A tree is an acyclic connected graph. In a tree, we designate one node (any node) to be the root node. This is conventionally drawn at the top of the tree: all other nodes are measured by their distance from the root.

1. Because a tree is connected we know there is a path from every node to the root.
2. Because a tree is acyclic we know that there is only one path to the root.
3. Note that an acyclic disconnected graph is known as a forest: it is composed of many trees.
4. The parent of a given node v is the node that is connected directly to v but is closer to the root. (Thus the root has no parent).
5. u is a parent of v if and only if v is a child of u.
6. If a node has no children, it is called a leaf node.

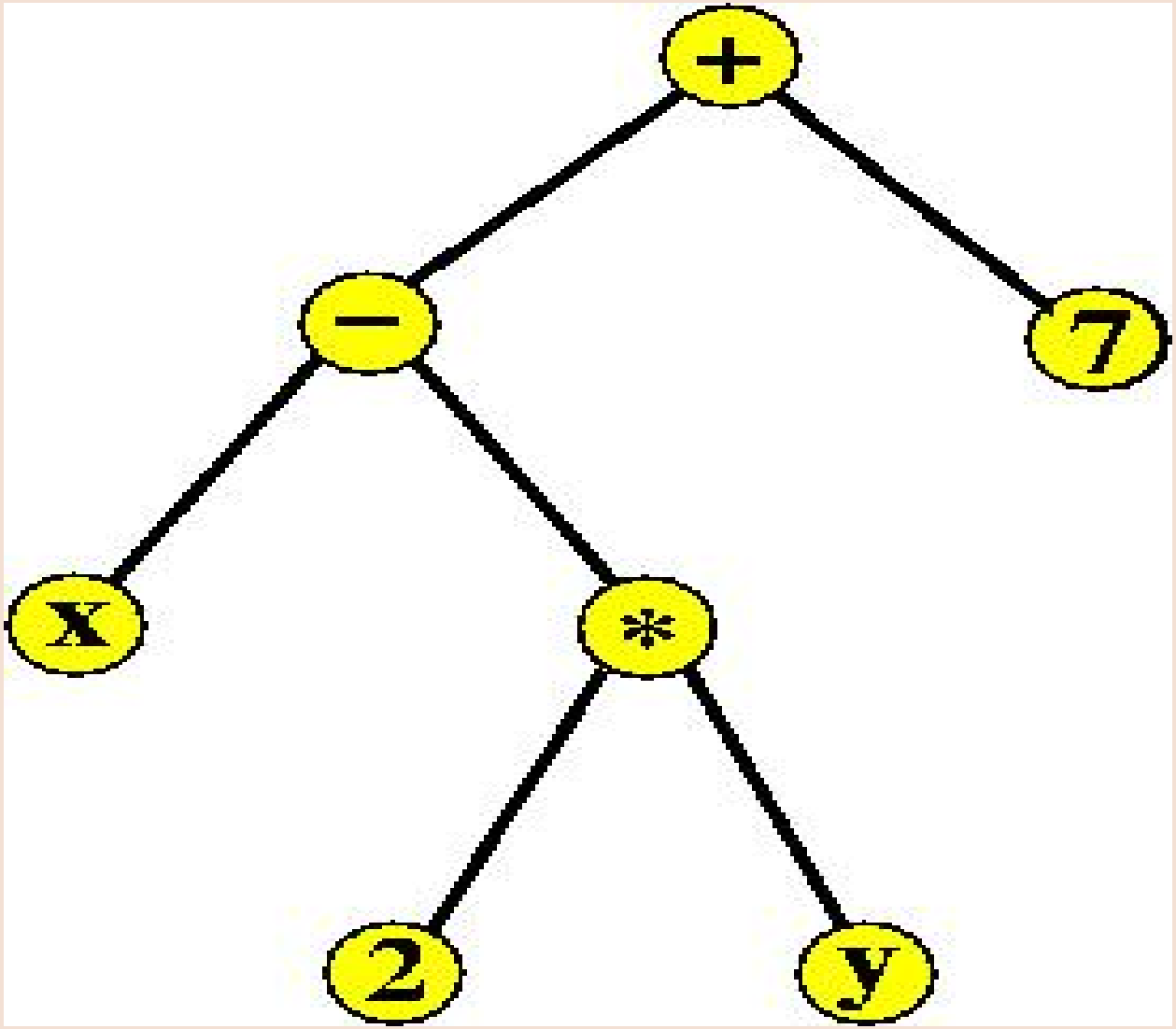
7. The distance between two nodes u and v on a tree is the length of the (unique) path between them (i.e. the number of edges in that path).
8. A node is at level n if its distance from the root is n.
9. The height (or depth) of a node is its distance from the root node.
10. The height (or depth) of a tree is the maximum depth or height of any of its nodes.
11. An n-ary tree is one where each node has a maximum of n children. For n=1, this is called a unary tree, for n=2 a binary tree, for n=3 a ternary tree.
12. A complete n-ary tree is one where at each level we have the maximum possible number of nodes (i.e. n^L where L is the level).

Example: Expression Tree

In a simple mathematical expression such as $a + b$, we call a and b the operands and the sign $+$ the operator. Here $+$ is a binary operator, because we add two things together: we can view $+$ as a "black box" with two inputs and one output.
A general mathematical expression (involving $+, -, \times, /, \ldots$) can be represented as a binary tree: In such a tree, the operands are the leaves (not necessarily at the same level), while the operators are internal nodes. Since (standard) mathematical operators are binary, these trees are (normally) binary.

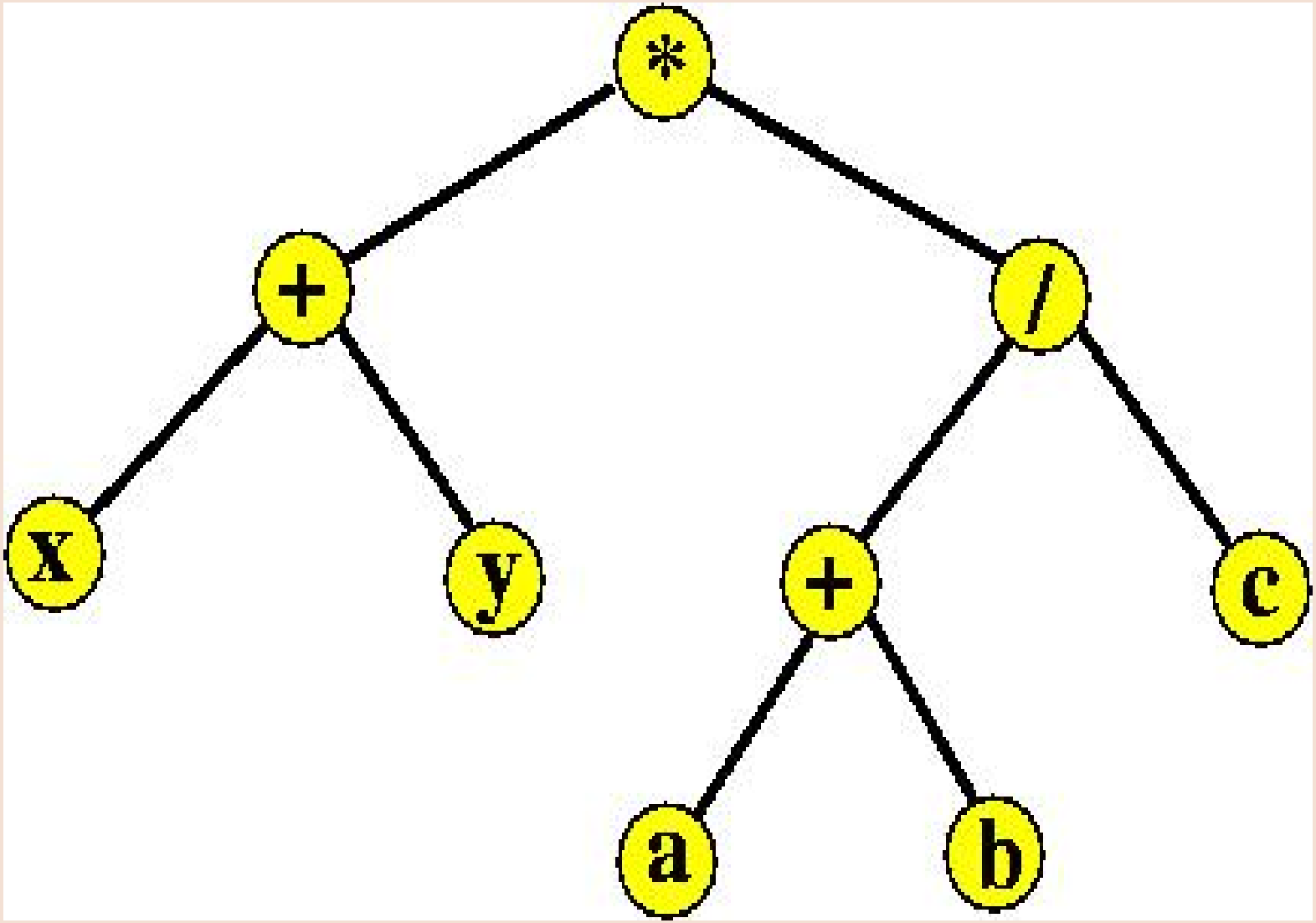
Example: Expression Tree

$x - 2y + 7$



Example: Expression Tree

$(x + y) \times ((a + b)/c)$



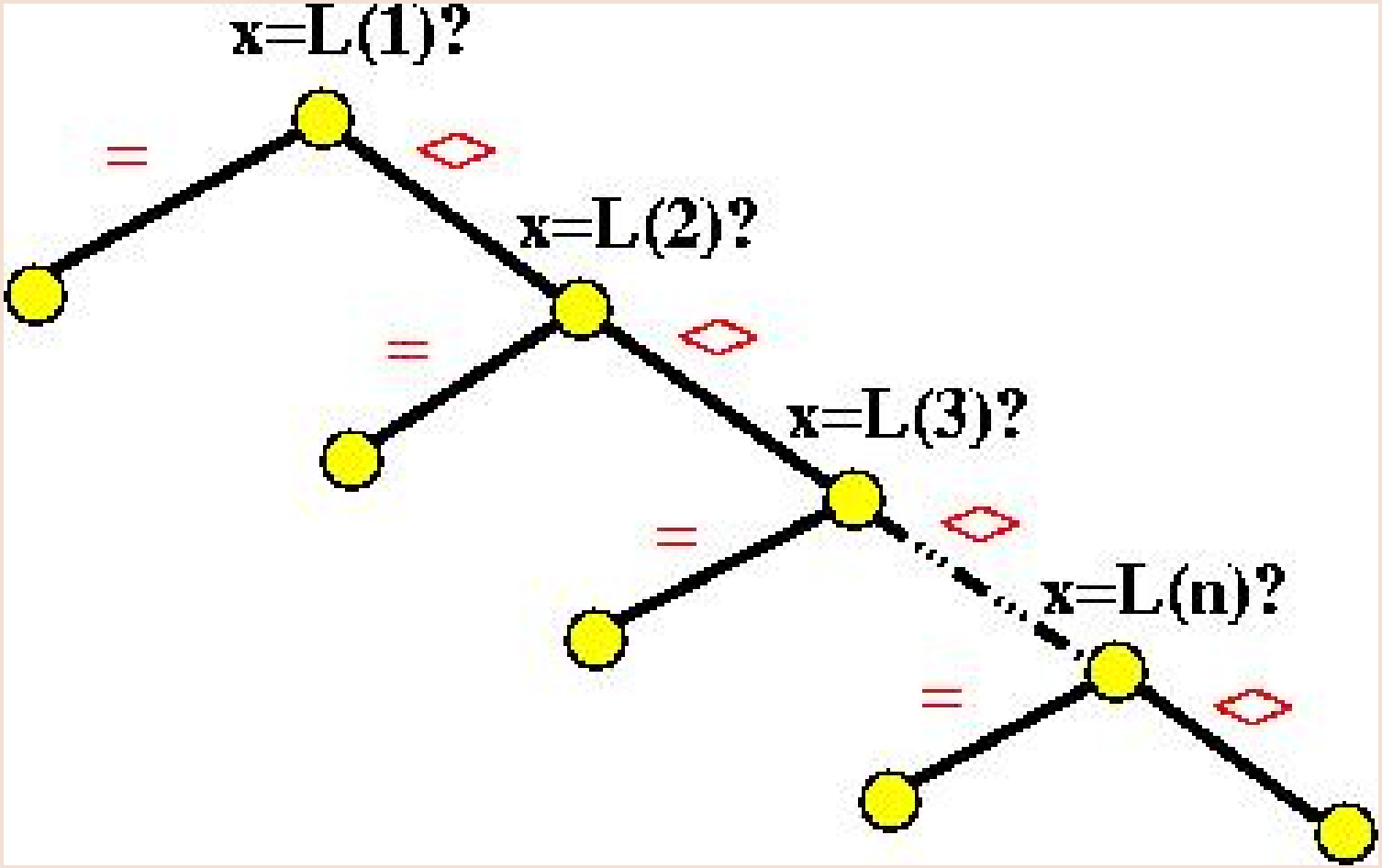
Example: Decision Tree

In an algorithm, the execution path is dictated by the evaluation of conditions (`IF` statements, conditions in `WHILE` loops, etc.). Since a condition is a boolean expression (i.e. it is either TRUE or FALSE), we can represent the condition as a node, and the result of its evaluation as two edges (TRUE/FALSE) leading to the next step in the algorithm. In the sequential search algorithm:

1. Nodes correspond to comparisons.
2. Edges correspond to the the resulting decisions.
3. Leaves correspond to (output) results, and program termination.

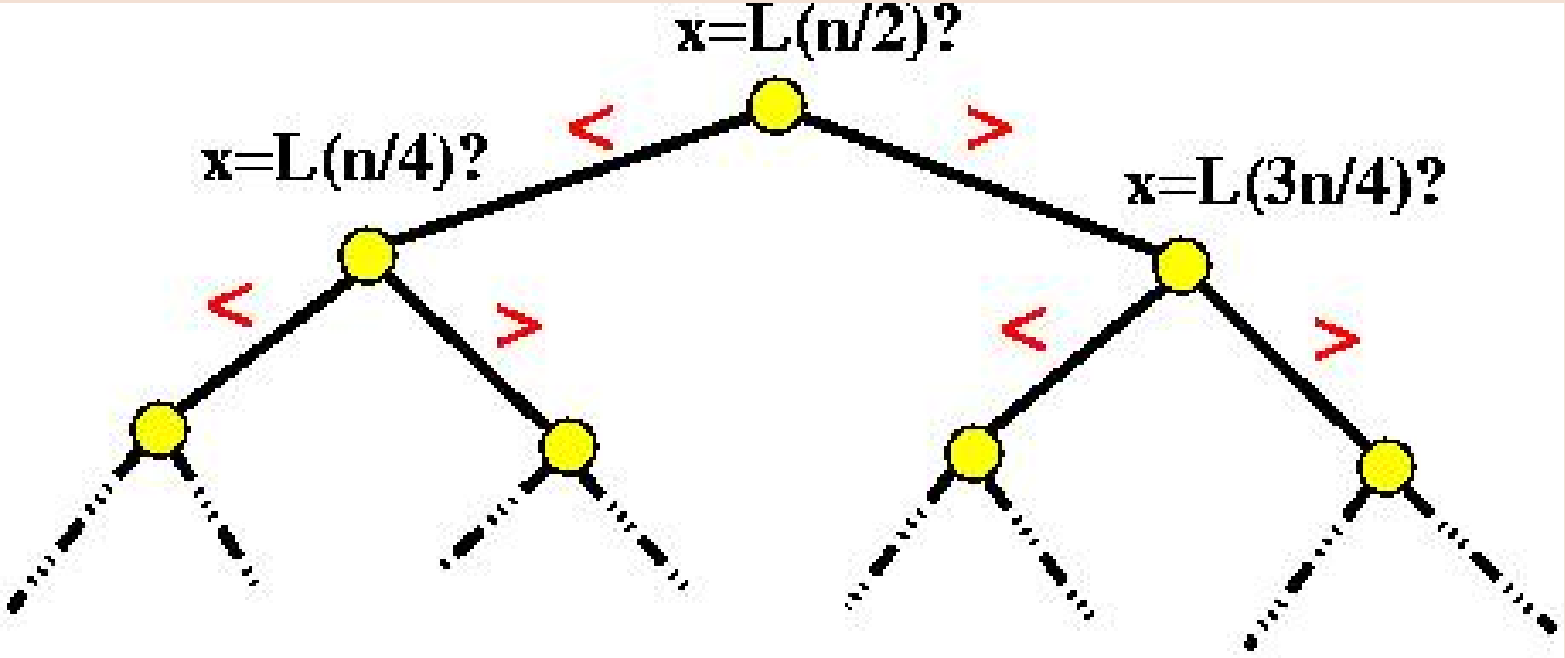
Example: Decision Tree

Linear/Sequential Search



Example: Decision Tree

Binary Search



Here all nodes correspond to comparisons, and also to (output) results, and program termination.

How many nodes are in a complete n-ary tree of depth d?

Level 0 1 node (the root)

Level 1 n nodes (since it is complete, n-ary)

Level 2 n^2 nodes

...

Level d n^d nodes

So the total is

$$1 + n + n^2 + n^3 + \cdots + n^d = \frac{1(1 - n^{d+1})}{1 - n}$$

$$= \frac{n^{d+1} - 1}{n - 1}$$

Examples are:

A complete binary tree of depth 0 This is just a single node (root)!

$$\frac{n^{d+1}-1}{n-1} = \frac{2^{0+1}-1}{2-1} = 1$$

A complete binary tree of depth 3

$$\frac{n^{d+1}-1}{n-1} = \frac{2^{3+1}-1}{2-1} = 2^4 - 1 = 15$$

Exercise: Draw this tree.

A complete ternary tree of depth 2

$$\frac{n^{d+1}-1}{n-1} = \frac{3^{2+1}-1}{3-1} = \frac{3^3-1}{2} = 13$$

Exercise: Draw this tree.

Exercise

Which has more nodes: A complete ternary tree of depth 4, or a complete 4-ary tree of depth 3?

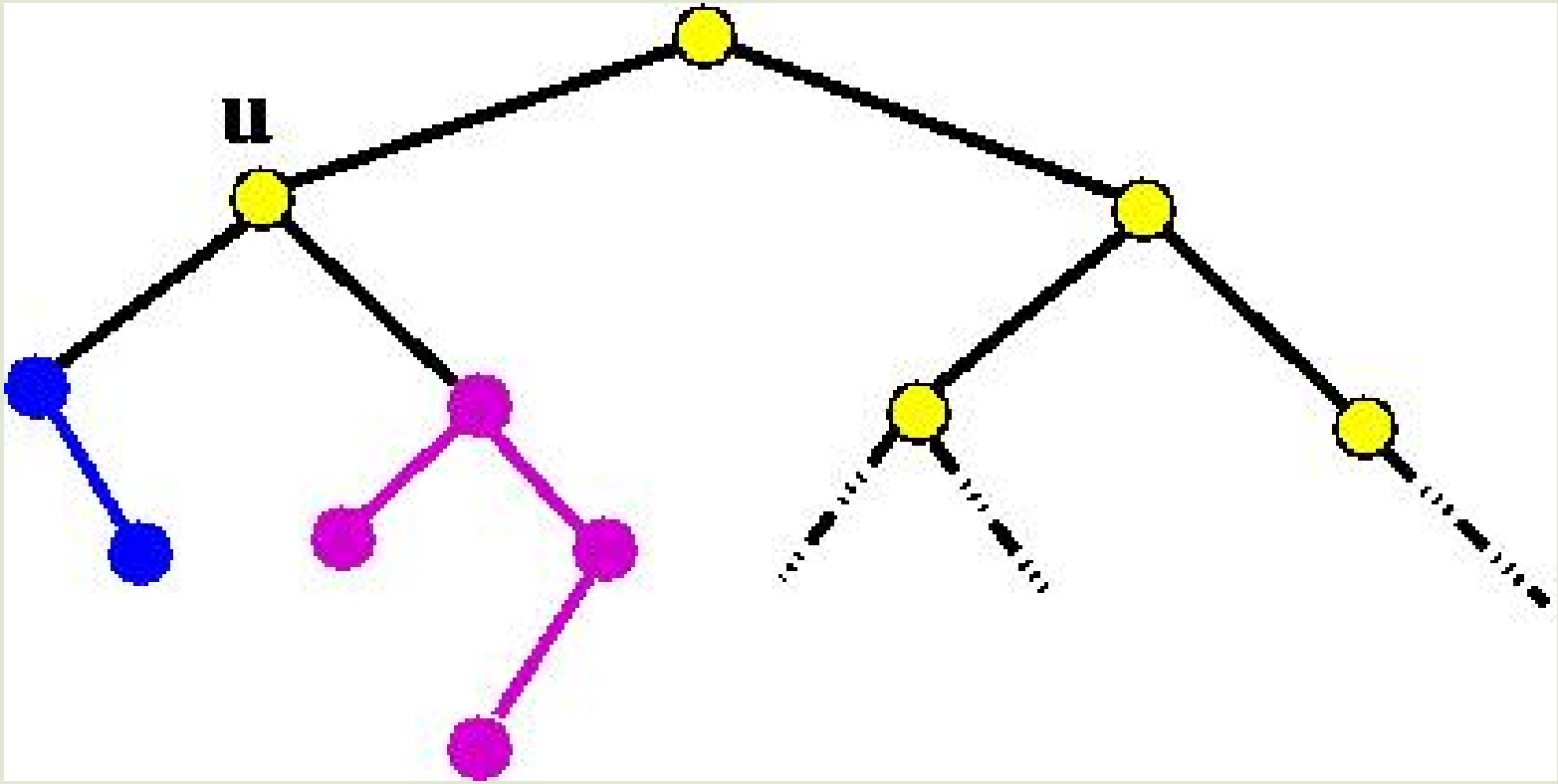
We will now consider a tree as a data structure, where at each node we can store one item of data.

Binary Search Trees

A Binary Search Trees is a tree that has the property that, *for each* node in the tree,

- the data items in the left subtree precede (alphabetically or numerically) the (data item at the) node
- the data items in the right subtree come after (alphabetically or numerically) the (data item at the) node

Left & Right Subtrees



The right sub-tree of node u is in magenta while the left sub-tree of node u is in blue

Constructing a Binary Search Tree (BST)

Suppose we have a list of data objects out of which we want to build a BST. We can procede as follows:

1. For the first object, draw the root node, and put the object there.
2. For each new object, starting at the root, compare it to the nodes on the tree:
   - If it is larger than the current node, then move to the right along the tree to the next node (if there is no “next node”, create a new edge and node to the right and put the object there)
   - If it is smaller than the current node, then move to the left along the tree to the next node (if there is no “next node”, create a new edge and node to the left and put the object there)

Examples

Construct a BST for each of the following data:

1. 5,8,2,12,10,14,9

2. 9,12,10,5,8,2,14
3. mayo / carlow / cork / tyrone / sligo / meath / louth

ALGORITHM DESIGN STRATEGIES

Brute Force Strategy

The Brute Force method is normally the most obvious method of solving a problem, but often the least efficient: It consists in listing *all* possible solutions, and then simply checking which is the best one (sometimes it is known also as *exhaustive search*. It can only be applied to problems where we can actually ennumerate all possible solutions.

Brute Force: Example 1

Suppose we have a number (n) os shops/businesses in a row on a street. We know the yearly profit/loss each makes. We want to "buy" a block of shops such as to maximize our yearly profit. Note that

1. We can ignore the "purchase price" of the shop (pretend it is being given away for free).
2. The shops we "buy" must all neighbor one another.

Given the list of n numbers, design an algorithm to decide which to purchase. Examples are:

Profits = $[6, 1, 4, 7]$

Profits = $[2, 6, 1, -4, 3]$

Profits = $[-1, -4, -7]$

Profits = $[-5, 8, 5, -3, 4, -2, 1]$