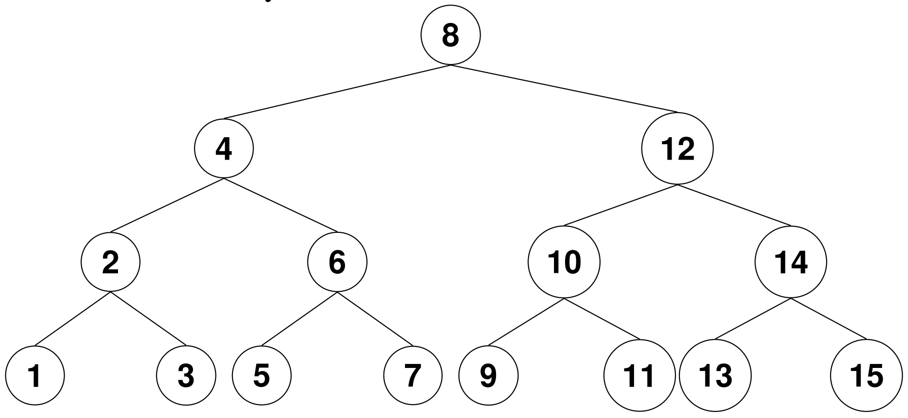
11.2 Applications of Trees

Binary Search Trees

A *binary search tree* is a binary tree with the following properties:

- Each vertex has a value called a key.
- The left subtree of a vertex contains only vertices with keys less than the vertex's key.
- The right subtree of a vertex contains only vertices with keys greater than the vertex's key.
- The left and right subtree must also be a binary search tree.

A binary search tree for the numbers 1 to 15.



Binary Search Tree Insert

An iterative algorithm to insert an item into a binary search tree.

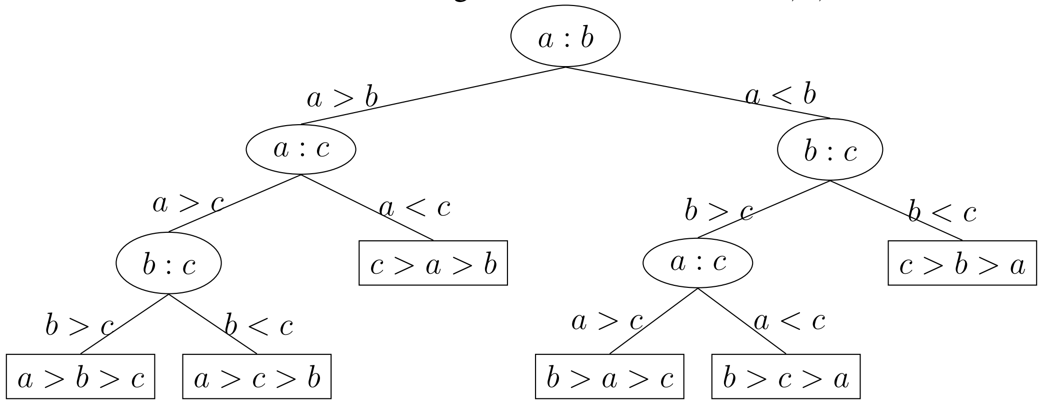
Algorithm $insert(T$: binary tree, x : item$)$

```
v = root of T
while v != null and v.label != x do
  if x < v.label then
    if v.leftchild != null then
      v = v.leftchild
    else
      add new vertex as a left child of v and set v = null
    end if
  else
    if v.rightchild != null then
      v = v.rightchild
    else
      add new vertex as right child of v and set v = null
    end if
  end if
end while
if root of T == null then
  add a vertex v to the tree and label it with x
else if v == null or v.label != x then
  label new vertex with x and let v be this new vertex
end if
return v{v is the location of x}
```

Decision Trees

A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a *decision tree*. The possible solutions of the problem correspond to the paths to the leaves of this rooted tree.

A decision tree for sorting three distinct elements a, b, and c.



Theorem 1

A sorting algorithm based on binary comparisons requires at least $\lceil \log n! \rceil$ comparisons.

Corollary 1

The number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is $\Omega(n \log n)$.

Theorem 2

The average number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is $\Omega(n \log n)$.

Prefix Codes

Encoding letters using varying number of bits is more efficient than using fixed-length bit strings

- Shorter bit strings for frequently occurring letters and longer bit strings for rarely occurring letters.
- Need to specify start and end of the bits for each letter.

Prefix codes encode letters so that the bit string for a letter never occurs as the first part of the bit string for another letter.

Huffman Coding

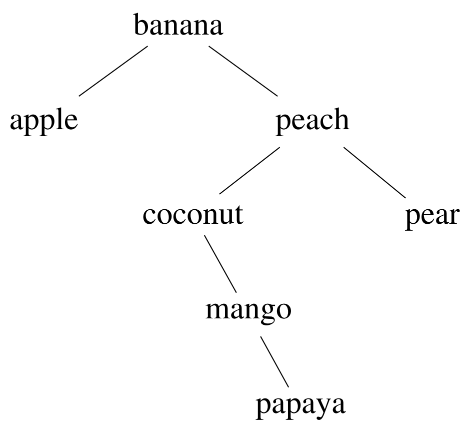
An algorithm that takes as input the frequencies of symbols in a string and produces an output a prefix code that encodes the string using the fewest possible bits, among all possible binary prefix codes for these symbols.

Algorithm *Huffman*(C: symbols a_i with frequencies w_i, $i = 1, \dots, n$)

```
1: $F$ = forest of $n$ rooted trees, each consisting of the single vertex $a_i$ and assigned weight $w_i$
2: while $F$ is not a tree do
3:     replace the rooted trees $T$ and $T'$ of least weighted from $F$ with $w(T) \geq w(T')$ with a tree
       having a new root that has $T$ as its left subtree and $T'$ as its right subtree. Label the new
       edge to $T$ with 0 and the new edge to $T'$ with 1.
4:     Assign $w(T) + w(T')$ as the weight of the new tree
5: end while
   {The Huffman coding for the symbol $a_i$ is the concatenation of the labels of the edges in the
   unique path from the root to the vertex $a_i$}
```

11.2 pg. 769 # 1

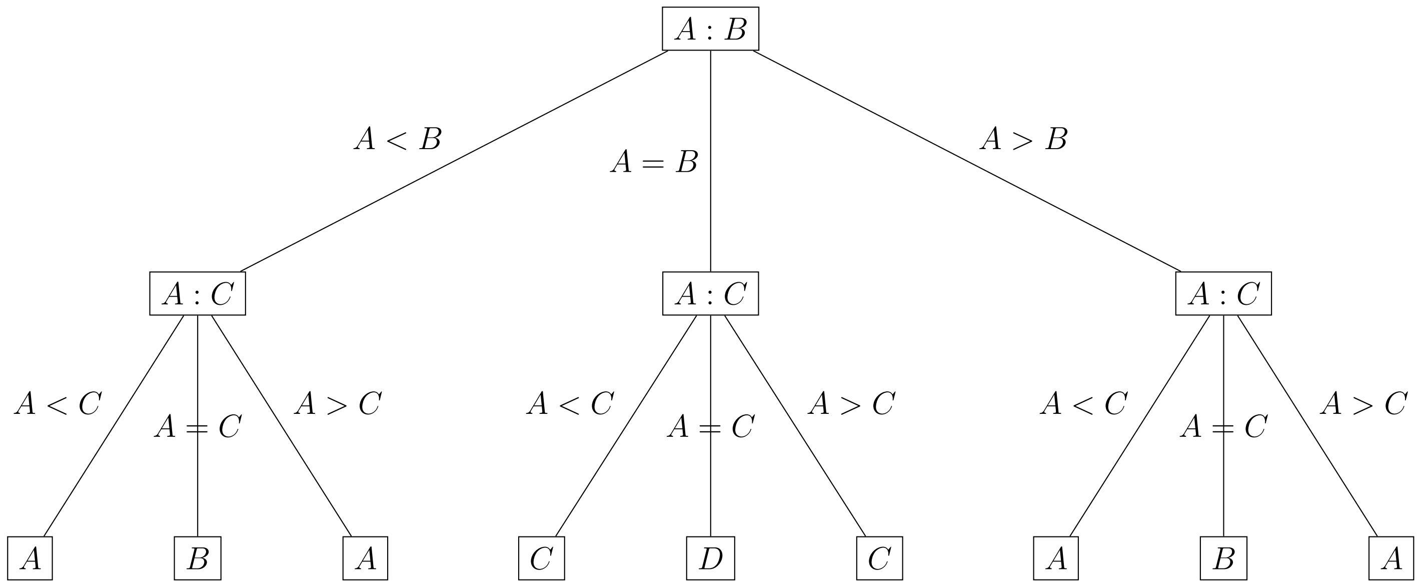
Build a binary search tree for the words *banana, peach, apple, pear, coconut, mango,* and *papaya* using alphabetical order.



11.2 pg. 769 # 7

How many weighings of a balance scale are needed to find a counterfeit coin among four coins if the counterfeit coin may be either heavier or lighter than the others?

We have 4 possible outcomes to this scenario and we know the decision tree is a 3-ary tree because our coins can weigh lighter, equal, or heavier. We know the height of the decision tree is at least $\lceil \log_3 4 \rceil = 2$, so we need at least two weighings. The decision tree for the four coins A, B, C, and D is as follows:



11.2 pg. 769 # 11

Find the least number of comparisons needed to sort four elements and devise an algorithm that sorts these elements using this number of comparisons.

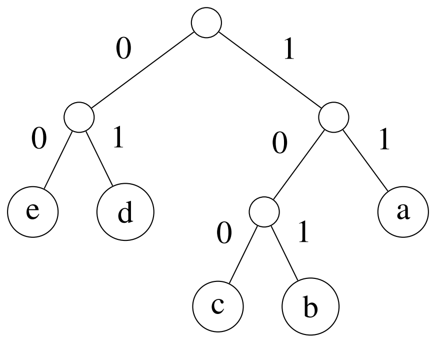
By Theorem 1, we know we need $\lceil \log 4! \rceil = 5$ comparisons. Suppose we have the elements a, b, c, and d that we need to sort. The algorithm is as follows:

First comparison: compare a and b. Assume that $a < b$.
Second comparison: compare c and d. Assume that $c < d$.
Third comparison: compare a and c. Assume that $a < c$.
Fourth comparison: compare b and c. Assume that $b > c$.
Fifth comparison: compare b and d. Assume that $b < d$.

Then our sort is as follows: a, c, b, d from least to greatest.

11.2 pg. 770 # 23

Use Huffman coding to encode these symbols with given frequencies: $a : 0.20, b : 0.10, c : 0.15, d : 0.25, e : 0.30$. What is the average number of bits required to encode a character?



Encoding:
a:11, b:101, c:100, d:01, e:00
The average number of bits is $2 \cdot 0.20 + 3 \cdot 0.10 + 3 \cdot 0.15 + 2 \cdot 0.25 + 2 \cdot 0.30 = 2.25$.