Data Structures for Interviews

Raymond Xu
raymond@adicu.com
raymondxu.io

This Talk

Covers the most crucial and common data structures in interviews

for each:

- -Overview (what is it, what does it do)
- -Methods (what can we do with it)
- -Common Interview Themes

Assumes basic programming knowledge

- -Java syntax

Outline

Big O

Data Structures

Other Interview Topics

Outline

Big O

Data Structures

Other Interview Topics

Big O

Big O describes asymptotic runtime as a function of input size

Represents an upper bound

Smaller is better

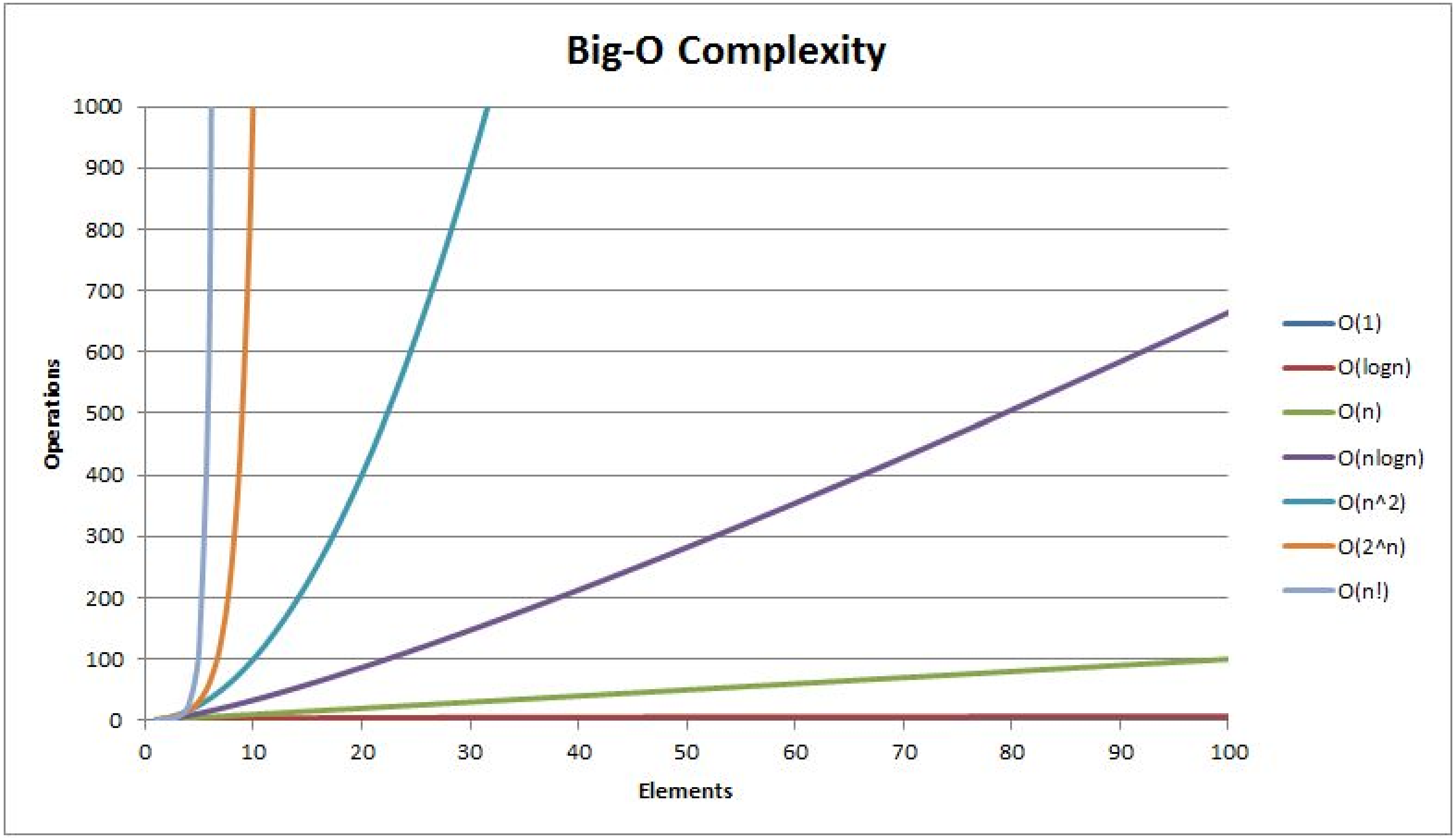
Big O

$O(1)$, $O(logn)$, $O(n)$, $O(nlogn)$, $O(n^2)$, $O(2^n)$, $O(n!)$

Drop constants and smaller components

Big O is applied to both time and space complexity

Big O



Big-O Complexity
1000
900
800
700
600
500
400
300
200
100
0
Operations
0
10
20
30
40
50
60
70
80
90
100
Elements
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

Big O

```
int sum(int[] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        sum += arr[i];
    }
    return sum;
}
```

Big O

```
int sum(int[] arr) {
    int sum = 0; // O(1)
    for (int i = 0; i < arr.length; i++) { // n times
        sum += arr[i]; // O(1)
    }
    return sum; // O(1)
}
// O(1) + n*(O(1)) + O(1) = O(n)
```

Big O

Know the runtime of all methods of all common data structures and algorithms

Be able to analyze the time and space complexity of functions

Big O informs the advantages and disadvantages of different data structures

Big O

Given a sorted array that has been rotated, find the minimum element.

Big O

What's faster than O(n)?

Big O

What's faster than O(n)?

Is O(1) intuitively possible?

Big O

What's faster than O(n)?

Is O(1) intuitively possible?

What does O(logn) entail?

Big O

Binary search!

Outline

Big O

Data Structures

Other Interview Topics

Arrays and Strings

Arrays are linear, sequential blocks of memory

Strings are arrays of characters

Access elements by index in O(1)

Arrays and Strings

```
int[] arr = {1, 3, 5, 2, 6, 9};
System.out.println(arr.length); // 6
System.out.println(arr[3]); // 2

String str = “hello”;
System.out.println(str.length()); // 5
System.out.println(str.substring(1,3)); // “el”
System.out.println(str.charAt(0)); // ‘h’
```

Arrays and Strings

How do you recursively reverse a string?

Arrays and Strings

```
String reverse(String str) {
    if (str == null || str.length() <= 1) {
        return str;
    }
    return reverse(str.substring(1)) + str.charAt(0);
}
```

Common Interview Themes

Arrays
- sums, searches, sorts

Strings
- reversal, palindromes, anagrams

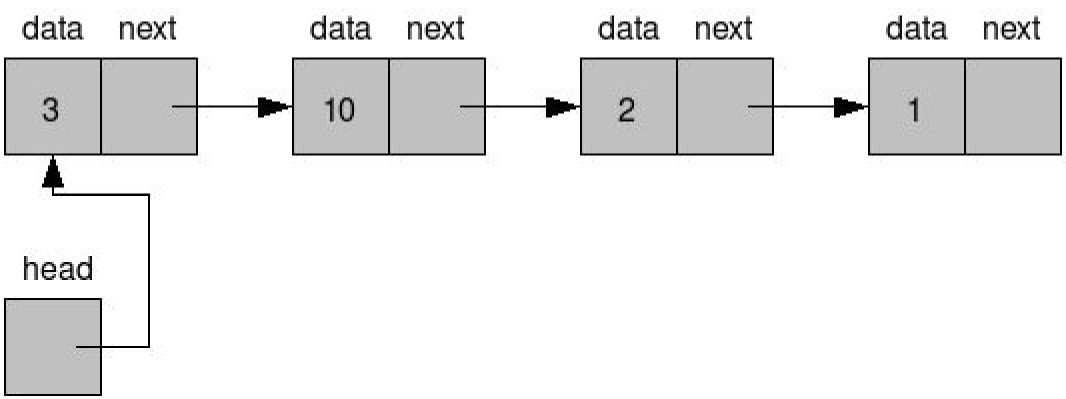
Linked Lists

Linked Lists are sequences of nodes

A node contains a value as well as a pointer to one other node

Interviews commonly focus on singly-linked lists but there are other types as well

Linked Lists



Linked Lists

Questions specifically about linked lists tend to deal with node manipulation

Define your own Node class (not Java's LinkedList)

Linked Lists

```
public class Node {
   int value;
   Node next;
}

public class LinkedList {
   Node head;
}
```

Linked Lists

How do you find the middle node of a linked list?

Linked Lists

```
Node getMiddleNode(Node head) {
    Node slow = head;
    Node fast = head;
    while (fast != null && fast.next != null) {
        fast = fast.next.next;
        slow = slow.next;
    }
    return slow;
}
```

Common Interview Themes

Implementing a method

e.g. insert, remove, reverse, etc.

Accessing a specific node’s data

e.g. middle, kth from end, cycle start

Merging/Sorting

e.g. merge 2 sorted linked lists

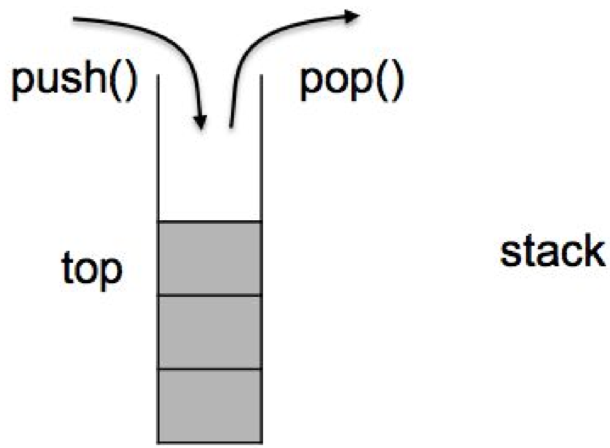
Stacks and Queues

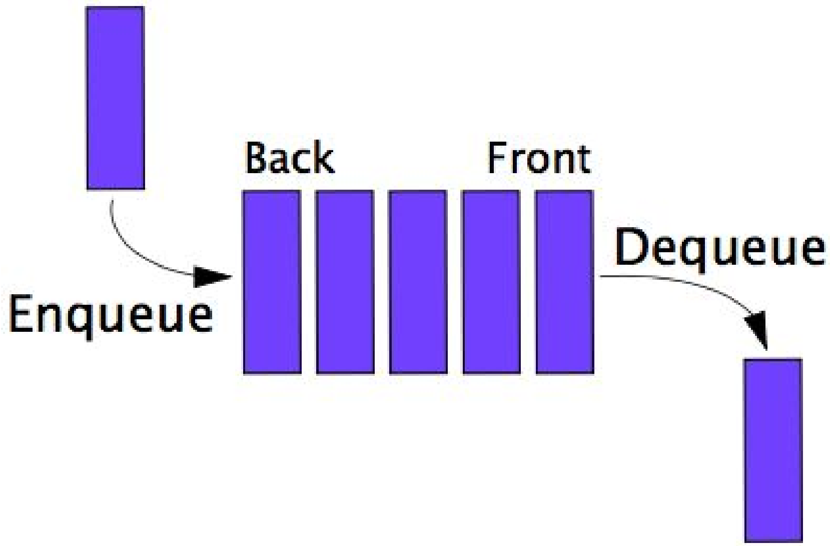
Stacks and Queues maintain a linear ordering of elements based on insertion order

Stacks: LIFO (Last In First Out)

Queues: FIFO (First In First Out)

Stacks and Queues





Stacks and Queues

```
Stack<Integer> s = new Stack<Integer>();
s.push(1);
s.push(5);
System.out.println(s.peek()); // 5
System.out.println(s.pop()); // 5
System.out.println(s.pop()); // 1
System.out.println(s.empty()); // true

// “Queue” in Java is an interface
Queue<Integer> q = new ArrayDeque<Integer>();
q.addLast(2);
q.addLast(3);
System.out.println(q.removeFirst()); // 2
System.out.println(q.removeFirst()); // 3
```

Stacks and Queues

Write a function to determine if a string consisting of the characters '{', '}', '[', and ']' is balanced.

For example, "{[]}" is balanced, and "{[}]" is not.

Stacks and Queues

```
boolean isBalanced(String str) {
    Stack<Character> stack = new Stack<Character>();
    for (int i = 0; i < str.length(); i++) {
        switch (str.charAt(i)) {
            case '{': stack.push('{');
                break;
            case '[': stack.push('[');
                break;
            case '}': if (stack.pop() != '{') { return false; }
                break;
            case ']': if (stack.pop() != '[') { return false; }
                break;
        }
    }
    return stack.isEmpty();
}
```

Common Interview Themes

Implementation

-stack, queue, queue using 2 stacks

Utility data structure

HashMaps and HashSets

HashMaps map keys to values

-also known as Hashtables or Dictionaries

HashSets store a set of elements

O(1) insertion, deletion, and lookup!

There are other types of Maps and Sets too (check your language)

HashMaps and HashSets

```
Map<Integer, String> map = new HashMap<Integer, String>();

map.put(3, “triangle”);
map.put(4, “square”);
System.out.println(map.get(3)); // “triangle”
System.out.println(map.containsKey(4)); // true
System.out.println(map.containsValue(3)); // false

for (Integer i: map.keySet()) {
    System.out.println(i + “ : “ + map.get(i));
}

Set<String> set = new HashSet<String>();
set.add(“paypal”);
set.add(“venmo”);
System.out.println(set.contains(“paypal”)); // true
System.out.println(set.contains(“braintree”)); // false
```

Common Interview Themes

Almost always a utility data structure

Counting/Frequency/Histogram

Constructing mappings

Tracking seen elements

HashMaps and HashSets

Return the most frequently occurring character in a string.

HashMaps and HashSets

```
Character findMostFrequentCharacter(String str) {
    Map<Character, Integer> map = new HashMap<Character, Integer>();
    for (int i = 0; i < str.length(); i++) {
        Character c = str.charAt(i);
        if (map.containsKey(c)) {
            map.put(c, map.get(c) + 1);
        }
        else {
            map.put(c, 1);
        }
    }

    int max = 0;
    Character maxChar = null;
    for (Character c: map.keySet()) {
        if (map.get(c) > max) {
            max = map.get(c);
            maxChar = c;
        }
    }
    return maxChar;
}
```

Trees

Trees store data in a hierarchical manner

A node has a value as well as multiple pointers to other nodes

A tree stores a pointer to the root node

Many different types
- Binary: each node has up to 2 children

Trees

Terminology

Root – the top node in a tree
Parent – the converse notion of child
Siblings – nodes with the same parent
Descendant – a node reachable by repeatedly proceeding from parent to child
Ancestor – a node reachable by repeatedly proceeding from child to parent
Leaf – a node with no children
Edge – a connection between one node to another
Path – a sequence of nodes and edges connecting a node with a descendant
Depth – the number of edges from the node to the root
Height – the largest number of edges from the node to a leaf

Trees

Terminology

A binary tree is balanced if and only if:

1. The left and right subtrees' heights differ by at most one
2. The left and right subtrees are balanced

Binary Search Trees

All nodes in the left subtree of a root node have values that are smaller than the root's

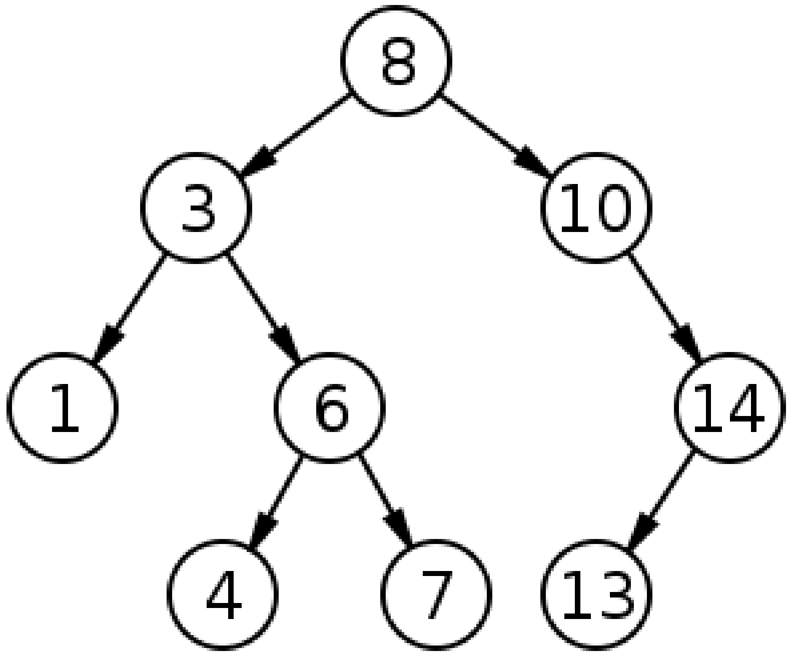
All nodes in the right subtree of a root node have values that are larger than the root's

Like Linked Lists, these questions typically involve node manipulation

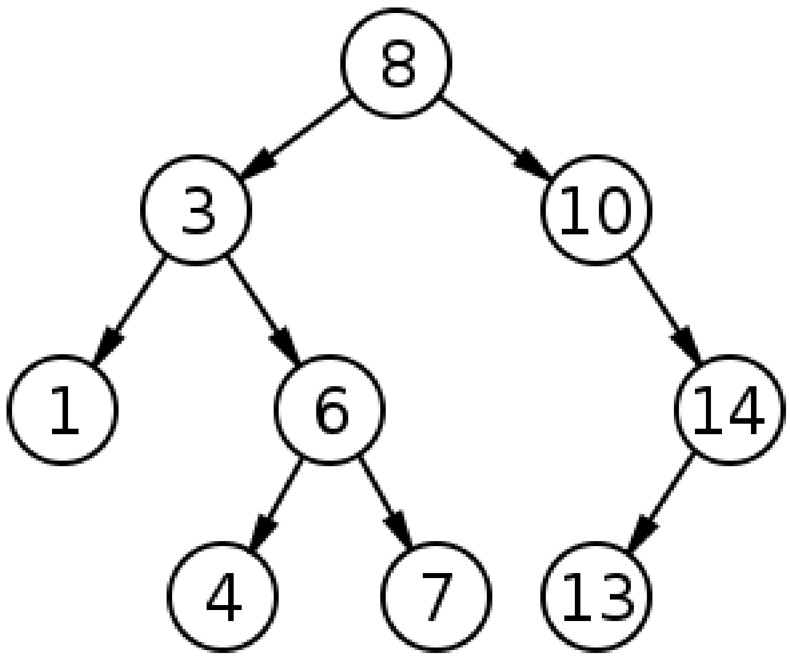
Binary Search Trees

```
public class Node {
   int value;
   Node left;
   Node right;
}

public class BinarySearchTree {
   Node root;
}
```

Binary Search Trees



Binary Search Trees



log(n) access, insertion, and removal
(if balanced)

Binary Search Trees

Write the insert function for a binary search tree.

Binary Search Trees

```
public void insert(int key) {
    if (root == null) root = new Node(key);
    else insert(root, key);
}

private Node insert(Node curr, int key) {
    if (curr == null) {
        return new Node(key);
    }
    if (key < curr.value) {
        curr.left = insert(curr.left, key);
    }
    else if (key > curr.value) {
        curr.right = insert(curr.right, key);
    }
    else {
        return null;
    }
    return curr;
}
```

Heaps

Also known as Priority Queues

Heaps provide fast access to the smallest or largest value.

Min-heap: log(n) access to smallest value
Max-heap: log(n) access to largest value

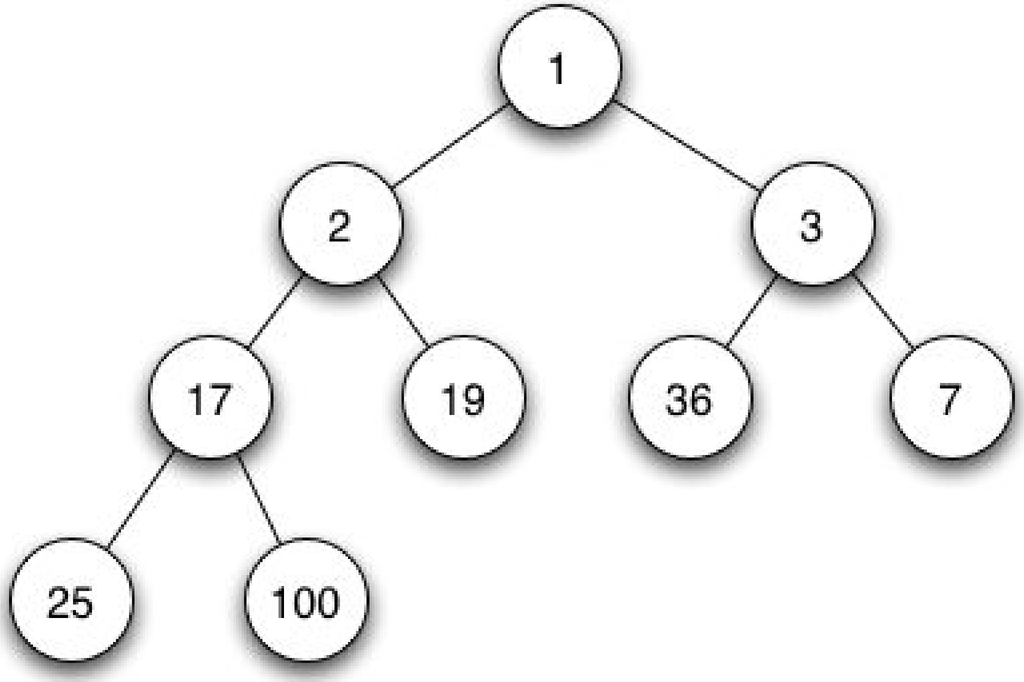
Heaps are technically arrays, but it's good to think of them as complete binary trees

Heaps

In a min-heap, the value at any node is smaller than both of its children's values

In a max-heap, the value at any node is larger than both of its children's values

Heaps



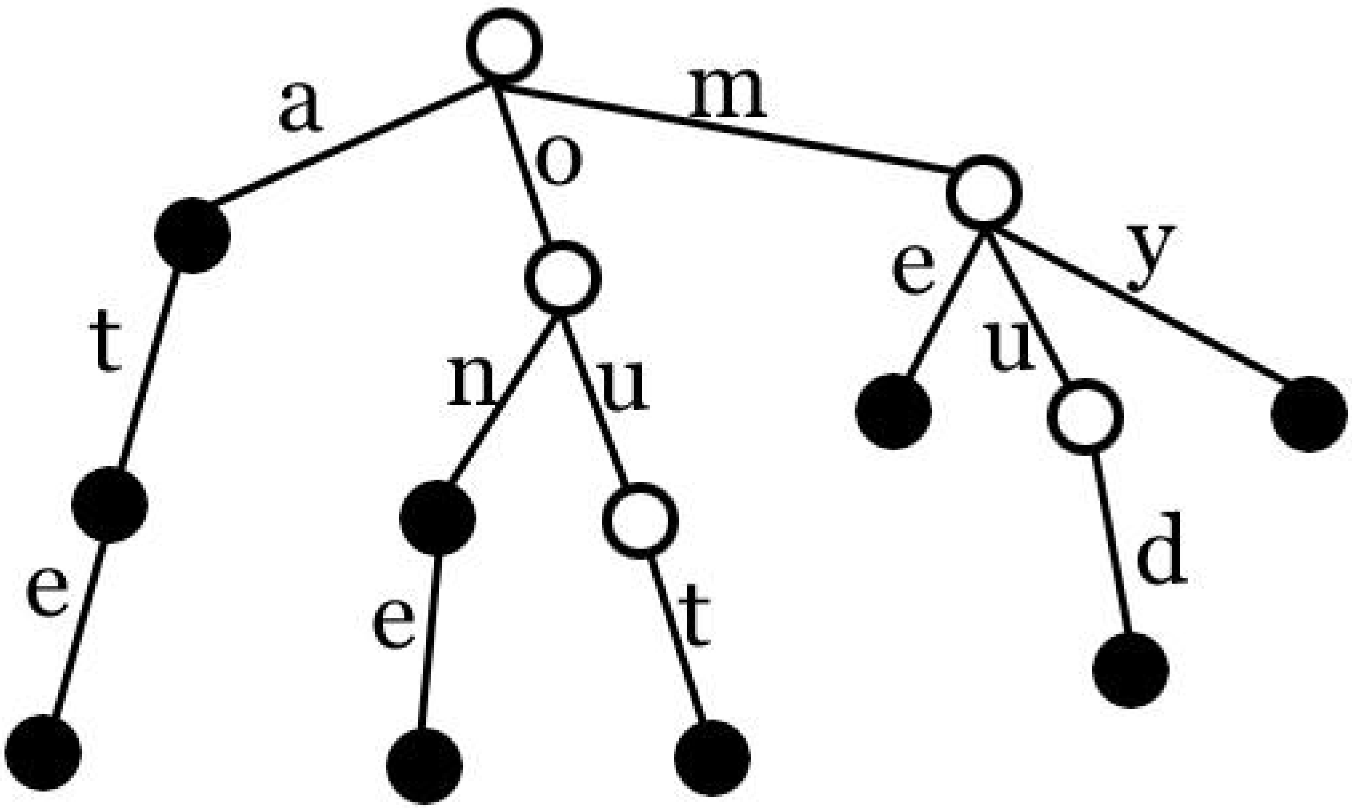
Tries

Also known as Prefix Trees or Radix Trees

Tries store a set of strings

Each node stores a character, pointers to other nodes, and a variable that indicates whether the end of a word has been reached

Tries



Common Interview Themes

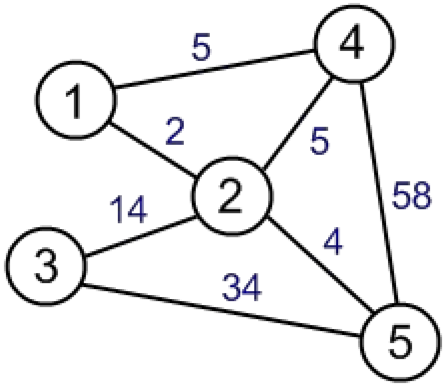
BST Methods
    -insert, isValid, isBalanced, isSymmetric
Relationships
    -print a path between 2 nodes, find LCA
Traversals
    -pre-order, in-order, post-order, level-order

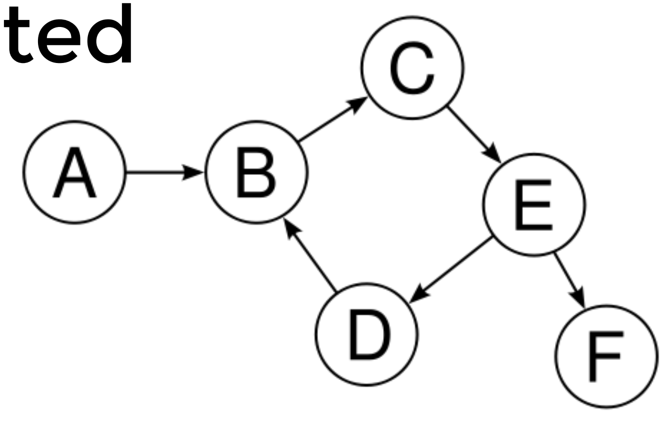
Heaps and Tries are usually utility data structures

Graphs

A graph is a set of nodes and a set of edges

Many types of graphs
- -Directed or Undirected
- -Weighted or Unweighted
- -Connected or Unconnected





Graphs

Representations:

- -Adjacency list
- -Adjacency matrix

2D arrays are graphs too!

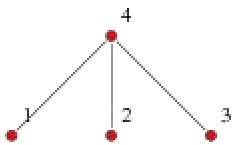
Graphs

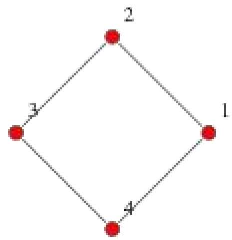
Adjacency List

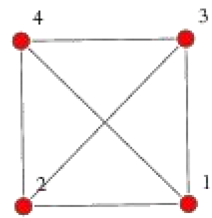
```
public class Node {
    public int value;
    public ArrayList<Edges> edges;
}

public class Edge {
    public Node destination;
    public int weight;
}

public class Graph {
    public ArrayList<Node> nodes;
}
```

Graphs

Adjacency Matrix







$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Graphs

2 key algorithms:

- -Breadth-first Search (BFS)
- -Depth-first Search (DFS)

Good-to-know algorithms:

- -Djiskstra's
- -Kruskal/Prim
- -Topological Sort

Graphs

Breadth-first Search (BFS)

```
boolean BFS(Node root, Node dest) {
    Queue<Node> q = new ArrayDeque<Node>();
    q.addLast(root);
    while (!q.isEmpty()) {
        Node curr = q.removeFirst();
        if (curr == dest) return true;
        curr.visited = true;
        for (Node n: curr.neighbors) {
            if (!n.visited) {
                q.addLast(n);
            }
        }
    }
    return false;
}
```

Graphs

Depth-first Search (DFS)

```
boolean DFS(Node curr, Node dest) {
    if (curr == dest) {
        return true;
    }
    curr.visited = true;
    for (Node n: curr.neighbors) {
        if (!n.visited) {
            if (DFS(n, dest)) {
                return true;
            }
        }
    }
    return false;
}
```

Graphs

Given a boolean 2D matrix, find the number of islands.

```
{1, 1, 0, 0, 0},
{0, 1, 0, 0, 1},
{1, 0, 0, 1, 1},
{0, 0, 0, 0, 0},
{1, 0, 1, 0, 1}
```

Graphs

Solution: Apply a search

Common Interview Themes

Typically the word “graph” won’t appear in the problem statement (disguised questions)

Translate the problem to a graph problem (connectivity, cycles, partitions, etc)

Apply a search (usually)

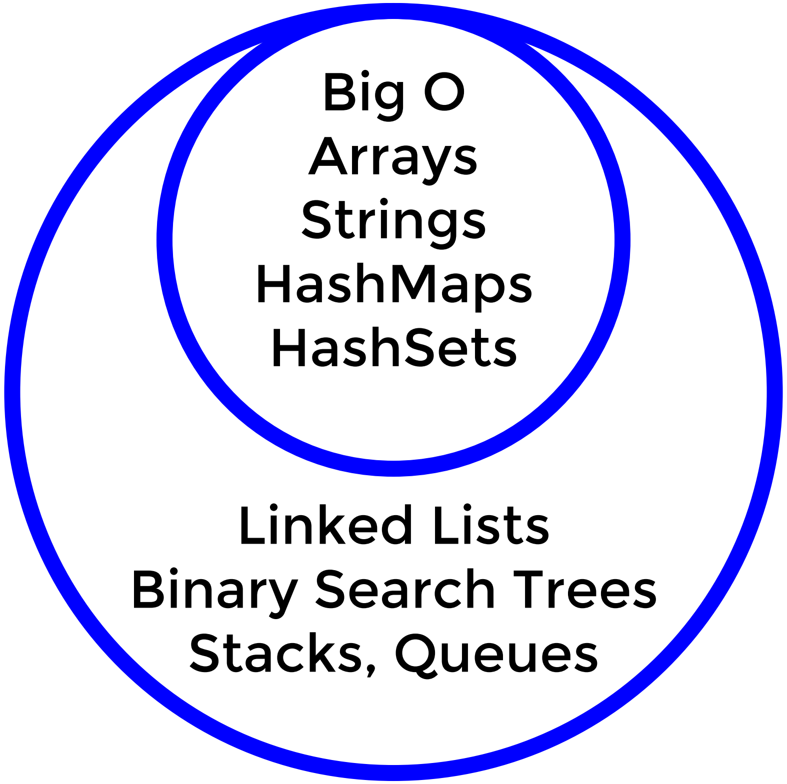
Rings of Knowledge

This is a lot of information! What order should I study them in?

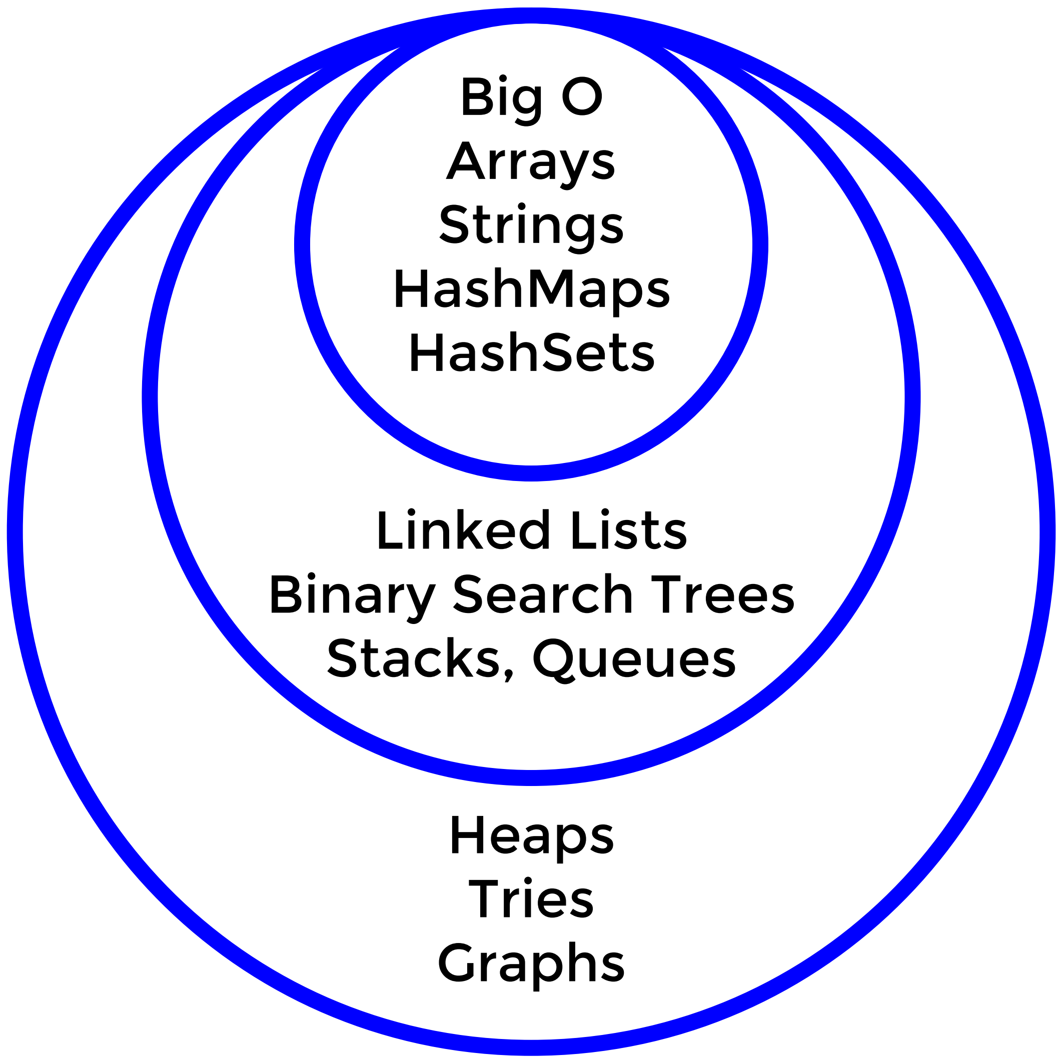
Ring 1 (Very common)

Big O
Arrays
Strings
HashMaps
HashSets

Ring 2 (Common)



Ring 3 (Uncommon)



Outline

Big O

Data Structures

Other Interview Topics

Other Topics

Data structures are the core of technical interviews, but they aren't <u>everything</u> you need to know!

Other Topics

Algorithms

- -Sorting
- -Divide and Conquer
- -Greedy
- -Dynamic Programming

Design/OOP

Language Knowledge

Discrete Math

Bits

Systems

Resources

Learning Data Structures:

- -3134/3137 + textbook
- -Wikipedia
- -Cracking the Coding Interview (CTCI)

Practicing Questions:

- -Leetcode
- -GeeksForGeeks
- -HackerRank
- -CTCI

Practice!

Online

Friends

Whiteboard

Cookies and Code

Thanks for Coming!

Data Structures for Interviews

Raymond Xu
raymond@adicu.com
raymondxu.io