Today, we'll look at some bounds connecting numbers of vertices with heights of n-ary trees and then apply this to binary sorting.

Observations about question 3 on the HW:

These questions are most easily addressed using the triangle inequality:

$$D(u,v) \leq D(u,r) + D(v,r).$$

Also use the fact that $D(u,r) \leq H$ and $D(v,r) \leq H$. Another observation: If you have a string of inequalities

$$A \leq B \leq C \leq D \ldots \leq Z \leq A$$

what can you conclude? Answer: All are equal. You'll be able to conclude that $D(u,r) = H$ and $D(v,r) = H$. Could either u or v be a parent?

For #4: You may use the following claims:

Claim (helpful for the maximum number of components; no need to prove this). Let G be a graph with $n \geq 2$ vertices, $m \geq 1$ edges, and k components. Then there exists a graph H with the same numbers of vertices, edges, and components where every component but one is an isolated vertex.

In the max case, try to make as many of the vertices isolated as you can, using the seven edges among the remaining vertices.

Claim (helpful for the minimum; you should try to prove this). Any graph with n vertices and m edges has at least $n - m$ components.

Proof. Suppose H has n vertices and m edges. If H is a forest, then the result follows at once. Otherwise, suppose we remove r non-bridges (one at a time)....

Theorem. For integers $n \geq 1$ and $L \geq 0$, an n-ary tree has at most n^L vertices at level L.

This leads directly to

Corollary. For $n \geq 2$ and $H \geq 0$, if T is an n-ary tree of height H, then T has at most

$$|V| \leq \frac{n^{H+1} - 1}{n - 1}$$

vertices.

Proof. We consider summing the numbers of vertices at each level. Let $N_L \leq n^L$ be the number of vertices at level L. Then

$$|V| = \sum_{L=0}^{H} N_L \leq \sum_{L=0}^{H} n^L = 1 + n + n^2 + n^3 + \cdots + n^H = \frac{n^{H+1} - 1}{n - 1}.$$

Furthermore, we have equality if and only if $N_L = n^L$ for all levels L.

Let's derive a lower bound for H in terms of $|V|$ based on

$$|V| \leq \frac{n^{H+1} - 1}{n - 1}.$$

This is a sequence of algebraic manipulations to isolate H:

$$\begin{aligned}
(n-1)\,|V| &\leq n^{H+1} - 1 \\
(n-1)|V| + 1 &\leq n^{H+1} \\
\log_n\big((n-1)|V| + 1\big) &\leq \log_n(n^{H+1}) = H + 1 \\
\log_n\big((n-1)|V| + 1\big) - 1 &\leq H \\
\big\lceil \log_n\big((n-1)|V| + 1\big) \big\rceil - 1 &\leq H
\end{aligned}$$

Suppose, as an example, $H = 4$ and $\log_n\big((n-1)|V| + 1\big) = 4.5324$. Then

$$\big\lceil \log_n\big((n-1)|V| + 1\big) \big\rceil = 5$$

So the last line would read

$$5 - 1 \leq 4$$

and the line before it would read

$$4.5324 - 1 \leq 4.$$

Claim. If $a \leq b$, then $\lceil a \rceil \leq \lceil b \rceil$.

We now have a tight (this means there are examples of equality) lower bound for H

$$H \geq \big\lceil \log_n\big((n-1)|V| + 1\big) \big\rceil - 1.$$

Corollary. If $n = 2$, i.e., for binary trees,

$$H \geq \lceil \log_2(|V| + 1) \rceil - 1.$$

Corallary. If T is a full binary tree with ℓ leaves (non-parents), then there are $\ell - 1$ internal vertices (parents), for a total of $|V| = 2\ell - 1$ vertices. The H-bound formula gives us

$$\begin{aligned} H &\geq \lceil \log_2(|V| + 1) \rceil - 1 \\ &= \lceil \log_2(2\ell - 1 \ + 1) \rceil - 1 \\ &= \lceil \log_2(2\ell) - 1 \rceil \\ &= \lceil \log_2(\ell) + \log_2 2 - 1 \rceil \\ &= \lceil \log_2(\ell) \rceil \end{aligned}$$

In a full binary tree, the height is at least the log base 2 of the number of leaves.

Application. Binary sorting algorithms

Definition. A **binary algorithm** is a sequence of steps where at each step where a choice of how to proceed exists, called a **decision point**, there are two choices of how to proceed. An **exit point** is a step where the algorithm stops.

We model binary algorithms with full binary trees. The decision points will be parents and the exit points will be non-parents.

Consider the problem of designing a binary algorithm to sort a list of n distinct items, presented in an unsorted list $\{a_1, a_2, \ldots, a_n\}$. A sorting algorithm must be capable of distinguishing $n!$ different inputs. As a result, we need our full binary tree to have $n!$ leaves, each one representing an exit point, a sorted list.

For instance, if $n = 3$, we need a full binary tree with six leaves. Our input will be the list $\{a_1, a_2, a_3\}$; the blue arrows are yes, the red arrows are no.