CS180 - Sorting + Trees

Announcements
Exam 1

Average:
Sorting Algorithms

Why do we care?

- Insertion
- Selection
- Merge
- Bubble
- Quick
- Bucket
- Radix
- Shell
- van Em de Boas
Bucket Sort

n elements, each between 0 and N-1
Can we do better than O(n \log n)?
Radix Sort: for multiple-key sorting

Ex: \((1,5), (2,1), (4,2), (3,3), (5,4), (3,1), (2,2), (5,1), (2,4)\)

Sort lexicographically. (Use repeated bucket sorts)
Practicalities

Experimentally, quicksort runs faster than merge on small inputs. Why?
More practicalities

- If implemented well, the running time of insertion sort is \( O(m+n) \), where \( m = \# \text{ of inversions} \) (or out of order elements).

Conclusion:

- If the range of values is small, bucket sort (or radix sort) are faster.
Trees:

Def: A tree $T$ is a set of nodes storing elements in a parent-child relationship.

$T$ has a special node $r$, called the root.
Each node (except $r$) has a unique parent.

Compare to lists:
More dfs

- child
- siblings
- leaves
- internal nodes
- rooted subtree
- descendant / ancestor
General Tree Implementation

Pointer based:

Need a list of children in each node.
Applications

Anything where relationships are more complex than linear orderings.

Ex: - family tree
     - file systems
     - Numeric expressions
Binary Tree

- Every node has ≤ 2 children.
Nice trick

Can be pointers or array based!
Depth and Height - defined recursively

\[
\text{depth} : \quad \text{depth}(r) = 0 \\
\text{depth}(v) = \text{depth} \left( \text{parent}(v) \right) + 1
\]

\[
\text{height} : \quad \text{height}(\text{leaf}) = 0 \\
\text{height}(v) = \max \left( \text{height of children} \right) + 1
\]