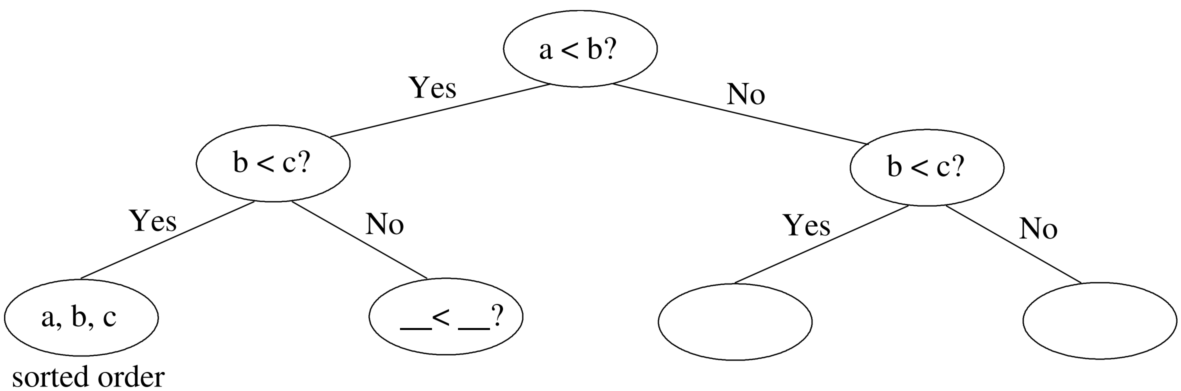
1) In Chapters 7 and 8 we switch from algorithm development (e.g., merge sort algorithm) and its analysis (e.g., $\Theta(n \log_2 n)$) to analyzing the *computation complexity of a problem* (e.g., the sorting problem).

Chapter 7: Computational Complexity for Sorting Problem
Goal: We what to determine a lower bound on the **best possible** sorting algorithm (for the worst-case problem instance -- data arrangement) that compares items.

<u>Result</u>: **best possible** sorting algorithm that compares items must do at least $\Theta(n \log_2 n)$) comparisons.
Important so:

- you do not waste your time looking for an $\Theta(n)$ sorting algorithm that compare items, and
- you think about sorts faster than $\Theta(n \log_2 n)$ that **do not** compare items.

To understand the computational-complexity argument for the sorting problem which compares items, consider 3 distinct items in variables: a, b, c.
a) Complete the decision tree to represent the necessary comparisons to determine the correct sorted order:



b) What do the leave nodes in the above decision tree for a sorting algorithm represent?

c) If we have n items to sort, how many leave nodes would be needed in the decision tree for any sorting algorithm?

d) What part of the decision tree represents the worst-case behavior for sorting?

e) To prove the result ("**best possible** sorting algorithm that compares items must do at least $\Theta(n \log_2 n)$) comparisons"), using the decision tree what are we going to have to show (proof)?

2. If m is the number of leaves in a binary tree (e.g., the decision tree), what must be the minimum depth (height) d of the binary tree? (Hint: consider the shape of the binary tree to minimize the height of the tree)

3. How might we make use of Stirlings approximation (stated below)?

$\log_2 n! = n \log_2 n - n/\ln 2 + (1/2)\log_2 n + O(1)$

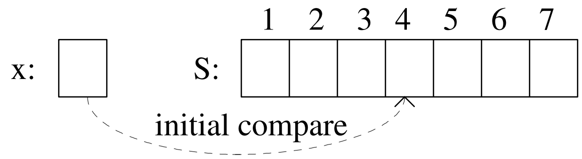
4. How might we get around this bound to do better than a $\Theta(n \log_2 n)$ sorting algorithm?

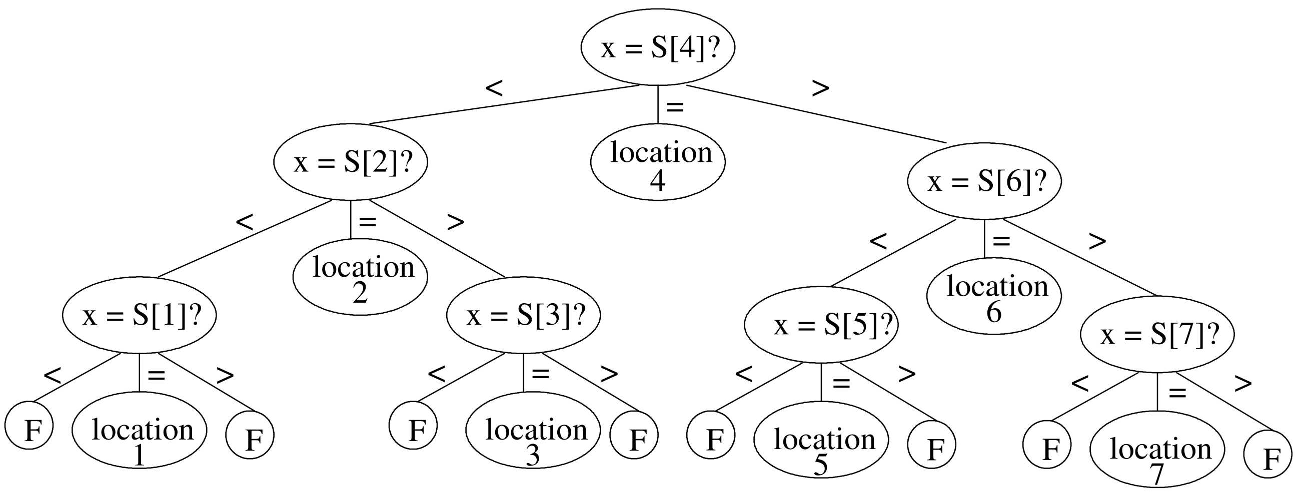
Chapter 8: Computational Complexity for Searching Problem

Goal: We what to determine a lower bound on the **best possible** searching algorithm (for the worst-case problem instance -- data arrangement) that compares items.

Result: **best possible** searching algorithm that compares items must do at least $\Theta(\log_2 n)$ comparisons.

For any searching algorithm, we can draw a decision tree. The following decision tree is for binary searching of an array S with 7 elements and a target of x.





5) Draw a similar decision tree for sequential search of unsorted array S with 7 elements and a target of x.

6) What are the general characteristics of decision trees for search algorithms that compare elements?

7) What part of the decision tree represents the worst-case behavior for searching?

8) To prove the result ("**best possible** searching algorithm that compares items must do at least $\Theta(\log_2 n)$ comparisons"), using the decision tree what are we going to have to show (proof)?

9) How might we get around this bound to do better than a $\Theta(\log_2 n)$ searching algorithm?