CS 600.226: Data Structures

Michael Schatz

Oct 10 2018

Lecture 18. Midterm review 2

Midterm Topics

Topics

01.Intro (kd-tree)
02.Interfaces
03.ArraysGenericsExceptions
04.Lists
05.Iterators
06.Complexity
07.MoreComplexity
08.Sorting
09.Stacks
10.StacksJunit
11.Queues and Dequeues
12.Lists (Single/Double)
13.MoreLists
14.Trees & Tree Iteration
15.Graphs
16.GraphSearch

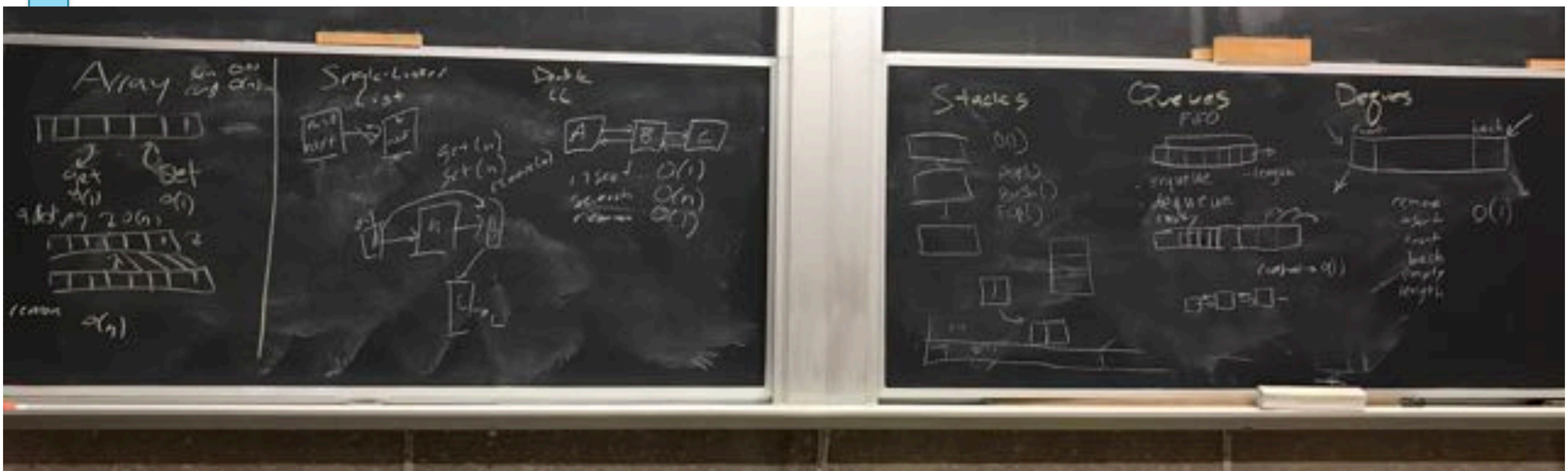
For each data structure discuss:

- Explain the interface
- Explain/Draw how it will be implemented
- Explain/Draw how to add/remove elements
- Iterate through the elements
- Explain the complexity of these

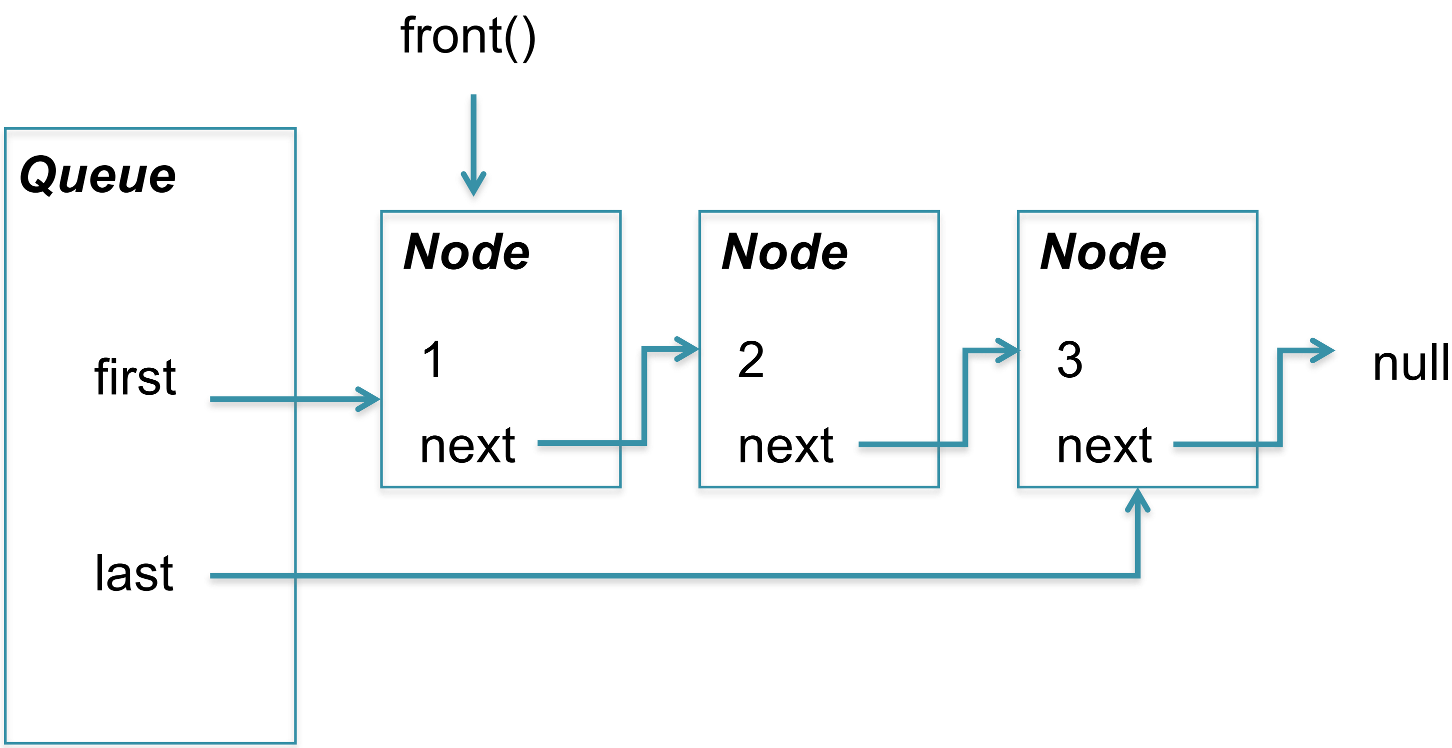
In addition:

- Can you discuss interfaces and ADTs
- Can you discuss computational complexity

Midterm Topics

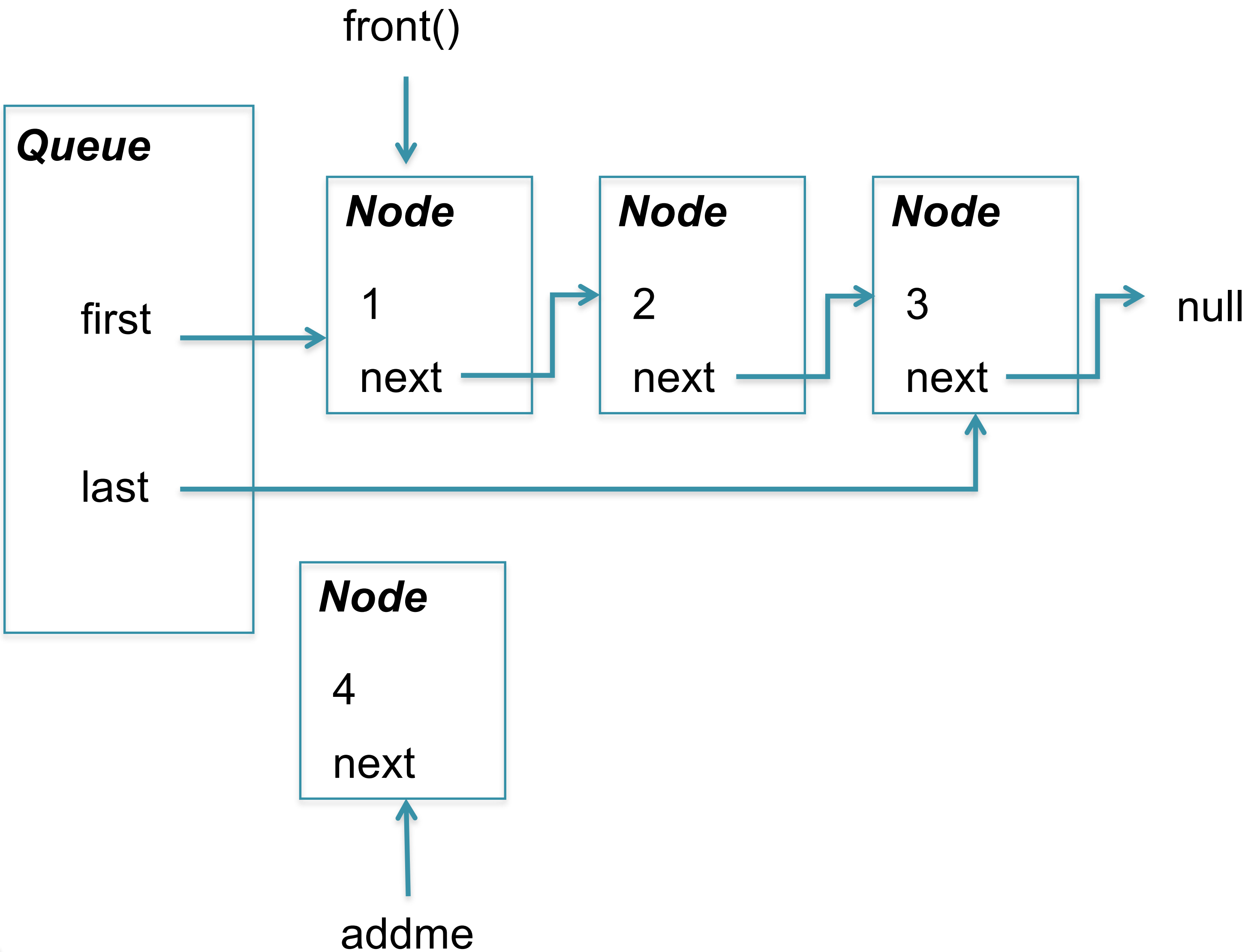


Enqueue last, Dequeue first

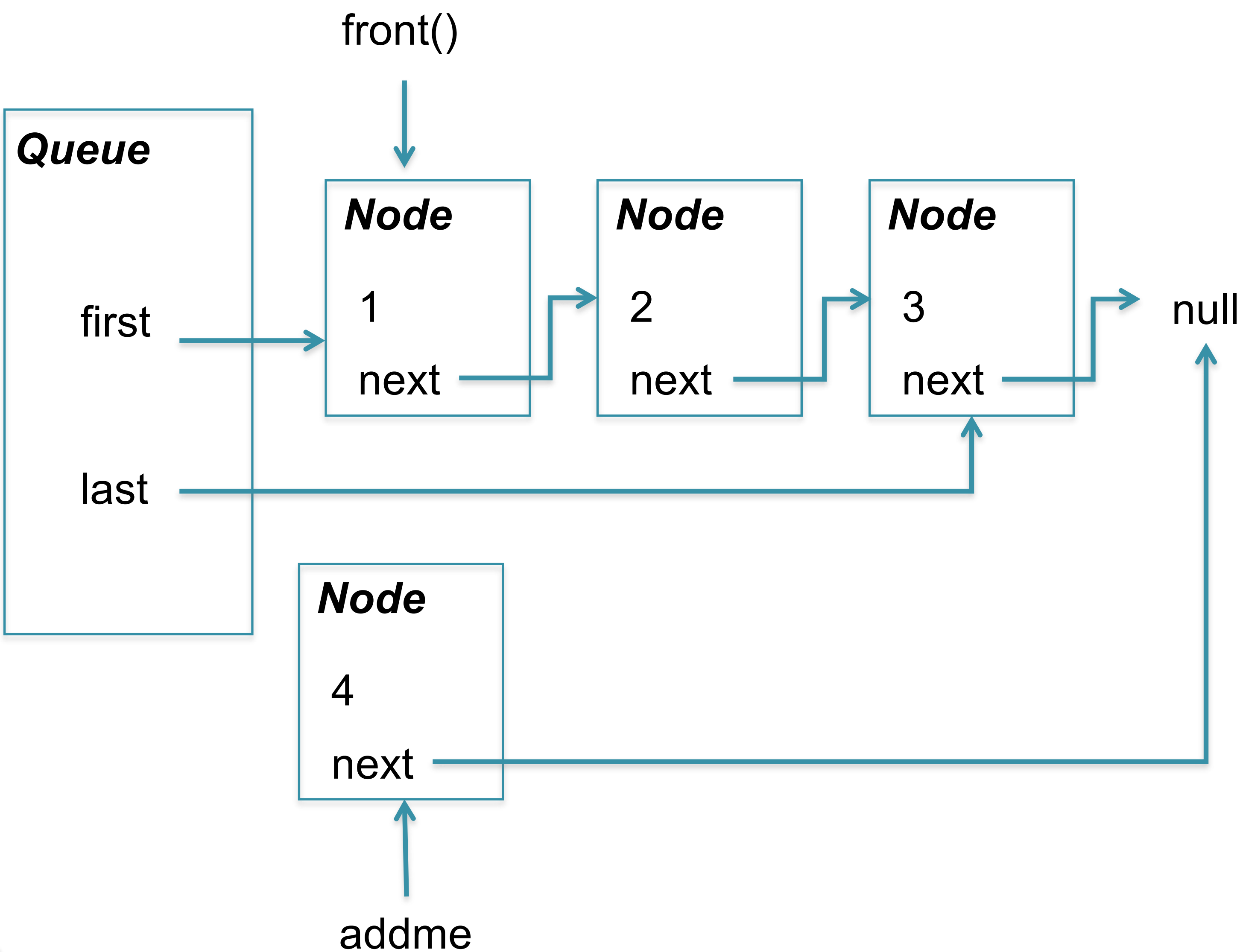


Lets try inserting at last and removing from first

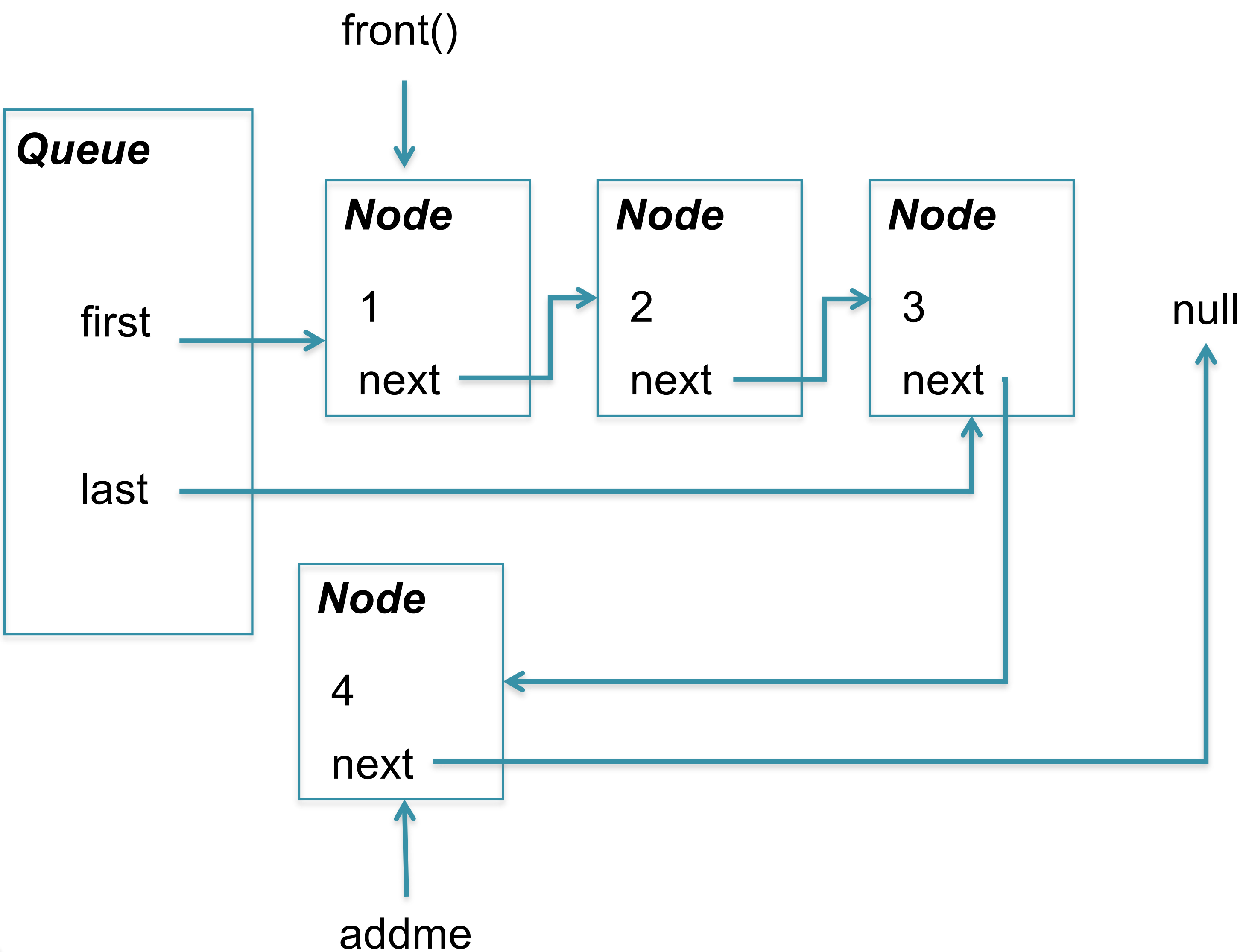
Enqueue last, Dequeue first



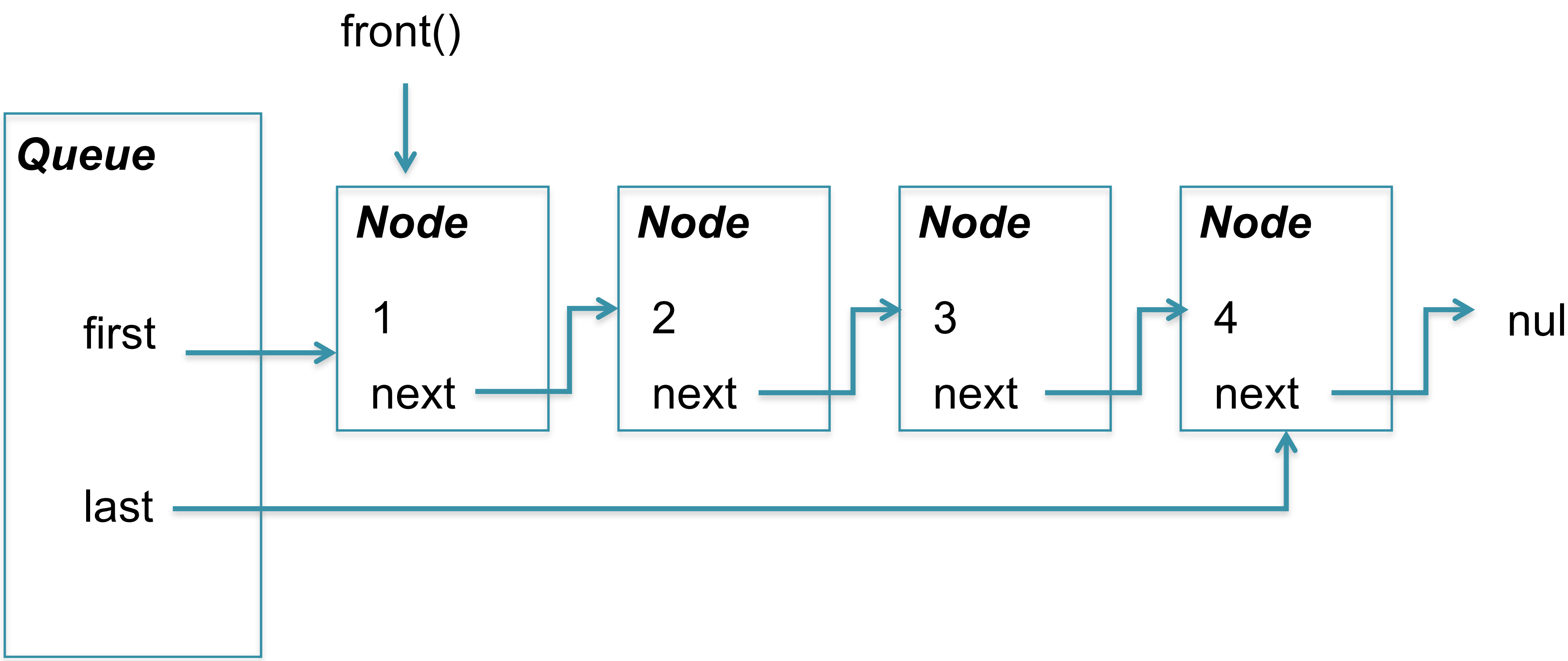
Enqueue last, Dequeue first



Enqueue last, Dequeue first

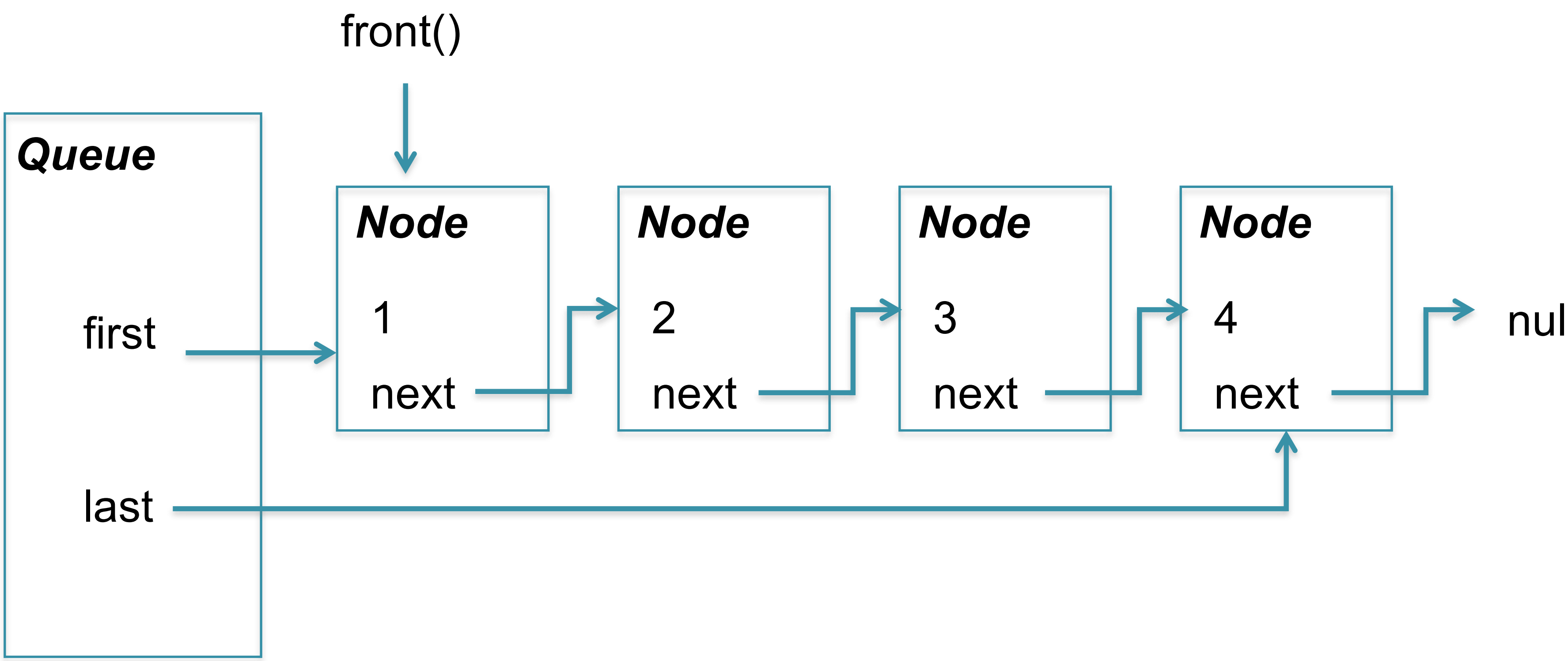


Enqueue last, Dequeue first



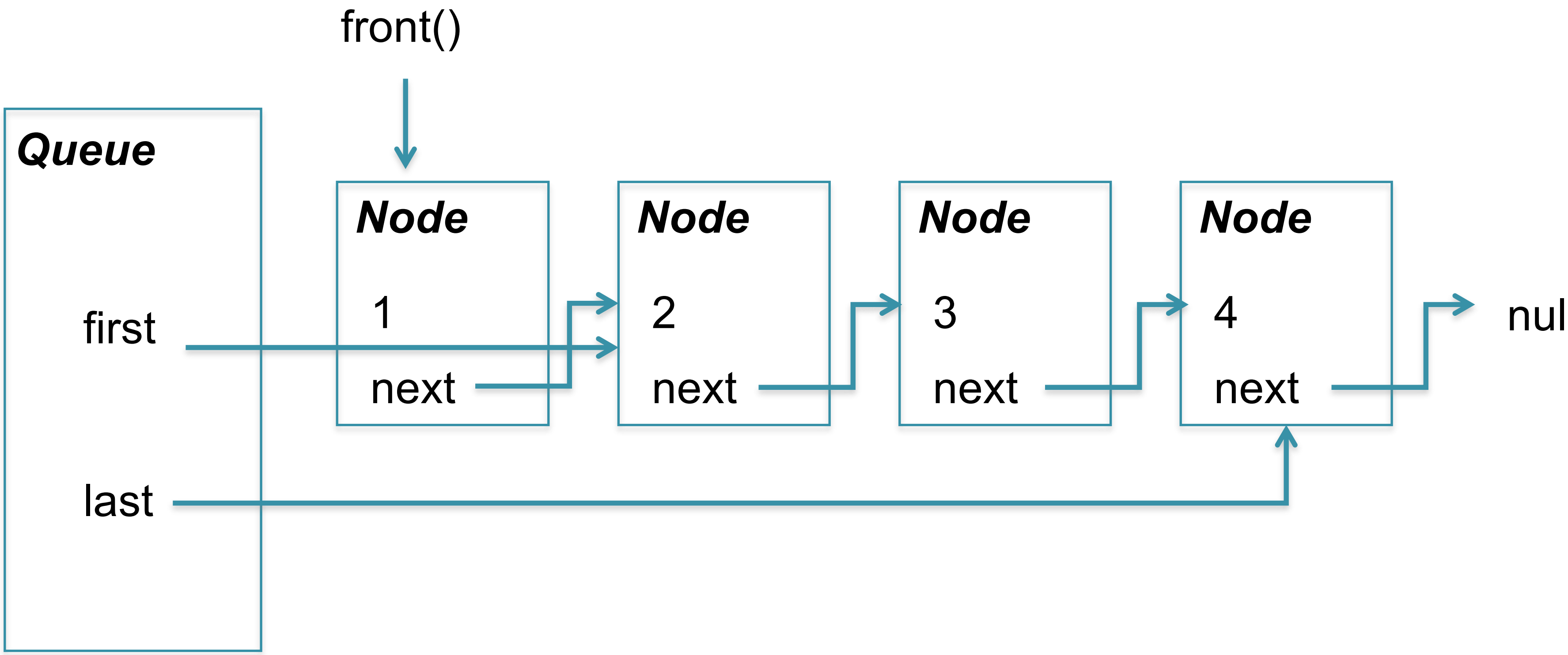
Enqueue is an O(1) operation ☺

Enqueue last, Dequeue first

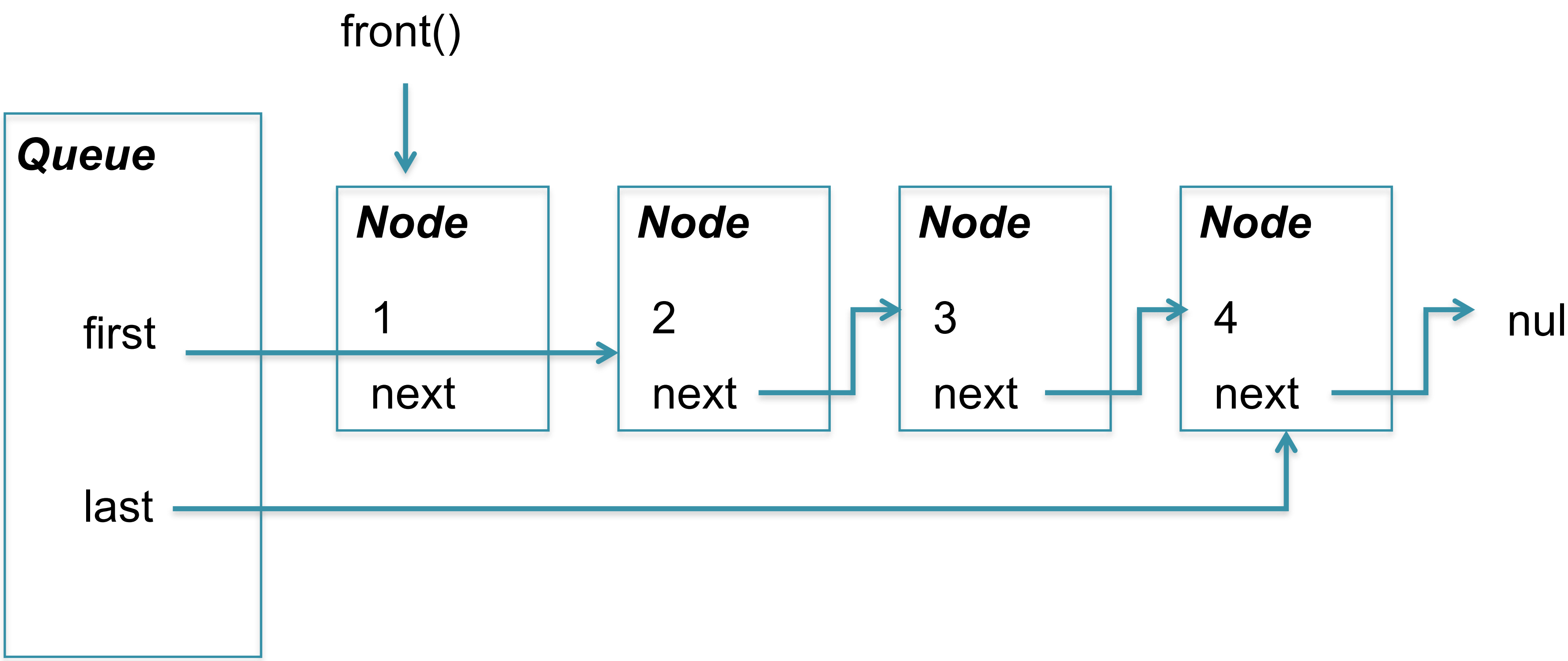


Now try dequeueing at first

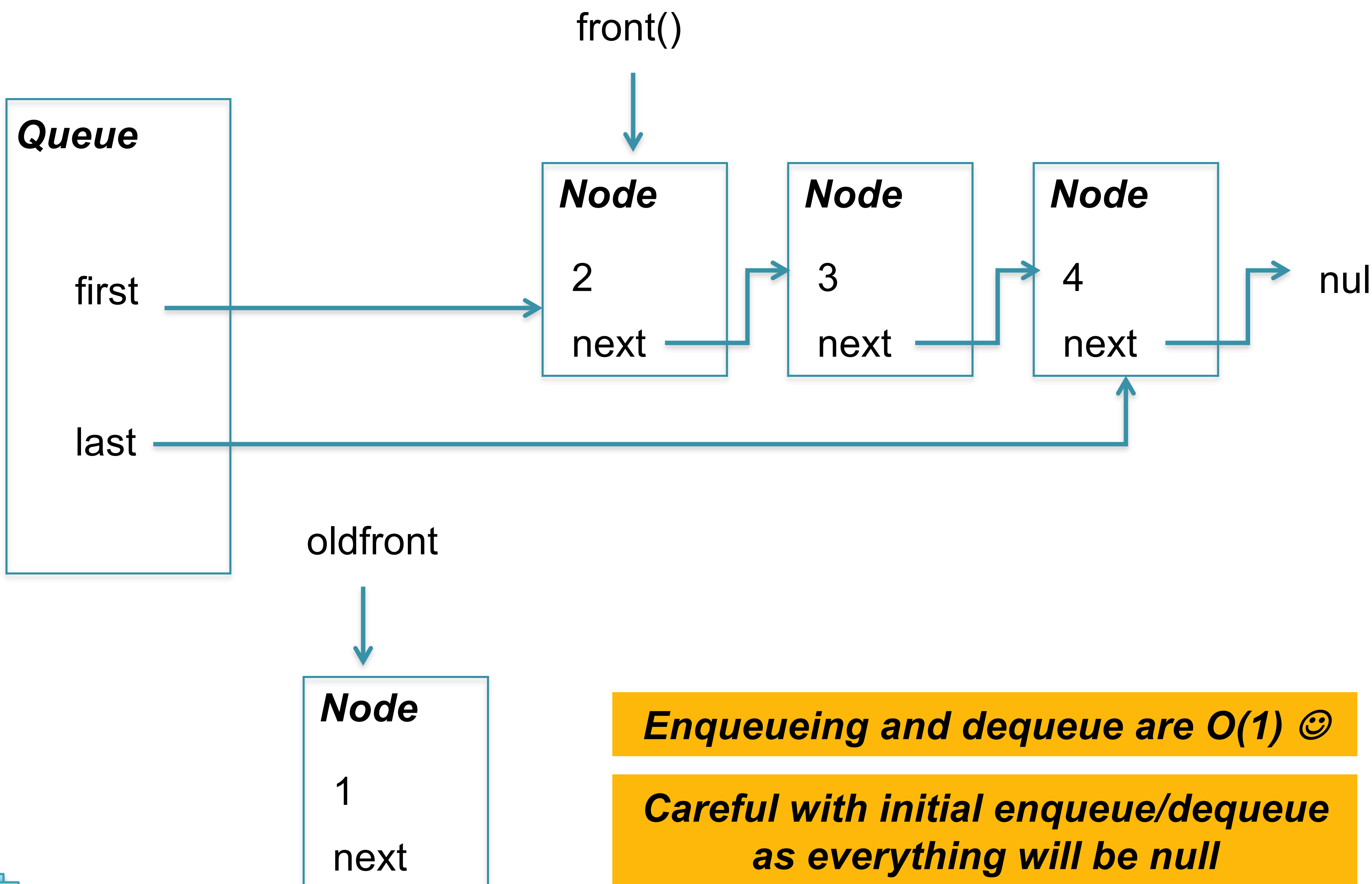
Enqueue last, Dequeue first



Enqueue last, Dequeue first



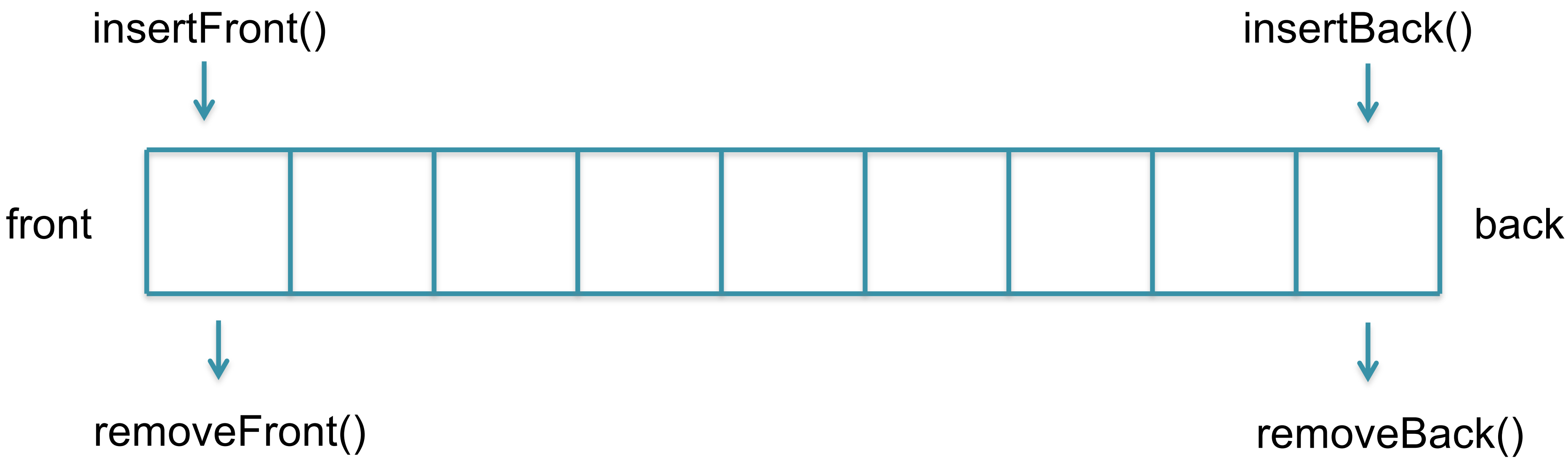
Enqueue last, Dequeue first



Enqueueing and dequeue are O(1) ☺

Careful with initial enqueue/dequeue as everything will be null

Dequeues



Dynamic Data Structure used for storing sequences of data

- Insert/Remove at either end in O(1)

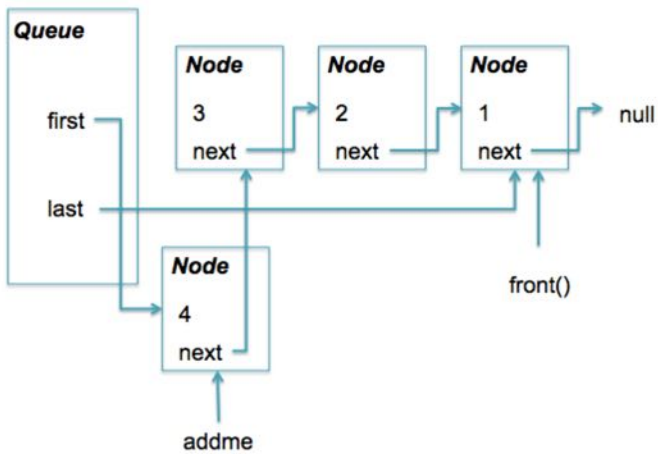
- If you exclusively add/remove at one end, then ***it becomes a stack***

- If you exclusive add to one end and remove from other, then ***it becomes a queue***

- Many other applications:
  - browser history: deque of last 100 webpages visited

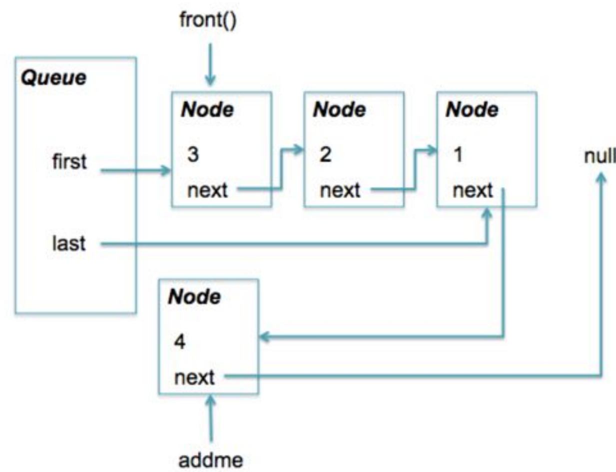
List Queue

insertFront



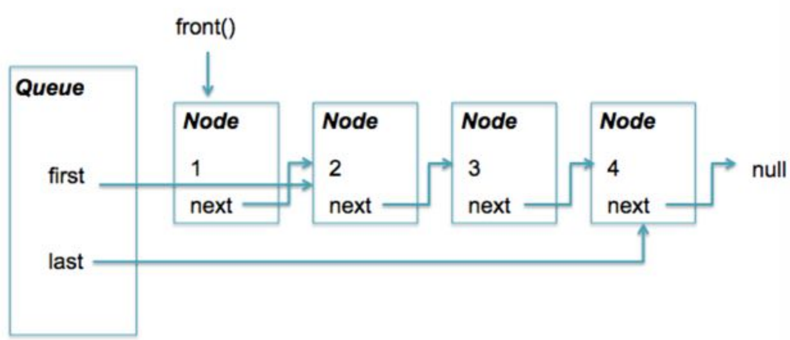
addme.next = first; first = addme;

insertBack



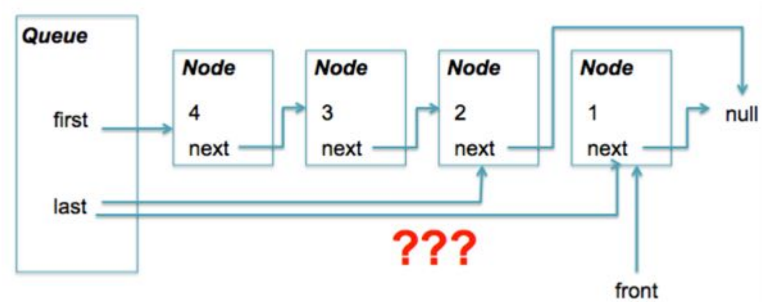
last.next = addme; addme.next = null

removeFront



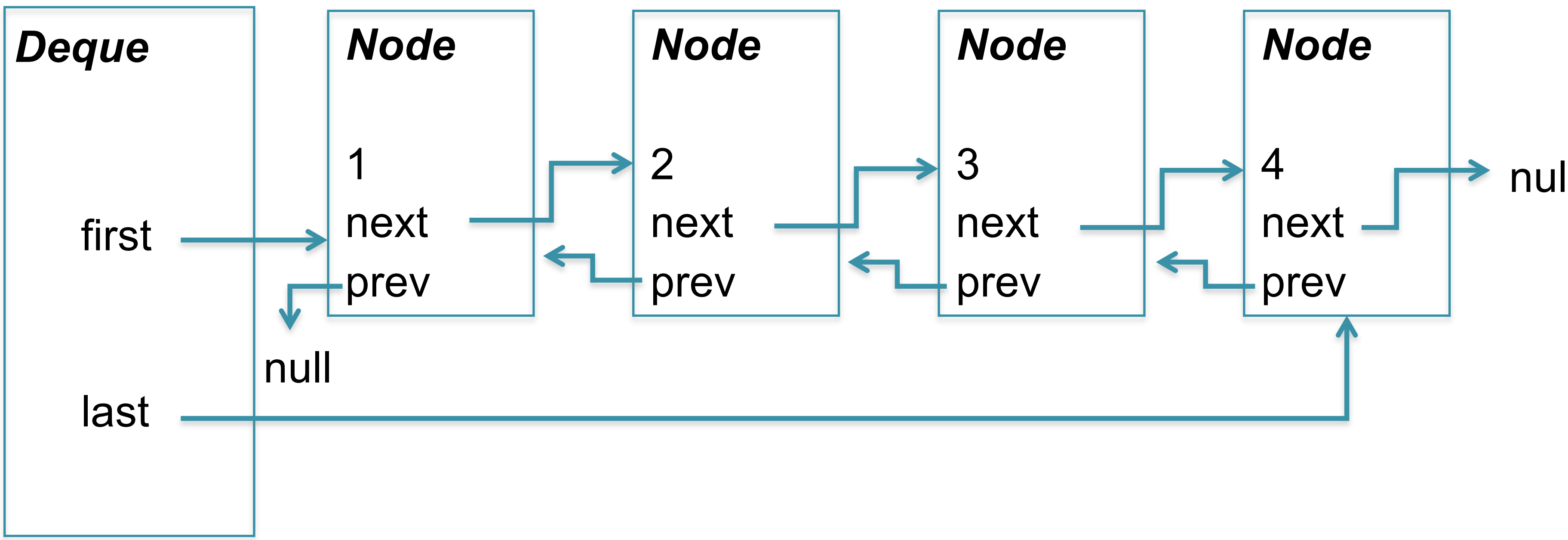
first = first.next;

removeBack



???

Deque with Doubly Linked List



Very similar to a singly linked list, except each node has a reference to both the next and previous node in the list

A little more overhead, but significantly increased flexibility: supports insertFront(), insertBack(), removeFront(), removeBack(), insertBefore(), removeMiddle()

Trees and Graphs

For each data structure discuss:

- Explain the interface
- Explain/Draw how it will be implemented
- Explain/Draw how to add/remove elements
- Iterate through the elements
- Explain the complexity of these

Trees!

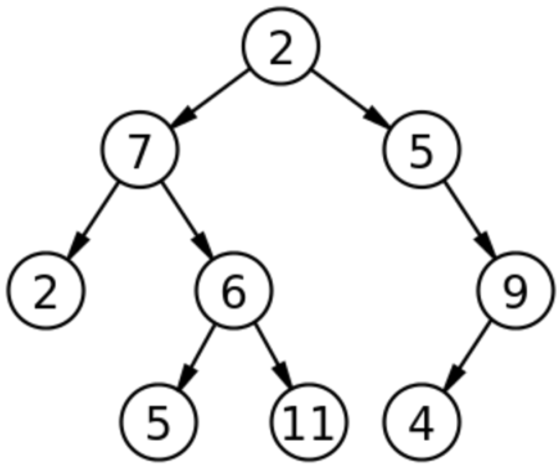
- All of the above
- How to implement pre-, in-, post-, level-order traversal

Graphs!

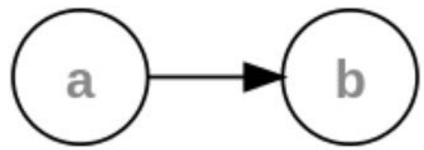
- All of the above
- How to implement DFS vs BFS

Trees are all around us ☺

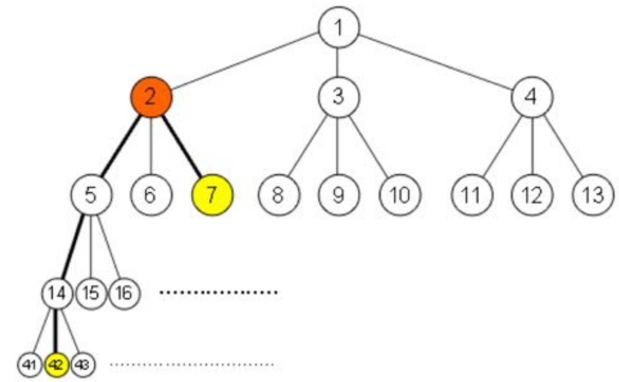
Types of Trees



Unordered
Binary tree



Linear
List

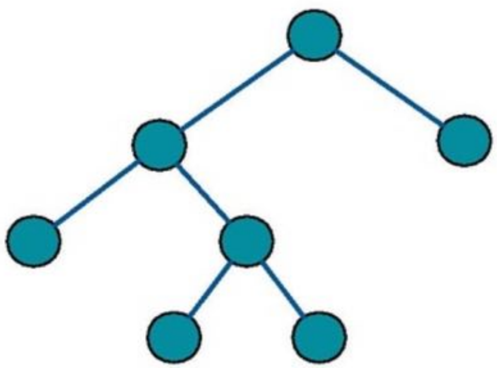


3-ary Tree
(k-ary tree has k children)

Single root node (no parent)
Each ***non-root*** node has at most 1 parent
Node may have 0 or more children

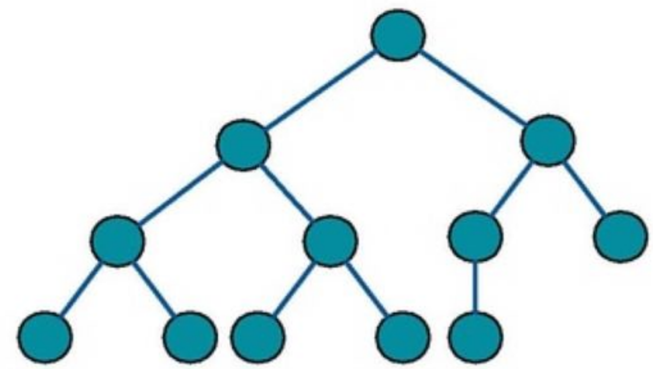
Internal node: has children; includes root unless tree is just root
Leaf node (aka external node): no children

Special Trees



Height of root = 0

Total Height = 3

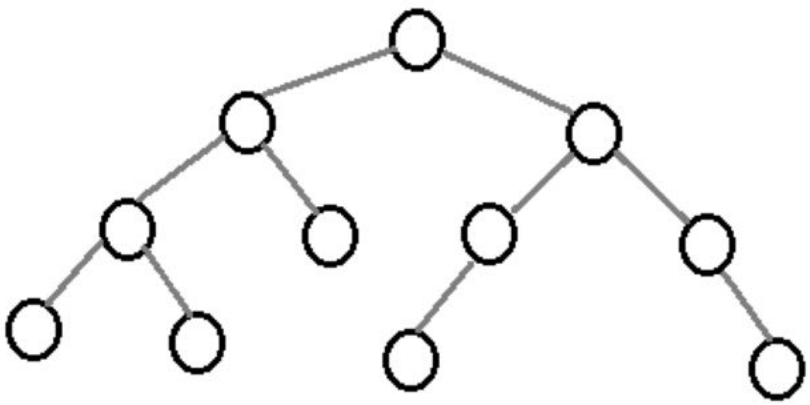


Full Binary Tree
Every node has 0 or 2 children

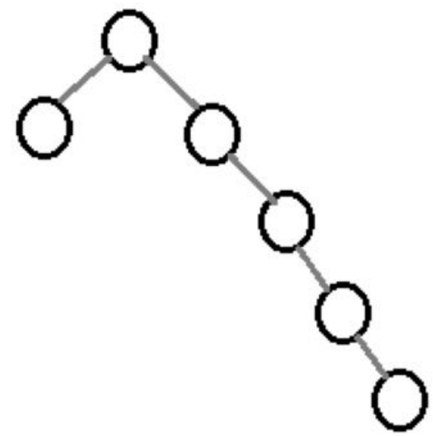
Complete Binary Tree
Every level full, except potentially the bottom level

What is the maximum number of leaf nodes in a complete binary tree?	2^h
What is the maximum number of nodes in a complete binary tree?	$2^{h+1} - 1$
What fraction of the nodes in a complete binary tree are leaves?	about half

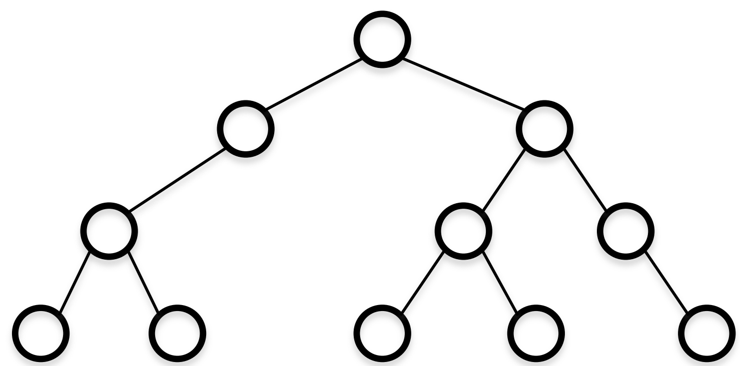
Balancing Trees



Balanced Binary Tree
Minimum possible height

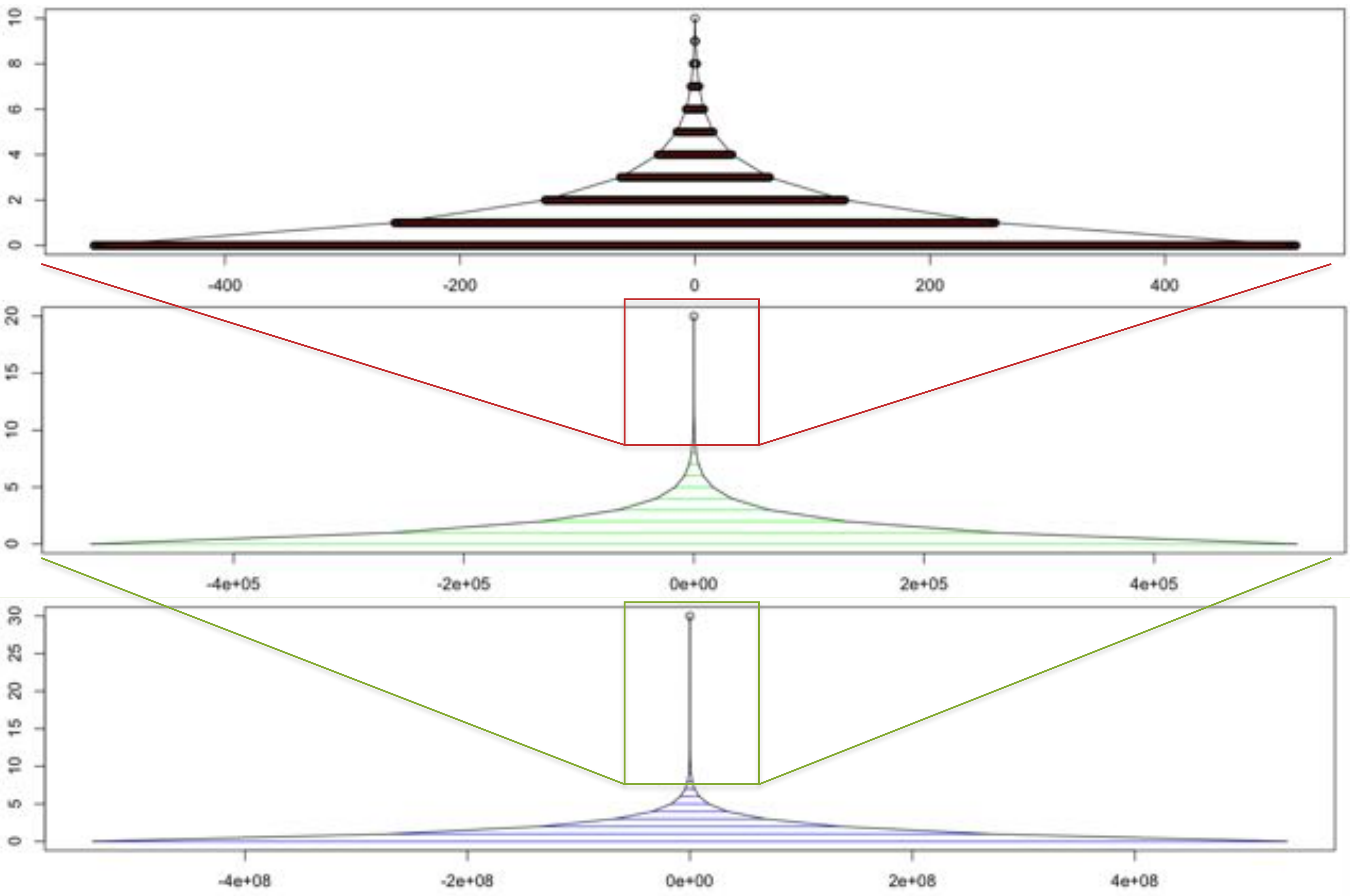


Unbalanced Tree
Non-minimum height

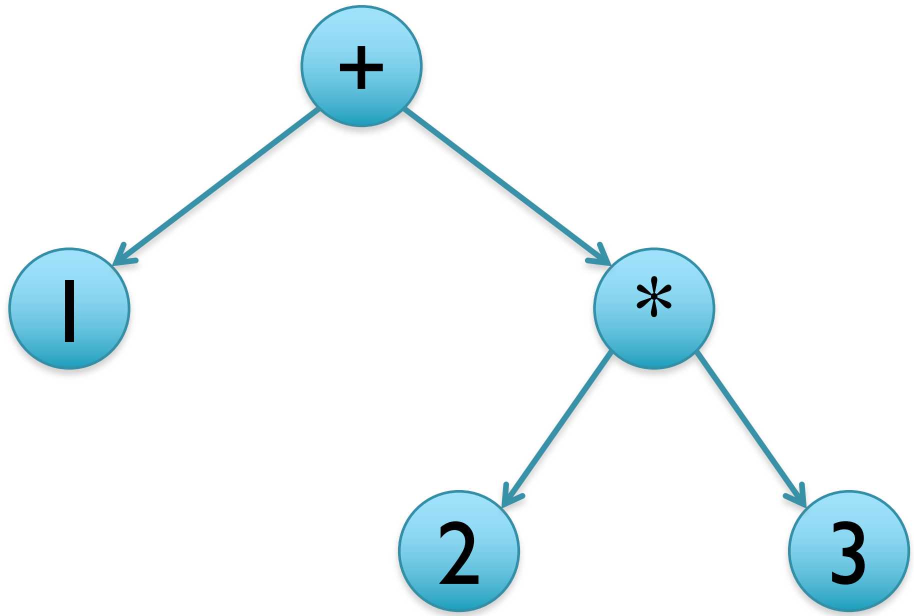


Balanced but not complete!

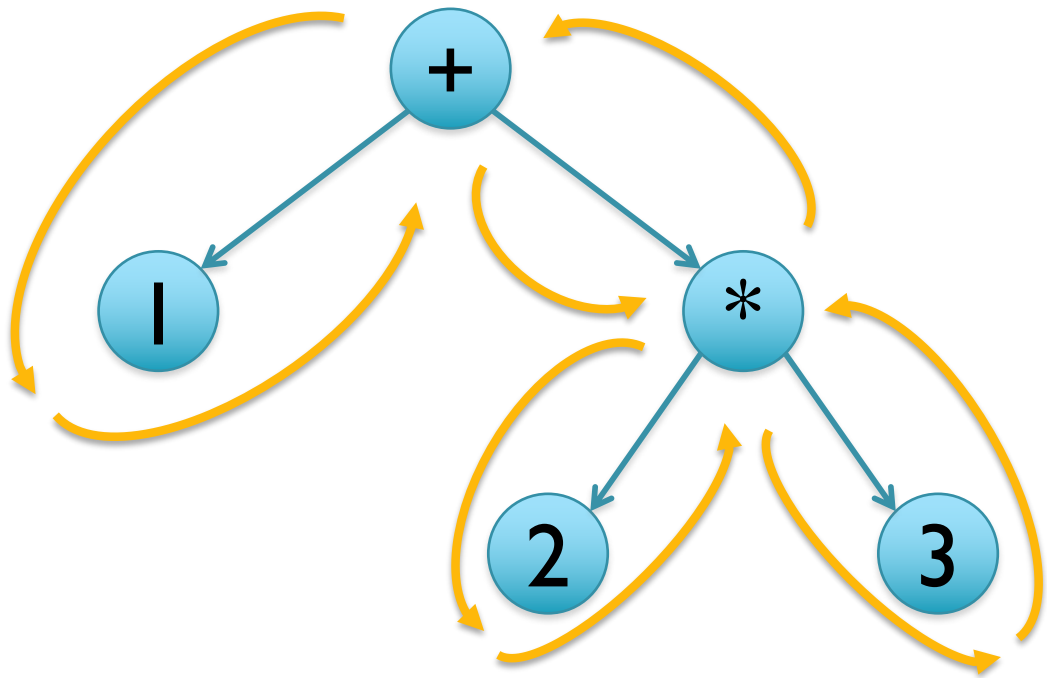
Tree Heights



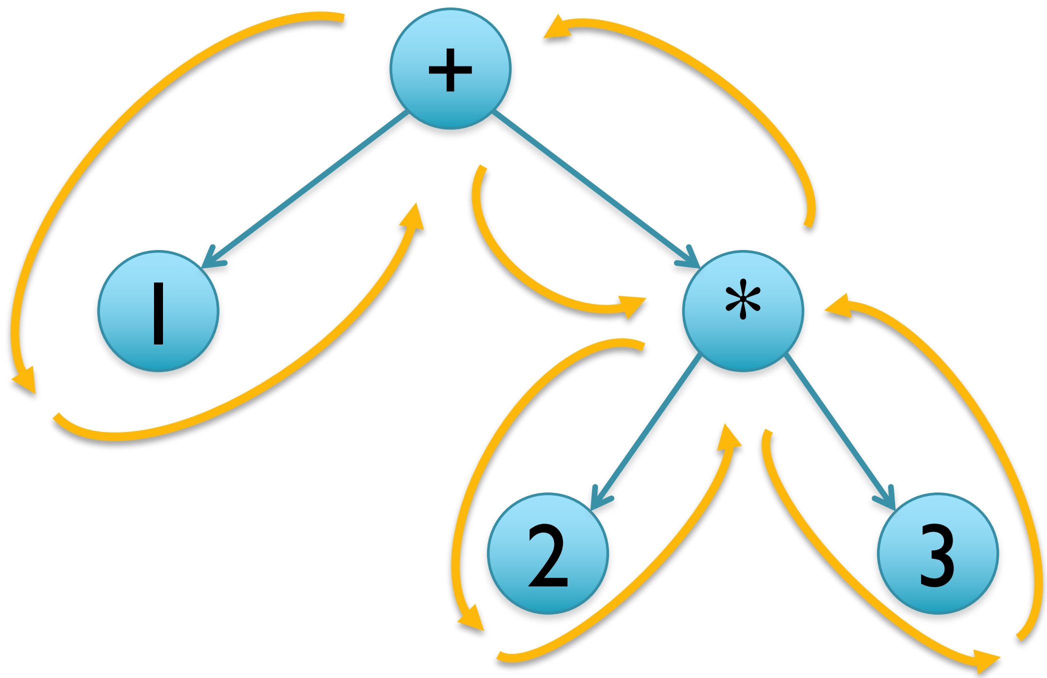
Tree Traversals



Tree Traversals



Tree Traversals



Note here we visit children from left to right, but could go right to left

Notice we visit internal nodes 3 times:
1: stepping down from parent
2: after visiting first child
3: after visiting second child

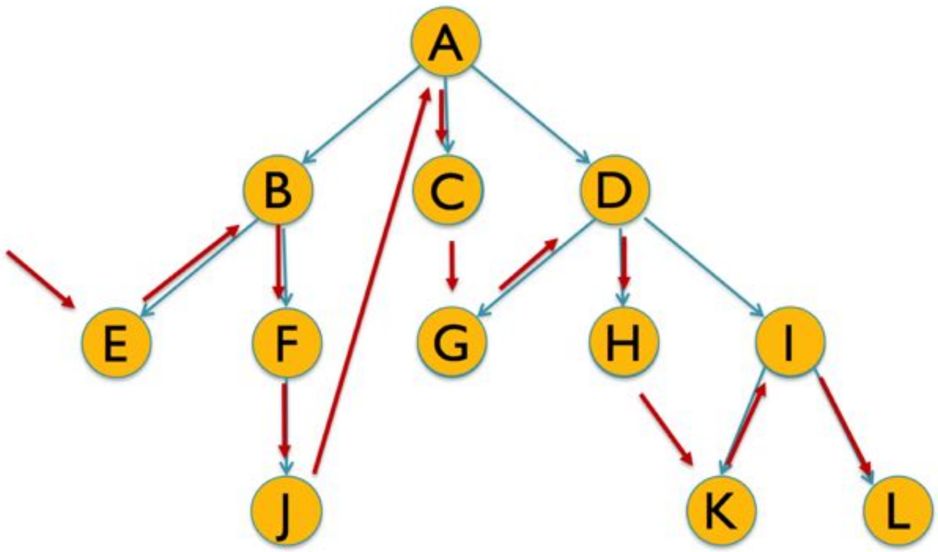
Different algs work at different times
1: preorder + 1 * 2 3
2: inorder 1 + 2 * 3
3: postorder 1 2 3 * +

InOrder vs PostOrder

What is the inorder print?

EBFJ AC GDHKIL

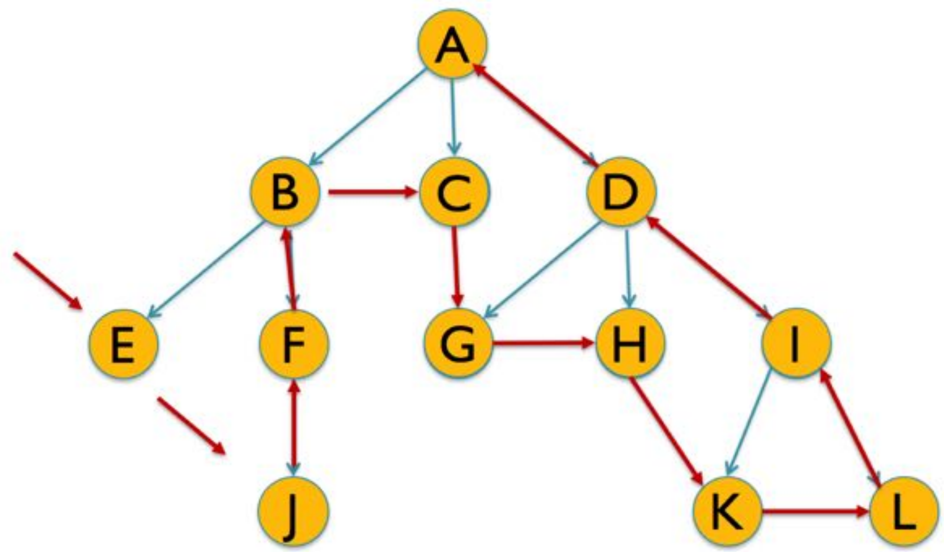
```
InOrderTraversal(Node n):
  if n is not null
     InOrderTraversal(n.left)
     print(n)
     InOrderTraversal(n.middle)
     InOrderTraversal(n.right)
```



What is the postorder print?

EJFB C GHKLIDA

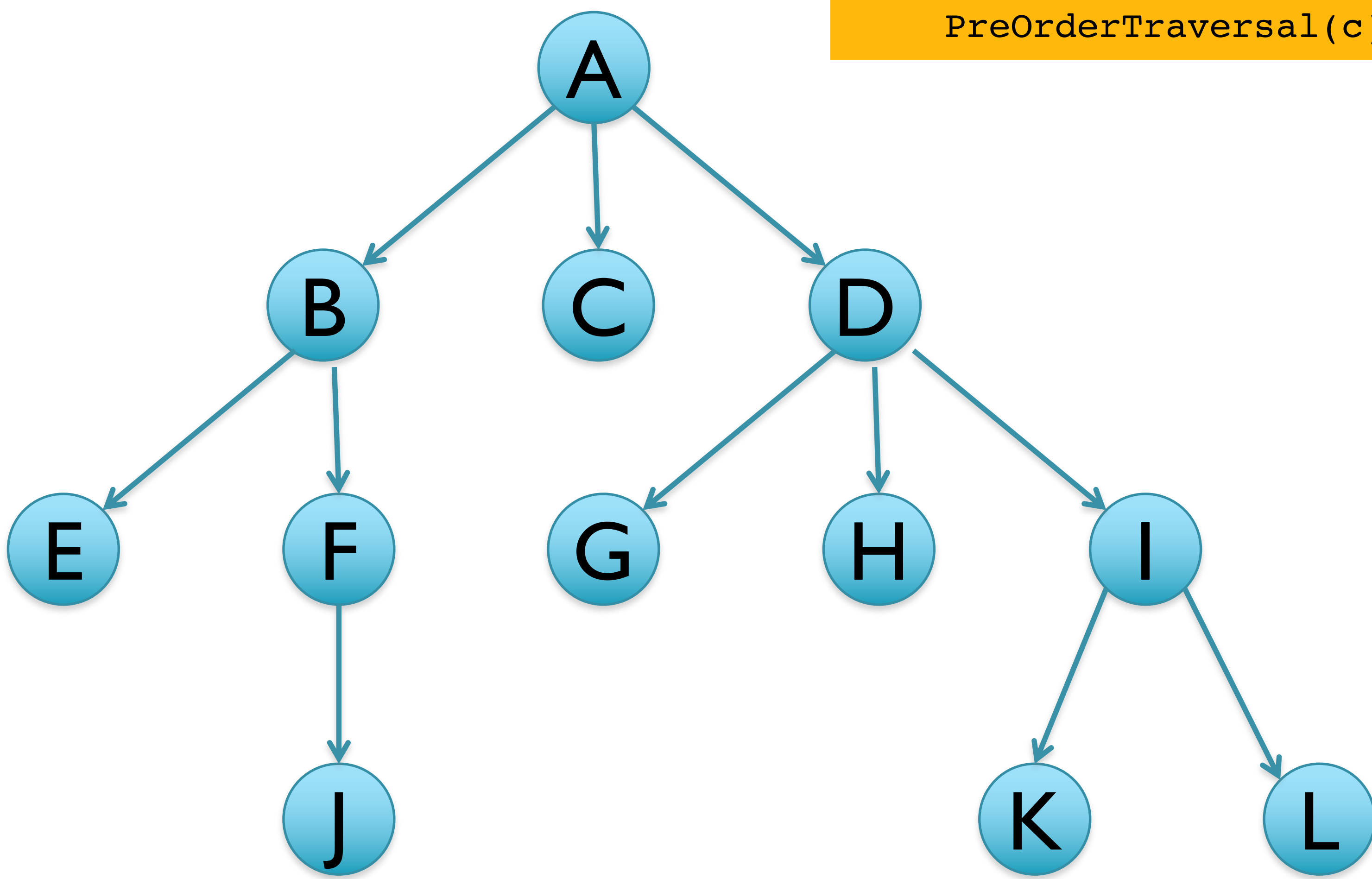
```
PostOrderTraversal(Node n):
  for c in x.children:
     PostOrderTraversal(c)
  print(n)
```



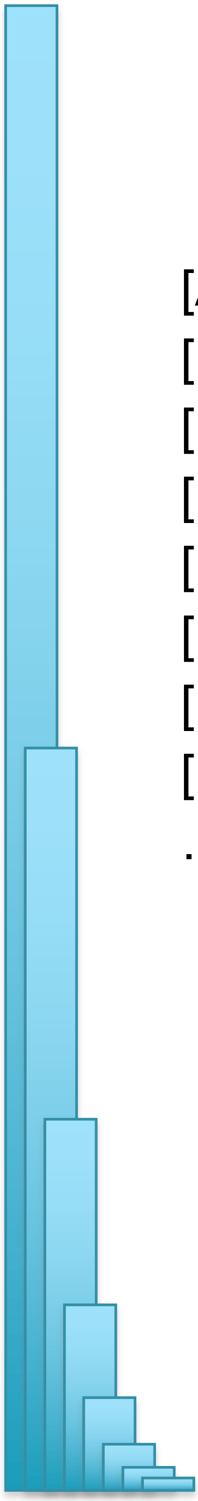
PreOrder Traversals

How to preorder print?

```
PreOrderTraversal(Node n):
 print(n)
 for c in x.children:
     PreOrderTraversal(c)
```



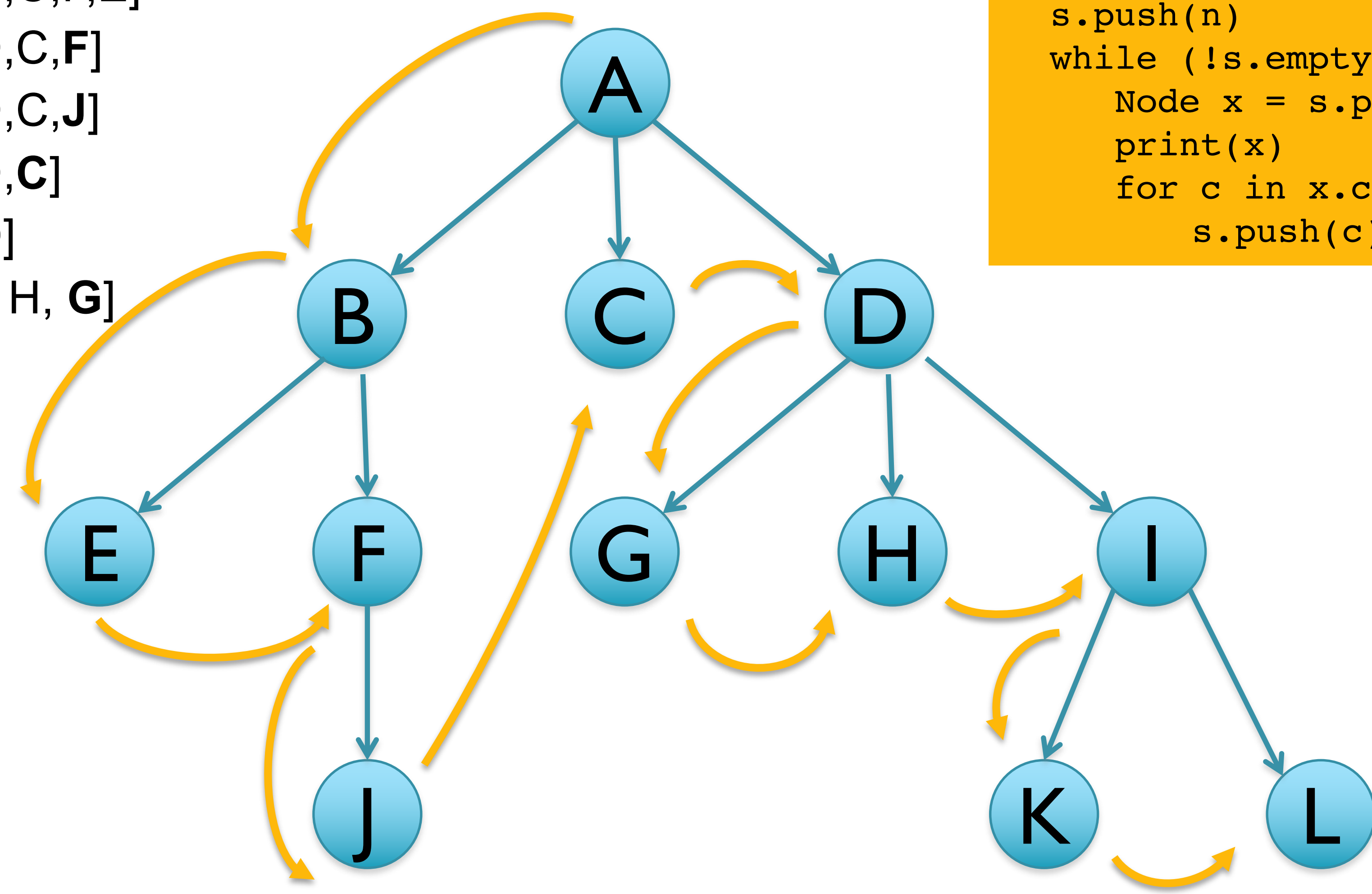
PreOrder Traversals



[**A**]
[D,C,**B**]
[D,C,F,**E**]
[D,C,**F**]
[D,C,**J**]
[D,**C**]
[**D**]
[I, H, **G**]
…

How to preorder print?

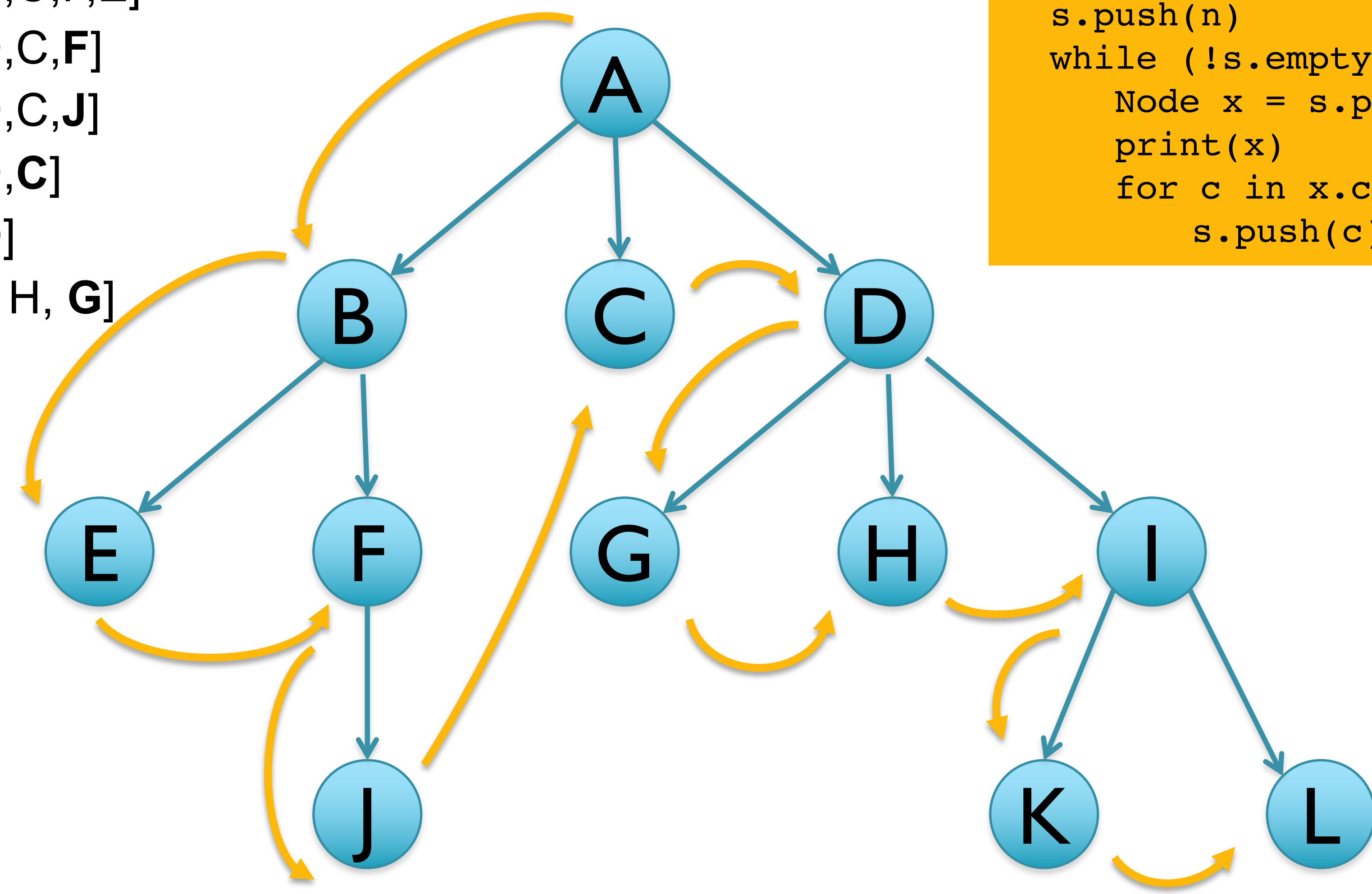
```
PreOrderTraversal(Node n):
  Stack s
  s.push(n)
  while (!s.empty()):
     Node x = s.pop()
     print(x)
     for c in x.children:
        s.push(c)
```



PreOrder Traversals

[**A**]
[D,C,**B**]
[D,C,F,**E**]
[D,C,**F**]
[D,C,**J**]
[D,**C**]
[**D**]
[I, H, **G**]
…

Stack leads to a Depth-First Search



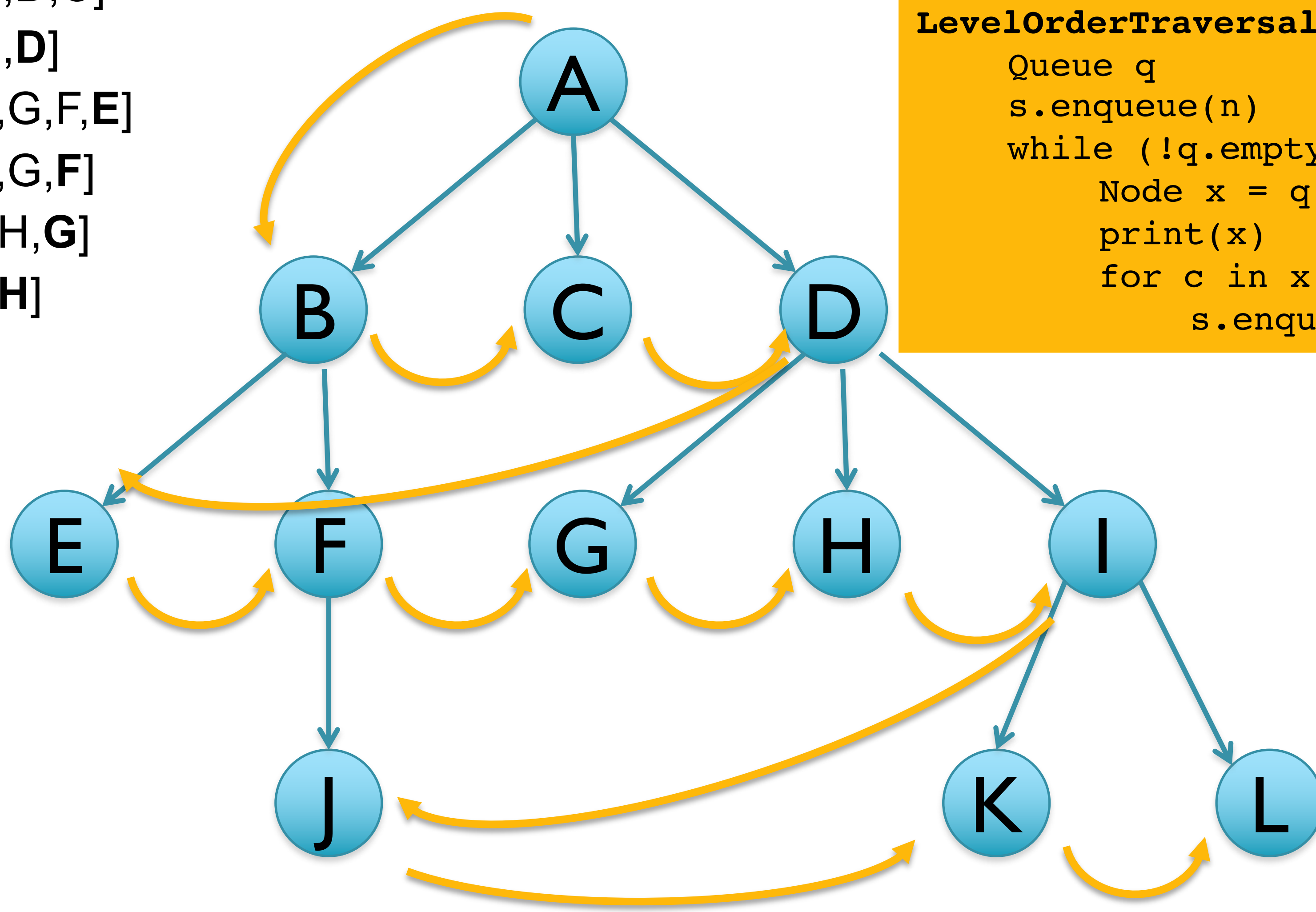
How to preorder print?

```
PreOrderTraversal(Node n):
  Stack s
  s.push(n)
  while (!s.empty()):
     Node x = s.pop()
     print(x)
     for c in x.children:
        s.push(c)
```

Level Order Traversals

[**A**]
[D,C,**B**]
[F,E,D,**C**]
[F,E,**D**]
[I,H,G,F,**E**]
[I,H,G,**F**]
[J,I,H,**G**]
[J,I,**H**]
…

Queue leads to a Breadth-First Search



How to level order print?

(A) (B C D) (E F G H I) (J K L)

```
LevelOrderTraversal(Node n):
    Queue q
    s.enqueue(n)
    while (!q.empty()):
        Node x = q.dequeue()
        print(x)
        for c in x.children:
            s.enqueue(c)
```

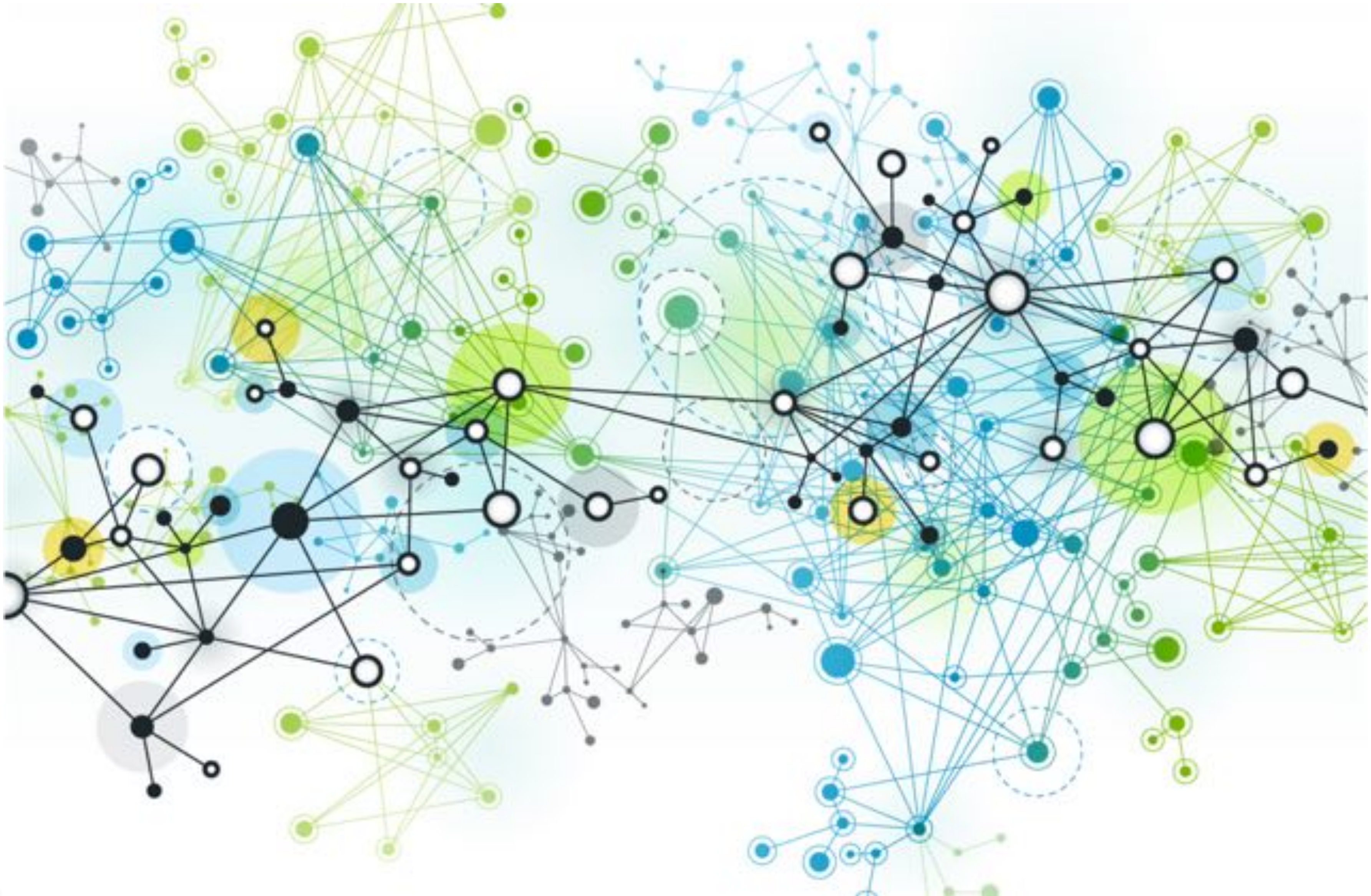
Multiple Traversals

```
public abstract class Operation<T> {
    void pre(Position<T> p) {}
    void in(Position<T> p) {}
    void post(Position<T> p) {}
}

public interface Tree<T> {
     ...
     traverse(Operation<T> o);
    ...
}

  // Tree implementation pseudo-code:
  niceTraversal(Node n, Operation o):
     if n is not null:
         o.pre(n)
         niceTraversal(n.left, o)
         o.in(n)
         niceTraversal(n.right, o)
         o.post(n)
}
```

Abstract class simplifies the use of function objects - functors

Client extends Operation<T> but overrides just the methods that are needed ☺

Graphs are Everywhere!



Computers in a network, Friends on Facebook, Roads & Cities on GoogleMaps, Webpages on Internet, Cells in your body, …

BFS

```
BFS(start, stop)
// initialize all nodes dist = -1
start.dist = 0
list.addEnd(start)
while (!list.empty())
  cur = list.begin()
  if (cur == stop)
    print cur.dist;
  else
    foreach child in cur.children
      if (child.dist == -1)
        child.dist = cur.dist+1
        list.addEnd(child)
```

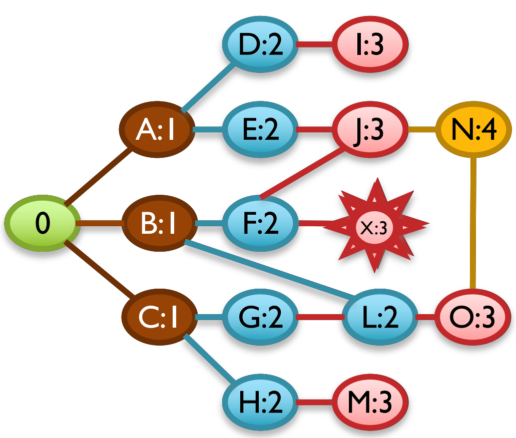
0

A,B,C
B,C,D,E
C,D,E,F,L

D,E,F,L,G,H
E,F,L,G,H,I
F,L,G,H,I,J
L,G,H,I,J,X
G,H,I,J,X,O
H,I,J,X,O

I,J,X,O,M
J,X,O,M
X,O,M,N
O,M,N
M,N

N



[How many nodes will it visit?]

[What's the running time?]

[What happens for disconnected components?]

BFS

```
BFS(start, stop)
// initialize all nodes dist = -1
start.dist = 0
list.addEnd(start)
while (!list.empty())
  cur = list.begin()
  if (cur == stop)
    print cur.dist;
  else
    foreach child in cur.children
      if (child.dist == -1)
        child.dist = cur.dist+1
        list.addEnd(child)
```

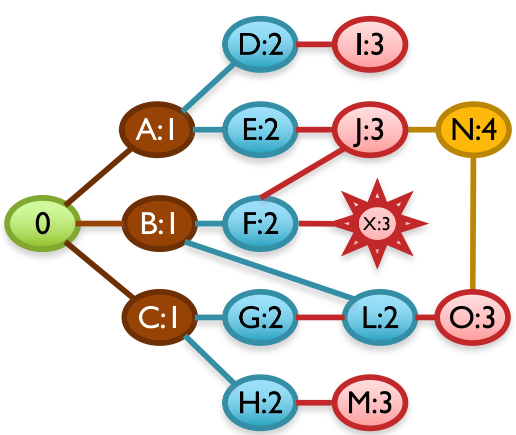
0

A,B,C
B,C,D,E
C,D,E,F,L

D,E,F,L,G,H
E,F,L,G,H,I
F,L,G,H,I,J
L,G,H,I,J,X
G,H,I,J,X,O
H,I,J,X,O

I,J,X,O,M
J,X,O,M
X,O,M,N
O,M,N
M,N

N



DFS

```
DFS(start, stop)
// initialize all nodes dist = -1
start.dist = 0
list.addEnd(start)
while (!list.empty())
  cur = list.end()
  if (cur == stop)
    print cur.dist;
  else
    foreach child in cur.children
      if (child.dist == -1)
        child.dist = cur.dist+1
        list.addEnd(child)
```

0

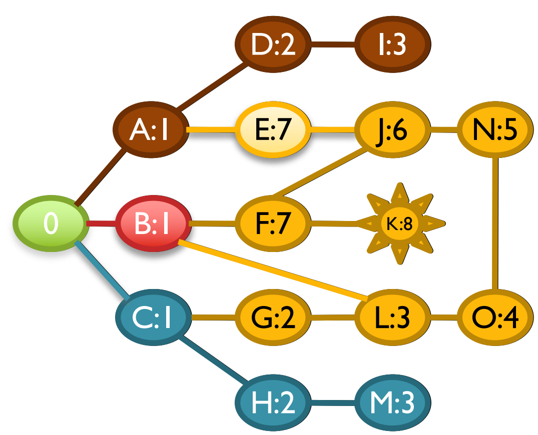
A,B,C

A,B,G,H
A,B,G,M

A,B,G
A,B,L
A,B,O
A,B,N
A,B,J
A,B,E,F
A,B,E,K
A,B,E

A,B

A
D
I



BFS: Queue

What is the runtime complexity?

What is the space complexity?

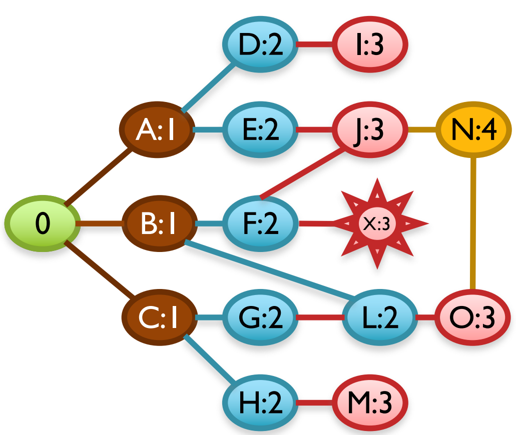
```
BF...
// i...
sta...
list...
while (!list.empty())
  cur = list.begin()
  if (cur == stop)
    print cur.dist;
  else
    foreach child in cur.children
      if (child.dist == -1)
        child.dist = cur.dist+1
        list.addEnd(child)
```

B,C,D,E
C,D,E,F,L

D,E,F,L,G,H
E,F,L,G,H,I
F,L,G,H,I,J
L,G,H,I,J,X
G,H,I,J,X,O
H,I,J,X,O

I,J,X,O,M
J,X,O,M
X,O,M,N
O,M,N
M,N

N



DFS: Stack

What is the runtime complexity?

What is the space complexity?

```
DF...
// i...
sta...
list...
while (!list.empty())
  cur = list.end()
  if (cur == stop)
    print cur.dist;
  else
    foreach child in cur.children
      if (child.dist == -1)
        child.dist = cur.dist+1
        list.addEnd(child)
```

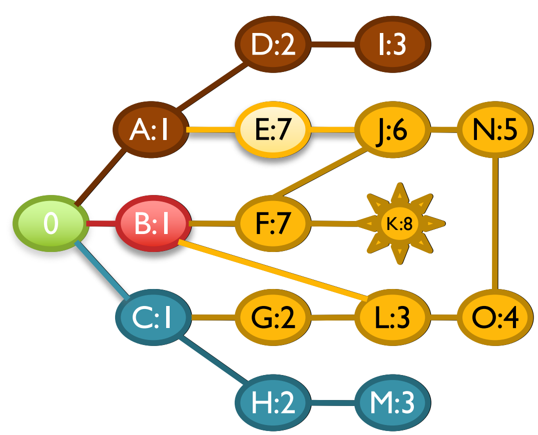
A,B,C

A,B,G,H
A,B,G,M

A,B,G
A,B,L
A,B,O
A,B,N
A,B,J
A,B,E,F
A,B,E,K
A,B,E

A,B

A
D
I



Complexity Analysis

How long will the algorithm take when run on inputs of different sizes:

- If it takes X seconds to process 1000 items, how long will it take to process twice as many (2000 items) or ten times as many (10,000 items)?

Generally looking for an order of magnitude estimate:

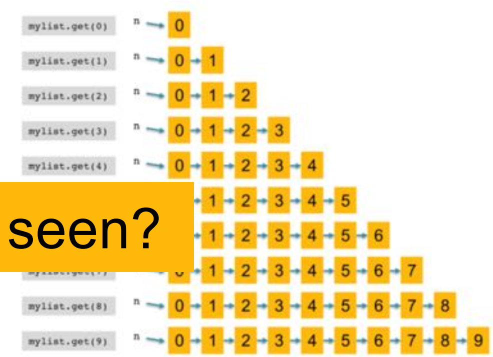
Constant time

Accessing 1st or 1 billionth entry from an array takes same amount of time

Linear time

Takes 10 times longer to scan a list that has 10 times as many values

Quadradic time



Nested loops grows with the square of the list length

What's another complexity we have seen?

Also very important for space characterization:

Sometimes doubling the number of elements will more than double the amount of space needed

FindMax Analysis

```
public static int findMaximum(int [] myarray) {
   int max = myarray[0];
   for (int i = 1; i < myarray.length; i++) {
     if (myarray[i] > max) {
       max = myarray[i];
     }
   }

   return max;
 }
```

What is the total amount of work done?

$T(n) = C(n) + A(n) = (2n) + (3n - 1) = 5n-1$

Should we worry about the "-1"?

Nah, for sufficiently large inputs will make a tiny difference

Should we worry about the 5n?

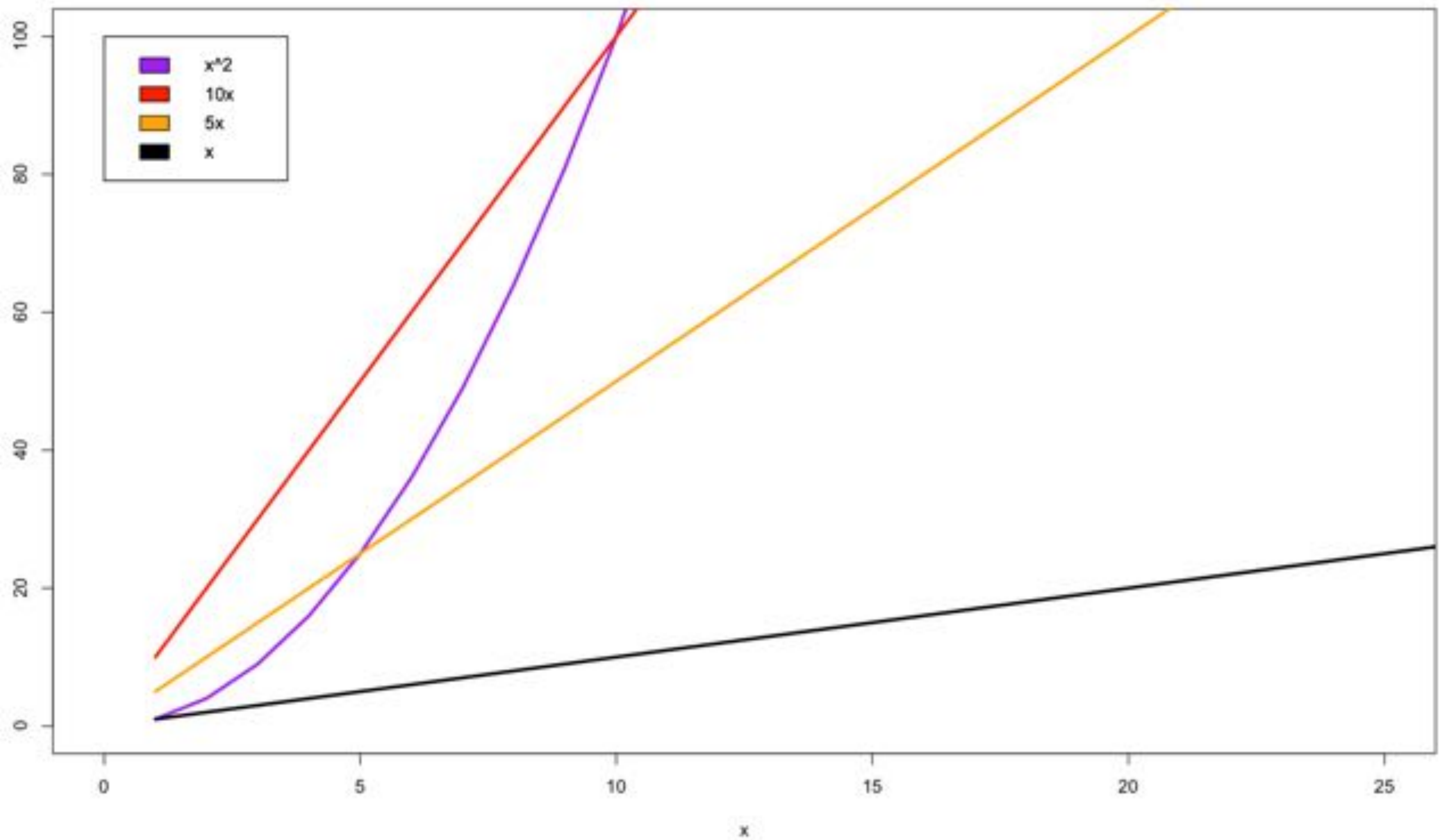
Nah, the runtime is linearly proportional to the length of the array

Big-O Notation

- ***Formally, algorithms that run in O(X) time means that the total number of steps (comparisons and assignments) is a polynomial whose largest term is X, aka asymptotic behavior***
  - ***$f(x) \in O(g(x))$ if there exists $c > 0$ (e.g., $c = 1$) and x_0 (e.g., $x_0 = 5$) such that $f(x) \le cg(x)$ whenever $x \ge x_0$***
    - $T(n) = 33$ => $O(1)$
    - $T(n) = 5n-2$ => $O(n)$
    - $T(n) = 37n^2 + 16n - 8$ => $O(n^2)$
    - $T(n) = 99n^3 + 12n^2 + 70000n + 2$ => $O(n^3)$
    - $T(n) = 127n \log(n) + \log(n) + 16$ => $O(n \lg n)$
    - $T(n) = 33 \log(n) + 8$ => $O(\lg n)$
    - $T(n) = 900*2^n + 12n^2 + 33n + 54$ => $O(2^n)$

- ***Informally, you can read Big-O(X) as "On the order of X"***
  - *O(1) => On the order of constant time*
  - *O(n) => On the order of linear time*
  - *$O(n^2)$ => On the order of quadratic time*
  - *$O(n^3)$ => On the order of cubic time*
  - *O(lg n) => On the order of logarithmic time*
  - *O(n lg n) => On the order of n log n time*

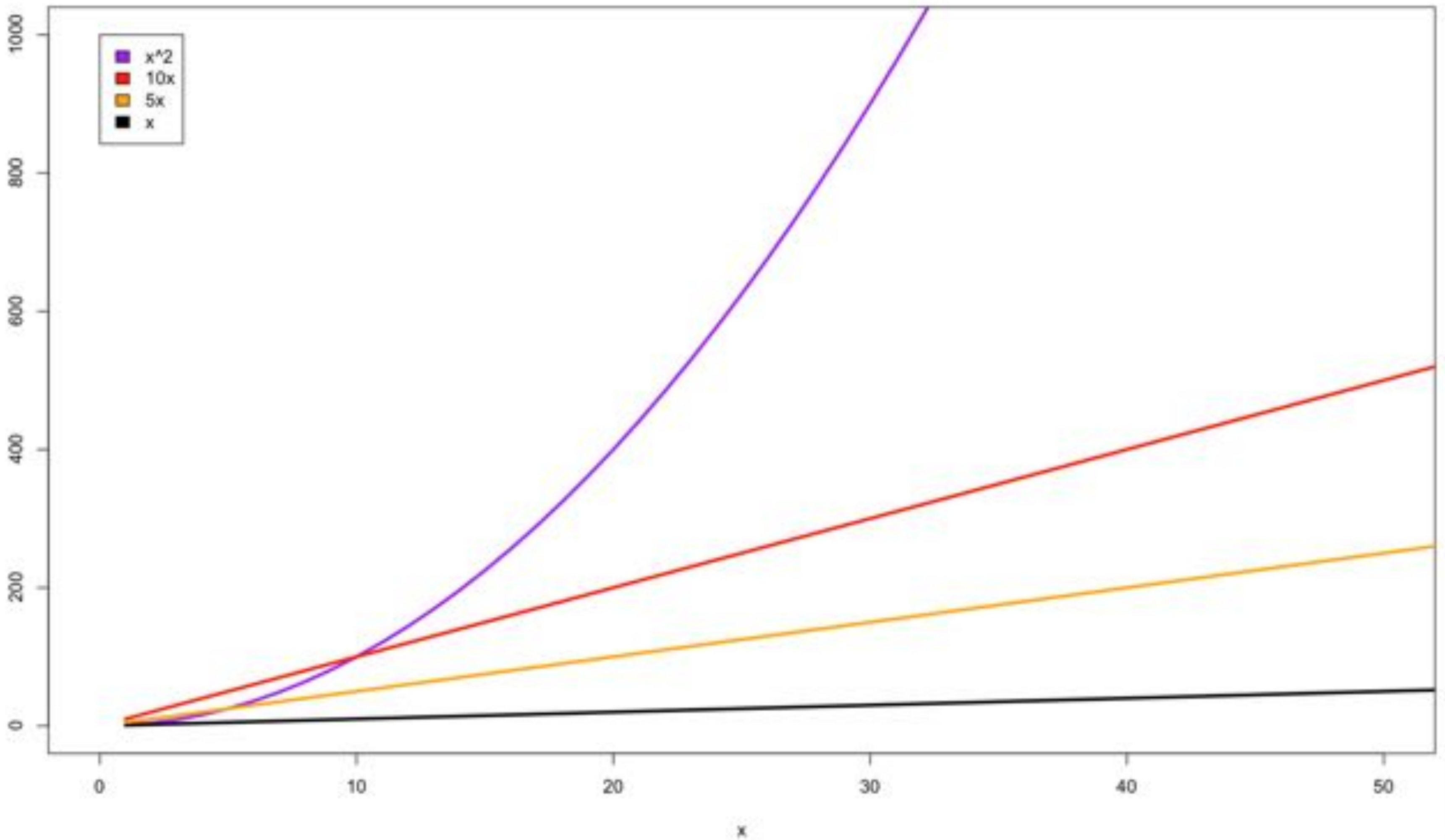
Growth of functions



A quadratic function isnt necessarily larger than a linear function for all possible inputs, but eventually will be

That largest polynomial term defines the Big-O complexity

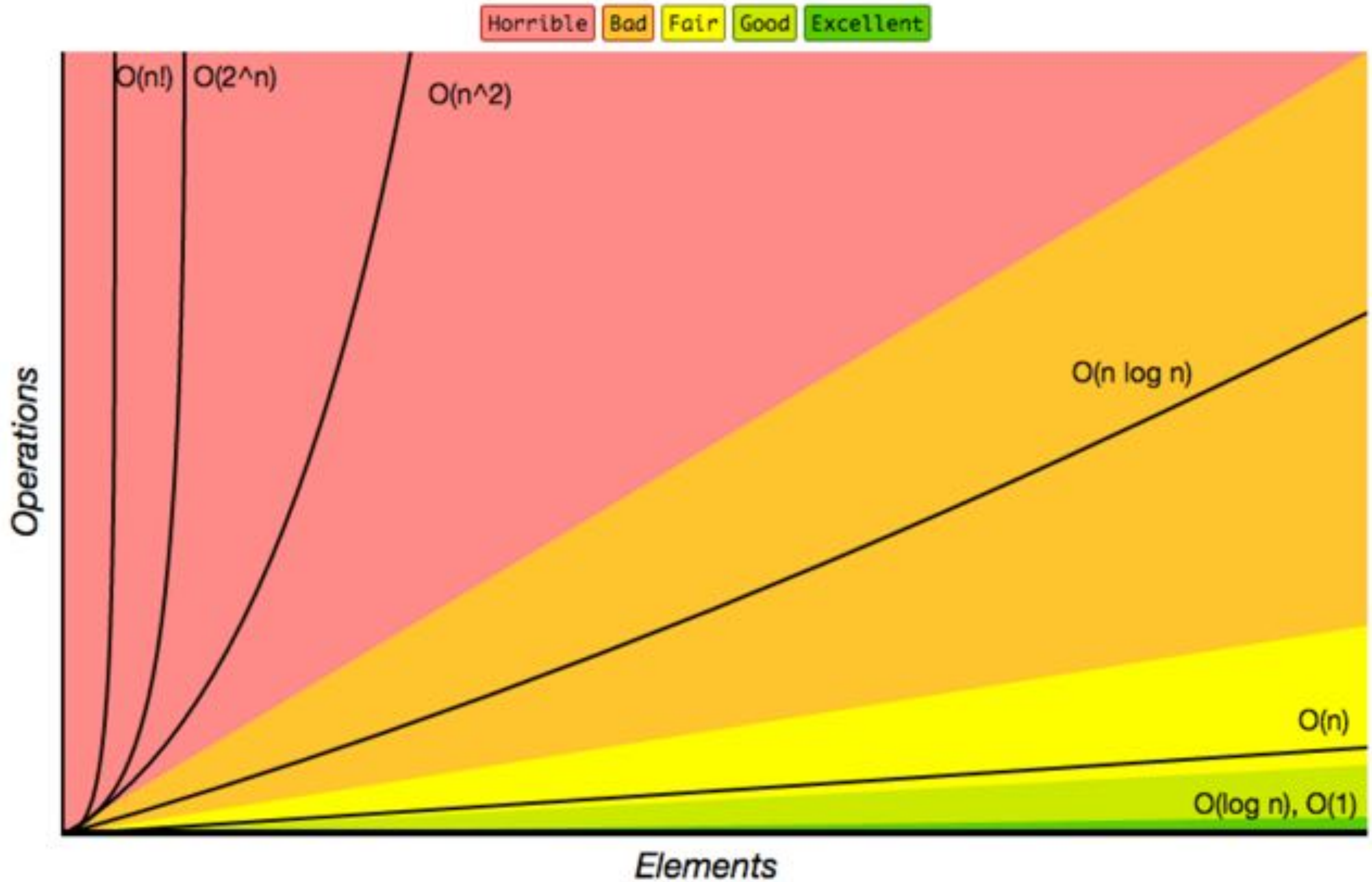
Growth of functions



A quadratic function isnt necessarily larger than a linear function for all possible inputs, but eventually will be

That largest polynomial term defines the Big-O complexity

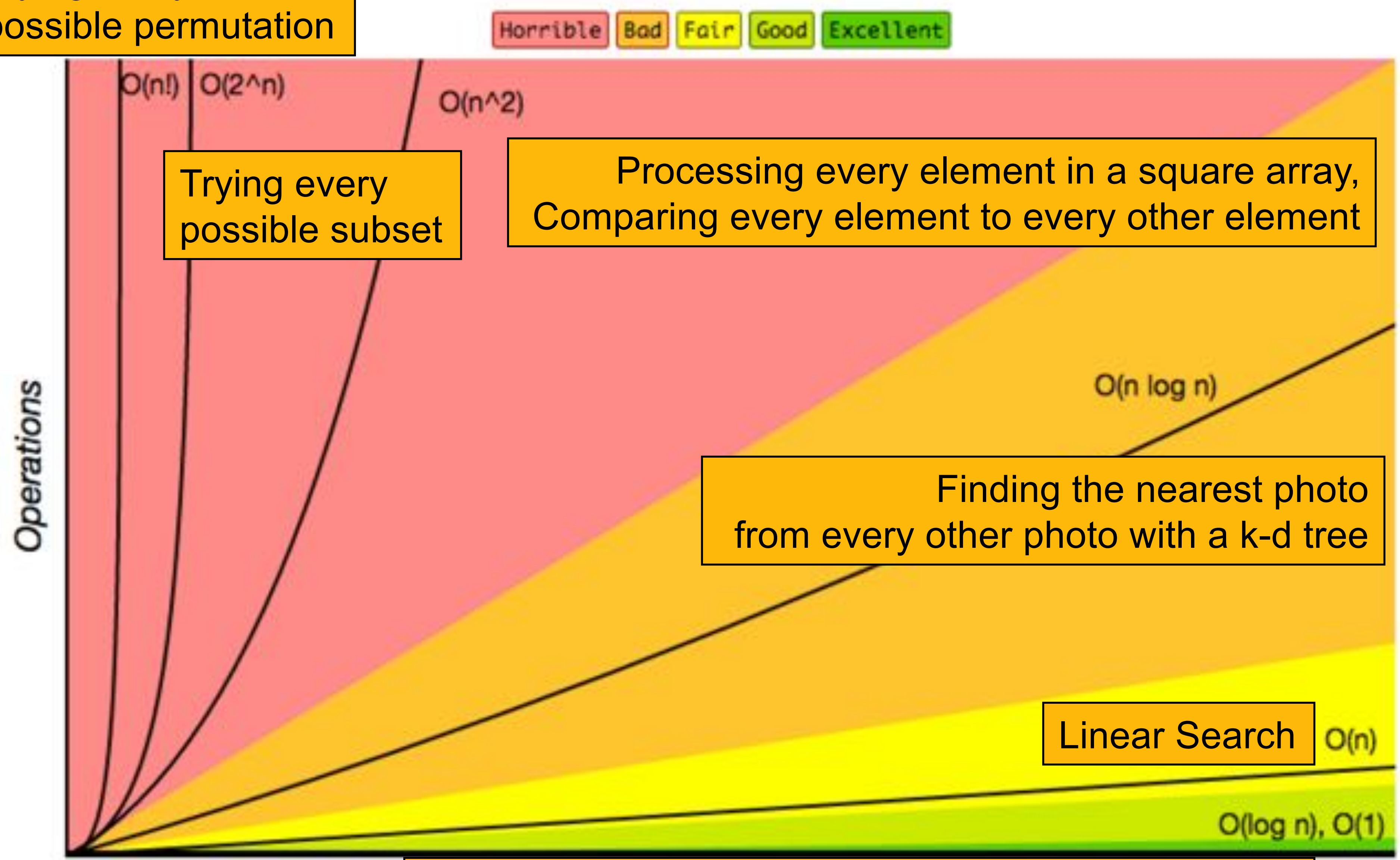
Growth of functions



http://bigocheatsheet.com/

Growth of functions

Trying every possible permutation



Trying every possible subset

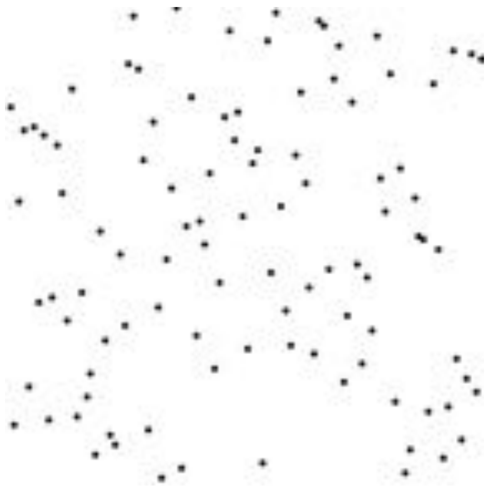
Processing every element in a square array,
Comparing every element to every other element

Finding the nearest photo
from every other photo with a k-d tree

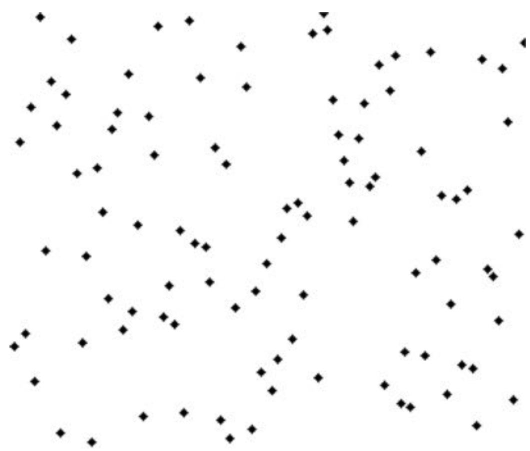
Linear Search

Finding an element in a balanced tree,
Simple arithmetic, simple comparison, array access

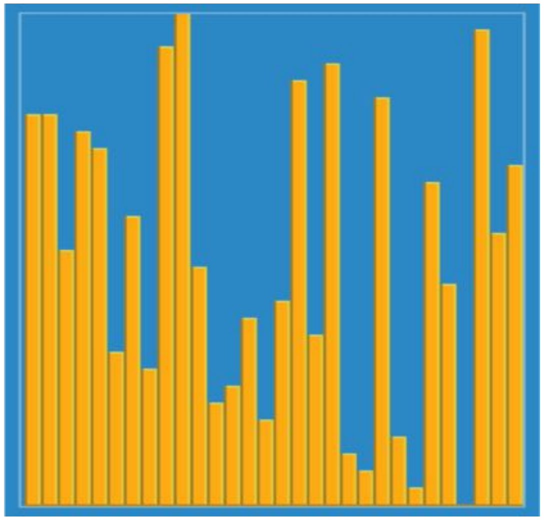
Quadratic Sorting Algorithms



Selection Sort
Move next smallest
into position



Bubble Sort
Swap up bigger
values over smaller



Insertion Sort
Slide next value into
correct position

Asymptotically all three have the same performance, but can differ for different types of data. HW 3 will compare them in more detail

600.226: Data Structures
Midterm

Peter H. Fröhlich
phf@cs.jhu.edu

July 29, 2013
Time: 40 Minutes <- 75 Minutes

Start here: Please fill in the following important information using a **permanent pen** before you do **anything** else! Your exam will **not** be graded if you use a pencil or erasable ink on this page.

Name (print): ______________________

Email (print): ______________________

Ethics Pledge: With your signature you **certify** the information above and you also **affirm** the following: *"I agree to complete this exam without unauthorized assistance from any person, materials, or device."*

Signature: ______________________

Date: ______________________

Instructions: Please read these instructions carefully before you start. **Switch off** your phones, pagers, and other noisy gadgets! You are **not** allowed to have anything but a pen (pencil, eraser) and this exam on your desk. You are **not** allowed to talk to anyone during the exam. If you have a question, please raise your hand **quietly**. You must **remain seated quietly** until all exams have been collected. Remember that you can **not** claim grading errors if you do not use a **permanent** pen for your answers.

Do not open before you are told to do so!

You got ________ out of 40 points.

Next Steps

1. Review for Midterm

2. Check on Piazza for tips & corrections!