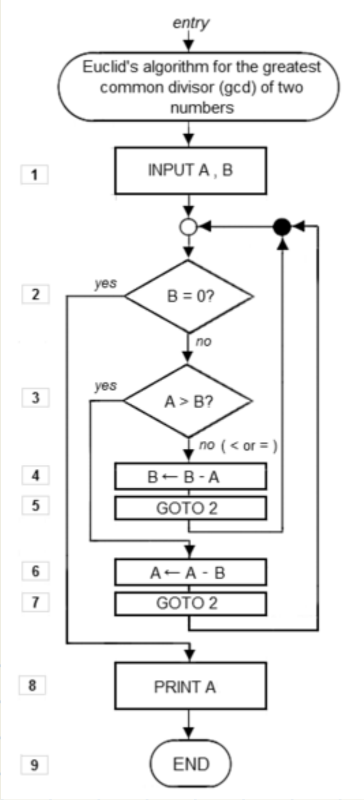
Visualizing Data Structures

Dan Petrisko

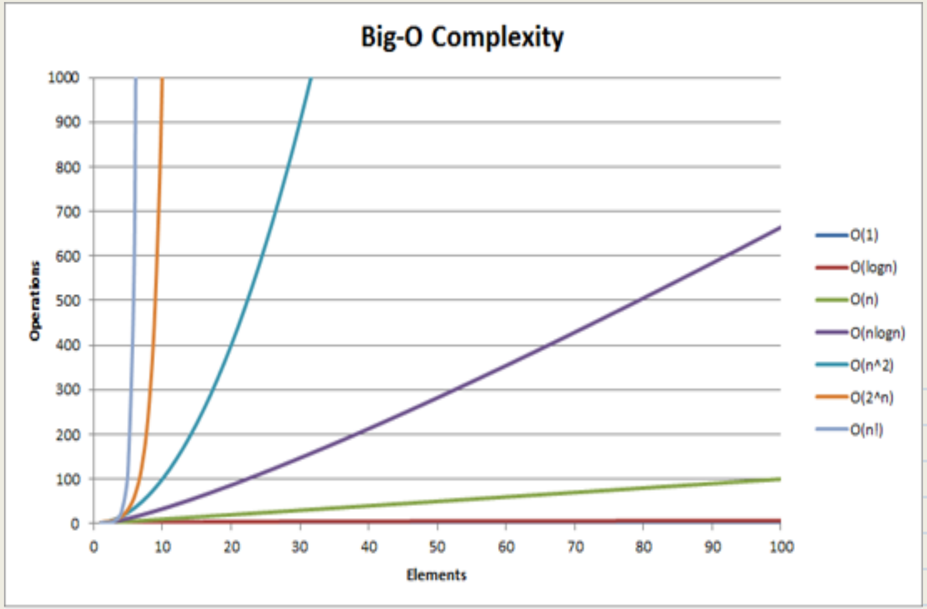
What is an Algorithm?

- A method of completing a function by proceeding from some initial state and input, proceeding through a finite number of well defined steps, and terminating at a final ending state
- Notable examples:
  - Solving a rubix cube
  - Finding the GCD of two numbers (Euclid's Algorithm)
  - Finding the shortest path between graph vertices
  - Various Searching and Sorting Algorithms



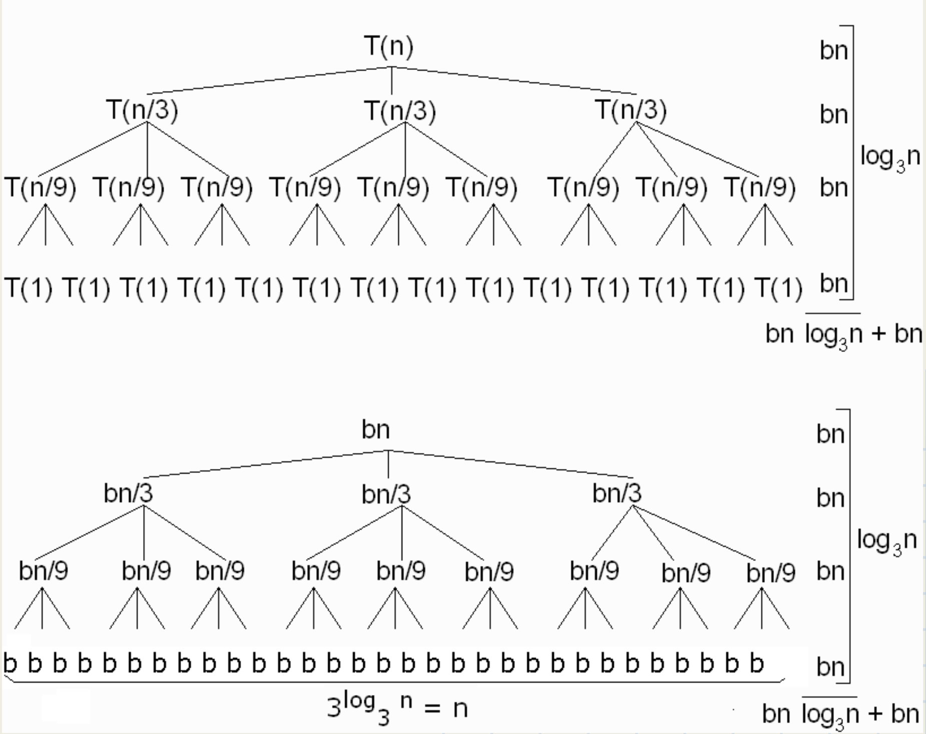
Algorithmic Analysis

- Big-O Notation describes the *limiting behavior* of an algorithm
- $f(x) = O(g(x))$ iff $f(x) < cg(x)$ for all $x > k$ where c and k are some positive values
- n is $O(n^2)$, n^2 is $O(n^3)$, $\log(n)$ is $O(n)$



Recurrence Relations

- T(n) is a function that takes the size of the data and returns the running time (in arbitrary computational units).
- Generally T(n) is given in two parts: the recursive definition and the base case
- We can *unroll* the recursive definition until we reach the base case to get the closed form



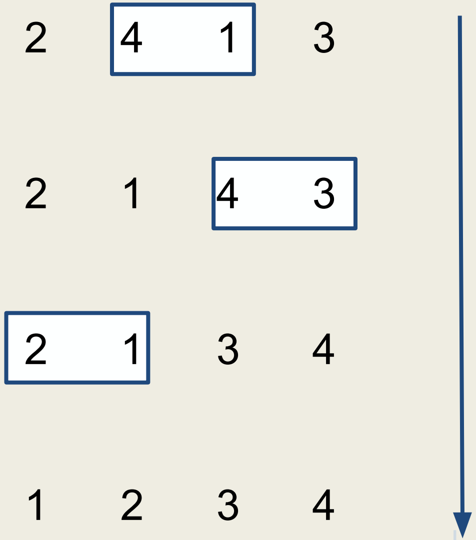
Sample Analysis - Bubble Sort

Bubble Sort works by iterating through the data set, comparing each element with the element adjacent to it

Recurrence: $T(n) = T(n-1) + n$
Base Case: $T(0) = 1$

$T(n) = T(n-1) + n = T(n-2) + n + n = T(n-3) + 3n$
$T(n) = T(0) + n*n = 1 + n^2$

We say that bubble sort is $O(n^2)$



Iterative Sorting Algorithms

- Process the set one step at a time, either:
  - Fully determining an element’s position
  - Moving closer to a fully sorted set
- Generally $O(n^2)$ performance
- Simple to program, very little memory usage

- Selection Sort
- Bubble Sort
- Insertion Sort
- Cocktail Sort

Linear Search

1. Go to each element
2. Check if the key matches the search key
3. If the end of list is reached, the list does not contain the search key

1	2	3	4	5	6	7	8	9	10
3	4	7	2	1	10	9	8	7	6

Linear Search

1. Go to each element
2. Check if the key matches the search key
3. If the end of list is reached, the list does not contain the search key

1	2	3	4	5	6	7	8	9	10
3	4	7	2	1	10	9	8	7	6

O(n)

Divide and Conquer Approach

- Attack the problem by dividing it into smaller problems
- ex: Split the list in half recursively and search each half
- This splitting indicates a logarithmic dependence on the data size: The most effective sorting algorithms have a lower efficiency bound of O(nlog(n))

- Mergesort
- Quicksort
- Binary Search
- Quickselect

Binary Search

1. Go to the middle of the list
2. Check if the key matches the search key
3. If the search key is greater than the key, repeat on right sublist
4. Else repeat of left sublist

1	2	3	4	5	6	7	8	9	10

3	4	7	2	1	10	9	8	7	6

Binary Search

1. Go to the middle of the list
2. Check if the key matches the search key
3. If the search key is greater than the key, repeat on right sublist
4. Else repeat of left sublist

1	2	3	4	5	6	7	8	9	10

3	4	7	2	1	10	9	8	7	6

O(log(n)) (doesn’t work for unsorted)

Maintaining a Sorted Structure

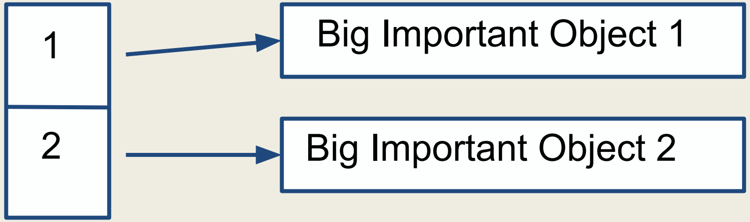
1	2	3	4	6	7	8	9	10

1	2	3	4	6	7	8	9	10	5

Structure is broken! Back to O(n), or need to re-sort

What is a Data Structure?

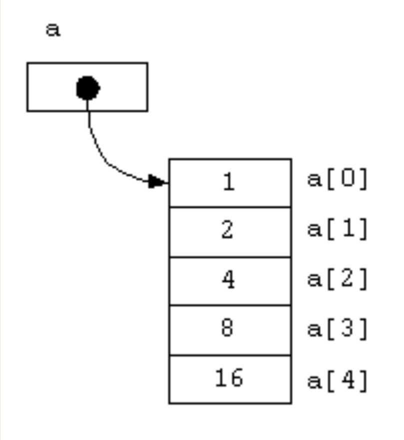
- A particular way of storing and organizing data so that it can be processed efficiently
- Most of the data structures we will examine can be related to graphs
- The data stored is easily comparable and benefits from sorting
- i.e. array of high scores in a game, not pixels in a PNG
- Usually we separate the data we want to analyze with a way to find it (key)



Arrays

- Arrays are one of the most basic structures: contiguous memory separated into values
- Analogous to an disjoint, indexed set of unconnected vertices

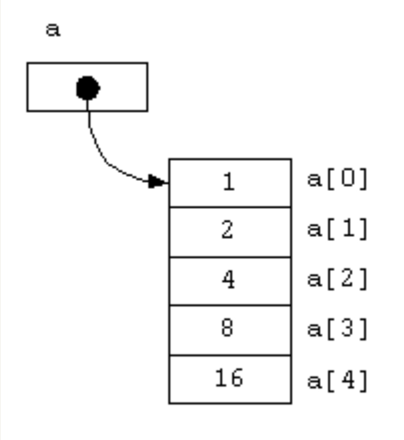
Insertion:
Insertion (maintain sort):
Growth:
Find at position n:
Find in sorted:
Find in unsorted:



Arrays

- Arrays are one of the most basic structures: contiguous memory separated into values
- Analogous to an disjoint, indexed set of unconnected vertices

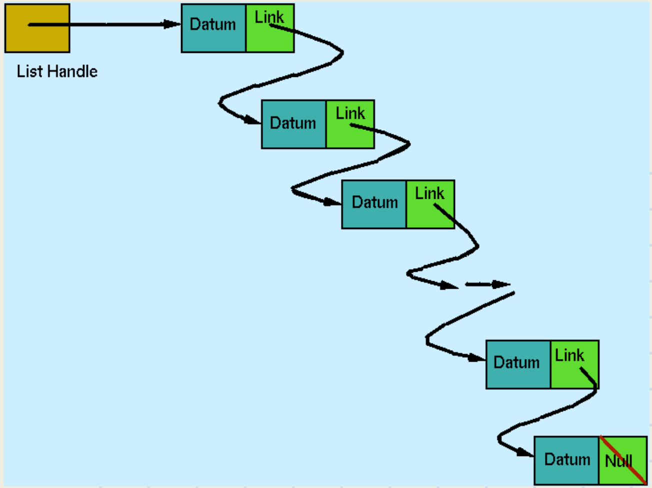
Insertion: O(1)
Insertion (maintain sort): O(n)
Growth: O(1) *amortized*
Find at position n: O(1)
Find in sorted: O(log(n))
Find in unsorted: O(n)



Linked Lists

- Linked lists are data connected by pointers to one another, forward and possibly backward
- Analogous to an unindexed, spanning graph of vertices with max in degree 1 or 2 and out degree 1 or 2 for singly or doubly linked lists

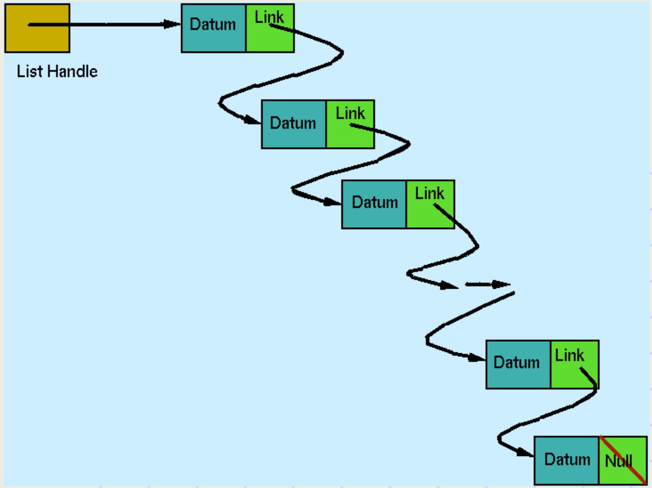
Insertion:
Insertion (maintain sort):
Growth:
Find at position n:
Find in sorted:
Find in unsorted:



Linked Lists

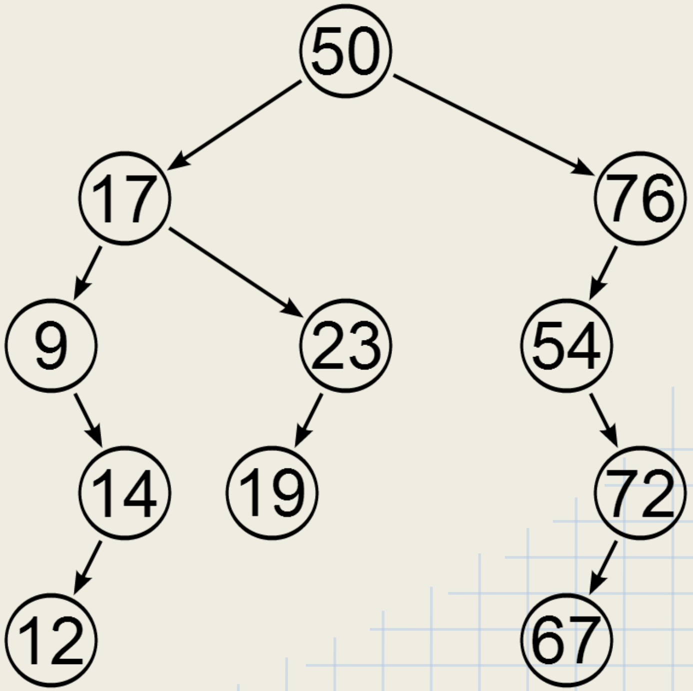
- Linked lists are data connected by pointers to one another, forward and possibly backward
- Analogous to an unindexed, spanning graph of vertices with max in degree 1 or 2 and out degree 1 or 2 for singly or doubly linked lists

Insertion: O(1)
Insertion (maintain sort): O(n)
Growth: O(1)
Find at position n: O(n)
Find in sorted: O(n)
Find in unsorted: O(n)



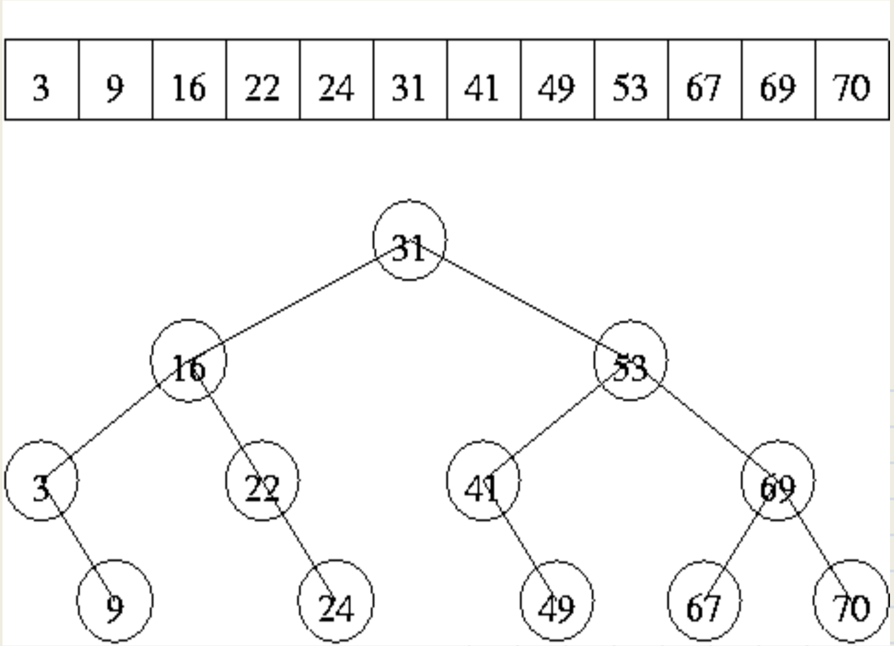
Binary Trees

- Binary Trees are graphs
- Directed, connected, rooted, ordered acyclic graphs with max in degree 1 and out degree 2



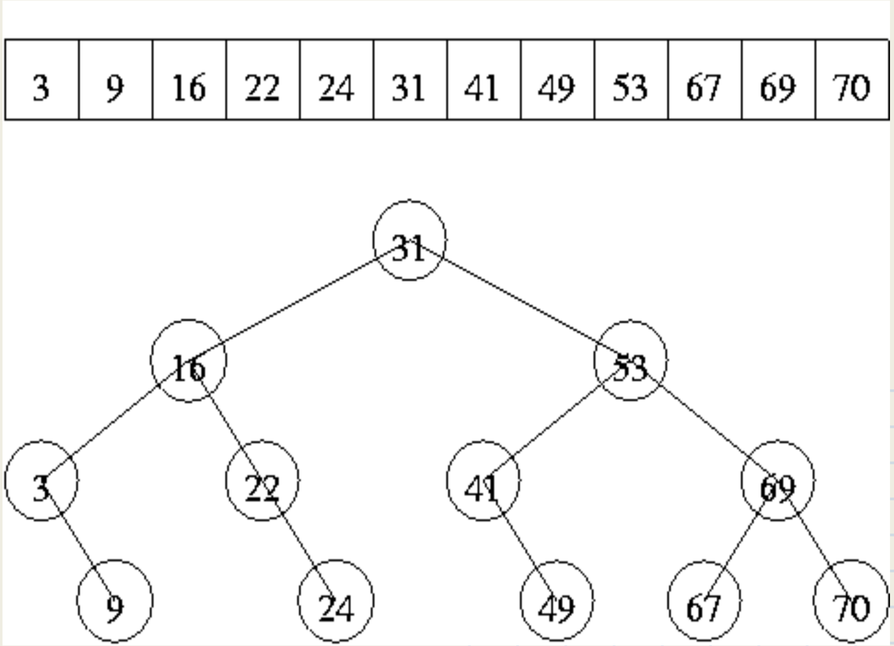
Binary Search Trees

- A balanced tree takes log(n) to maintain sortedness after insertion
- Therefore, it takes nlog(n) time to create a balanced binary tree: Where have we seen this before?



Binary Search Trees

- A balanced tree takes log(n) to maintain sortedness after insertion
- Therefore, it takes nlog(n) time to create a balanced binary tree: Where have we seen this before?
- Creating a balanced binary search tree is analogous to completely sorting an array



Hash Tables

- Non comparative method of quick search
- Only 1 memory access
- Hash function takes a key and outputs a hash value, where it is stored in an array
- Requires no sorting to find specific keys!
- But, no function is perfect: Collisions