✨ No Laptop Monday! ✨

Week10a

- binary search
- binary search trees

WARMUP

how long does it take to **find** (**search** for) something in...
- an array list?
- a linked list?

what if the list is sorted?

[record lecture]

review: sorted

sorted

sorted

- a sequence is **sorted** if its elements are "in order"
  - by convention, this means ascending order (going up from left to right)
    - `[1, 2, 5, 6, 9, 13]` is sorted
    - `[9, 5, 1, 2, 6, 13]` is **unsorted** (NOT sored)

search

search

- to **search** means to look for something (in some data structure)
  - a simple search problem is finding a given value in an array / list

```
// get index of the first element in array with this value
// returns -1 if value not found
int find(int[] array, int value) { ... }
```

```
// Option B
class FindResult {
    boolean success;
    int index;
}
FindResult find(int[] array, int value) { ... }
```

linear search
(of an array / array list)

linear search (brute force search)

- **linear search** looks at each element one by one
  - linear search works whether or not the list is sorted
  - linear search is **O(n)** (linear time) 😐

```
// get index of the first element in array with this value
// returns -1 if value not found
int linearSearch(int[] array, int value) {
    for (int i = 0; i < array.length; ++i) {
        if (array[i] == value) {
            return i;
        }
    }
    return -1;
}
```

example: linear search for 17

```
[13  2  55  7  17  100  77]
```

example: linear search for 17

```
[13  2  55  7  17  100  77]
```

example: linear search for 17

```
[13  2  55  7  17  100  77]
```

example: linear search for 17

```
[13  2  55  7  17  100  77]
```

example: linear search for 17

```
[13  2  55  7  17  100  77]
```

example: linear search for 17

```
[13  2  55  7  17  100  77]
```

example: linear search for 17

```
[13  2  55  7  17  100  77]
```

example: linear search for 17

```
[13  2  55  7  17  100  77]
```

example: linear search for 17

```
[13  2  55  7  17  100  77]
```

example: linear search for 17

[13 2 55 7 17 100 77]

example: linear search for 17

[13 2 55 7 17 100 77]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: linear search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

if we know that an array is sorted...
can we search it faster?

yes 🚀

unless it's a linked list
💀

binary search
(of a array / array list)

binary search

- **binary search** is an O(log n) algorithm for searching a sorted array 😊
  - binary search works by "cutting the array in half" over and over
  - ☠️ **binary search only applies if the array is sorted**

example: binary search for 17

```
[1  2  4  5  6  8  9  10  11  13  15  16  17  18  19]
```

example: binary search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

10 < 17

example: binary search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: binary search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

16 < 17

example: binary search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: binary search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

17 < 18

example: binary search for 17

[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

example: binary search for 17

```
[1  2  4  5  6  8  9  10  11  13  15  16  17  18  19]
                                                17 == 17
```

example: binary search for 17

```
[1  2  4  5  6  8  9  10  11  13  15  16  17  18  19]
```

implementating binary search is surprisingly tricky!

make sure you test thoroughly!

✨ **hint:** binary search

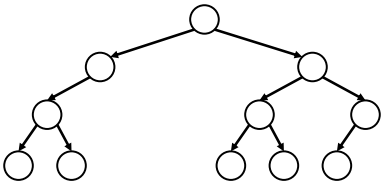
```
[1  2  4  5  6  8  9  10  11  13  15  16  17  18  19]
 ^                                                  ^
 |                                                  |
 i = 0                                              j
```

review: binary tree

binary tree

binary tree

- a **binary tree** is a tree in which each node has <= 2 children



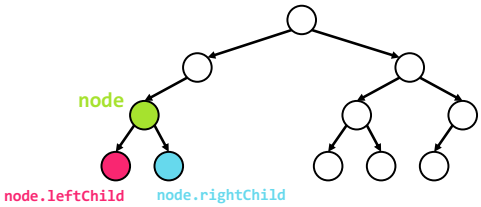
🧠 can a linked list be seen as a binary tree?

technically, yes (assuming no cycles)
all nodes have <= 2 children
(tail has 0 children, all other nodes have 1 child)

ordered binary tree

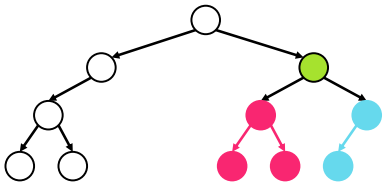
(ordered) binary tree

- each **node** has a **left child** and a **right child**
  (either or both children can be `null`)



(ordered) binary tree

- each node has a **left subtree** and a **right subtree**
  (and **left descendants** and **right descendants**)

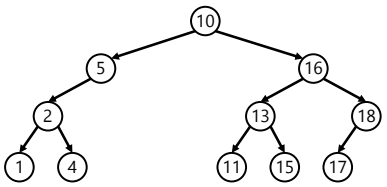


binary search tree

binary search tree

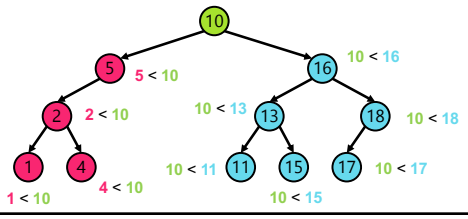
binary search tree

- in a **binary search tree** (**BST**, sorted binary tree) every node follows:
  "a node's value is **greater than the values of all nodes in its left subtree** and **less than the values of all nodes in its right subtree**"



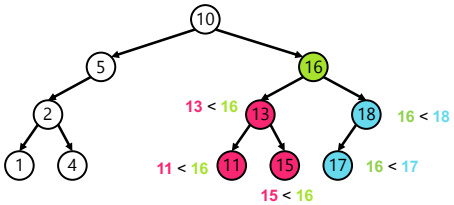
binary search tree

- in a **binary search tree** (**BST**, sorted binary tree) every node follows:
  "a node's value is **greater than the values of all nodes in its left subtree** and **less than the values of all nodes in its right subtree**"



binary search tree

- in a **binary search tree** (**BST**, sorted binary tree) every node follows:
  "a node's value is **greater than the values of all nodes in its left subtree** and **less than the values of all nodes in its right subtree**"



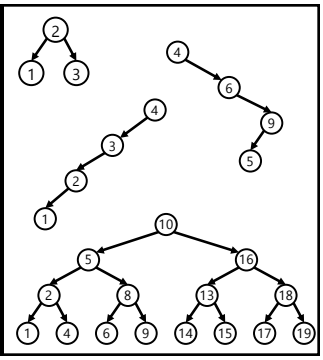
binary search tree

- in a **binary search tree** (**BST**, sorted binary tree) **every** node follows:
  "a node's value is **greater than the values of all nodes in its left subtree** and **less than the values of all nodes in its right subtree**"

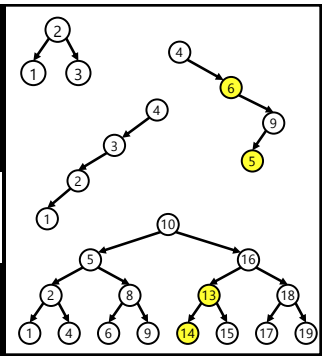
is it a BST?

Is It A Binary Search Tree?

in a **binary search tree** (sorted binary tree) every node follows:
"a node's value is **greater than the values of all nodes in its left subtree** and **less than the values of all nodes in its right subtree**"



in a **binary search tree** (sorted binary tree) every node follows:
"a node's value is **greater than the values of all nodes in its left subtree** and **less than the values of all nodes in its right subtree**"



binary search
(of a binary search tree)

binary search

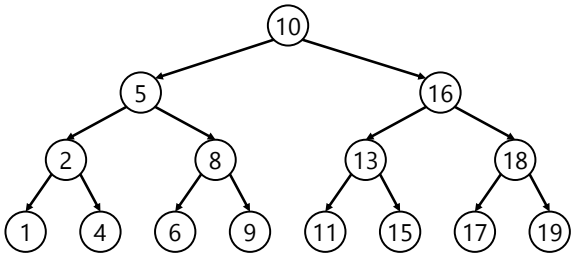
- **binary search** "cuts a BST in half" over and over
  - **binary search only works when the binary tree is sorted (is a binary search tree)**
  - binary search is **O(log n)** if the tree is "balanced" 😊
    - **balanced** means each node's children are similar in height (intuitive understanding of this is sufficient; will discuss more on Fri)

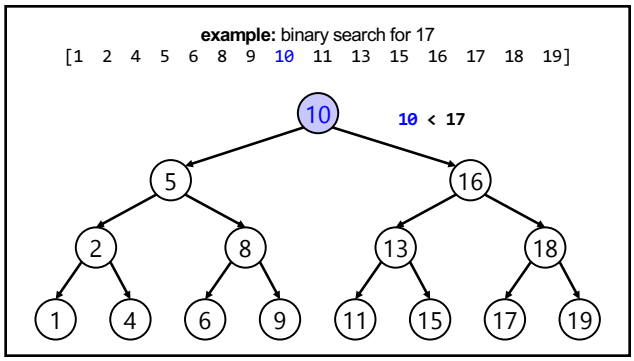
balanced

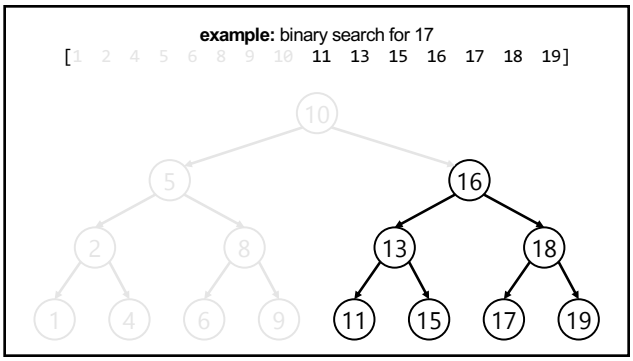
NOT balanced

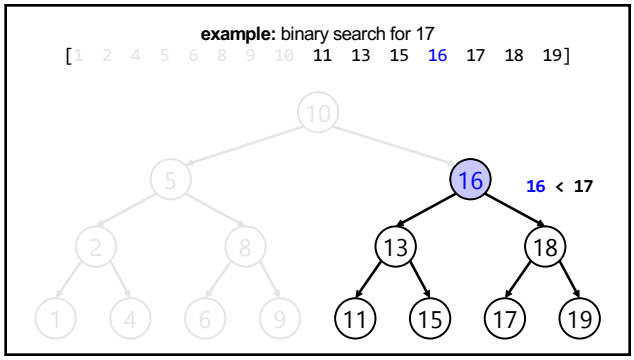
example: binary search for 17

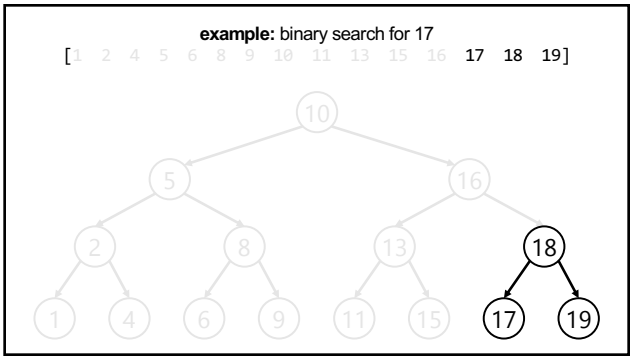
```
[1  2  4  5  6  8  9  10  11  13  15  16  17  18  19]
```

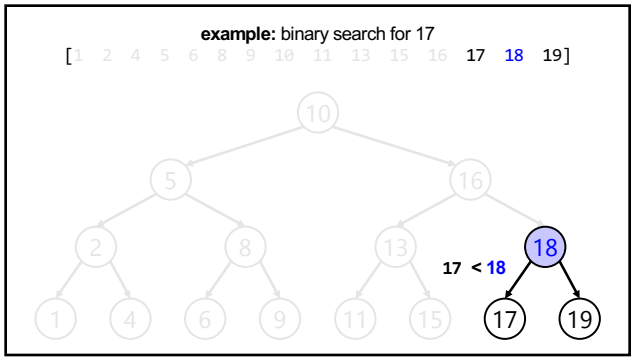


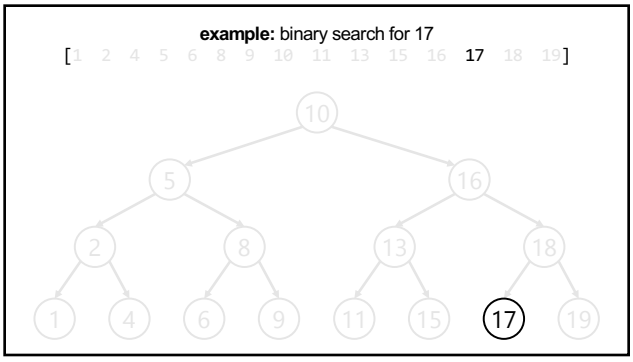

example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]
10 < 17

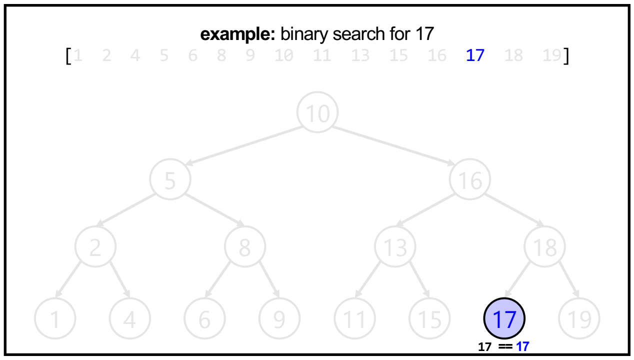

example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

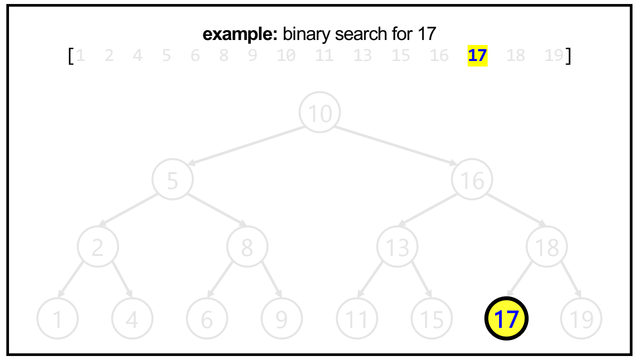

example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]
16 < 17


example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]


example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]
17 < 18

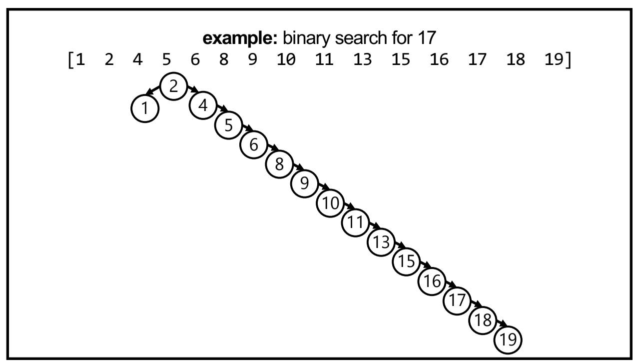

example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

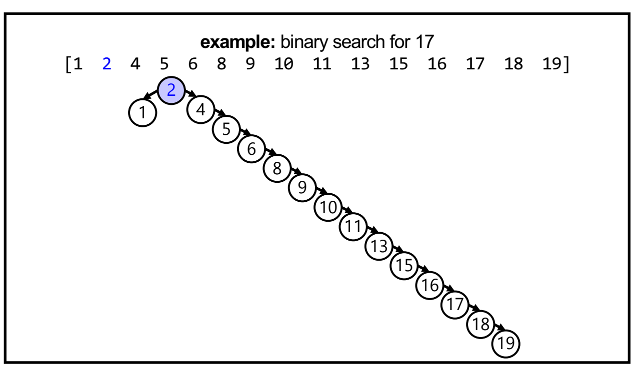


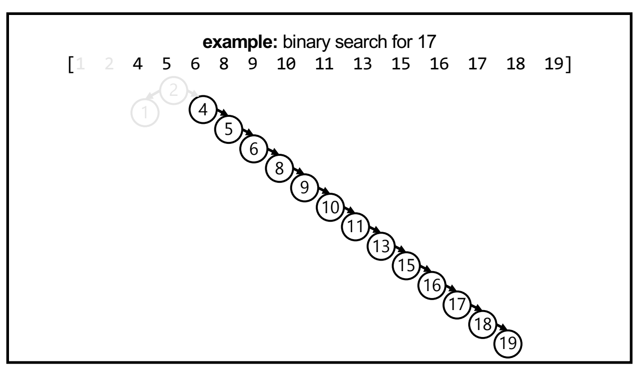


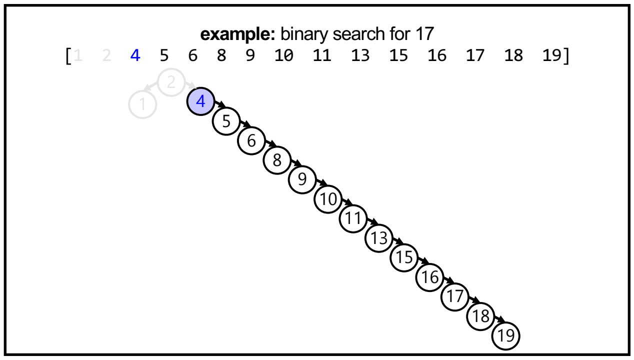
note: a binary search tree is NOT unique

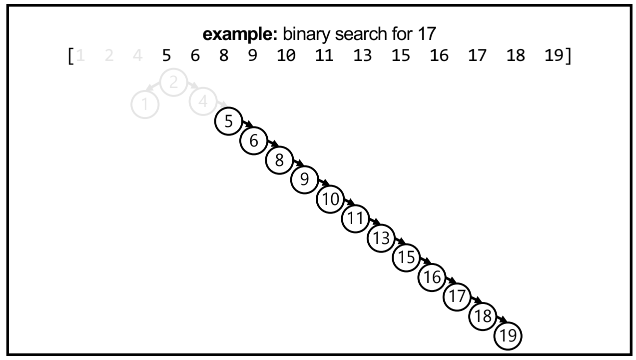
let's look at another BST for the same data!

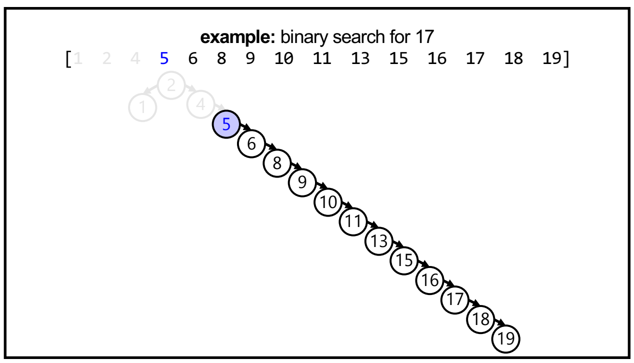


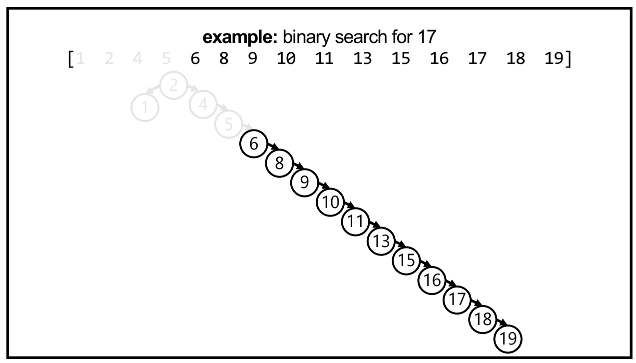


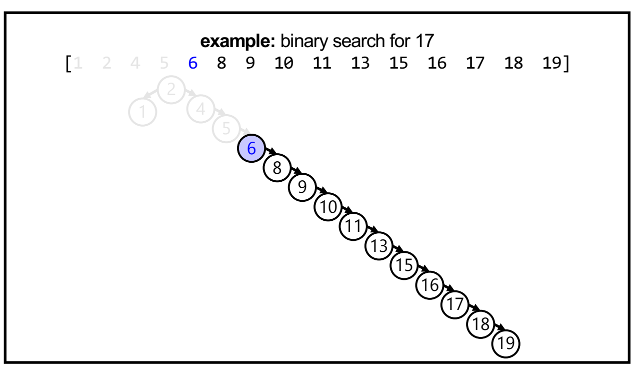


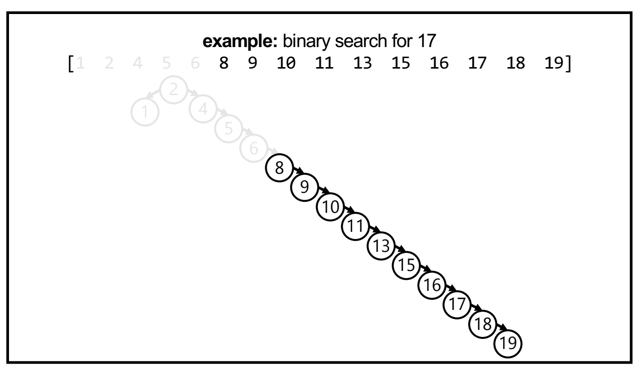

example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]


example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]


example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]


example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]


example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]


example: binary search for 17
[1 2 4 5 6 8 9 10 11 13 15 16 17 18 19]

...

life lesson: it is important that your binary search tree is ✨ balanced ✨

(otherwise your "binary search" degrades into linear search of a linked list 😩)

adding a new node to a binary search tree

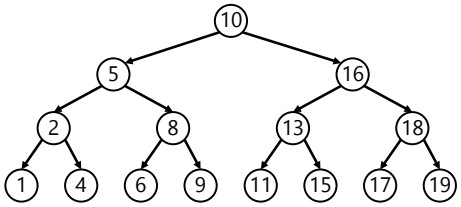
(the naive (simple but bad) method)

for a binary search tree,
add starts out just like **`search`**

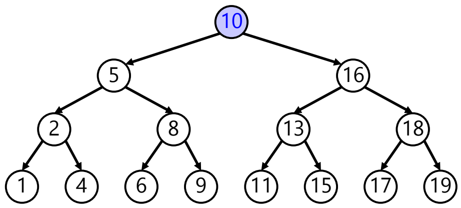
just keep going until you hit a `null` node

put the new node there 🙂 👍

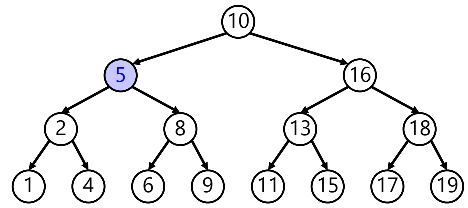
example: adding 7



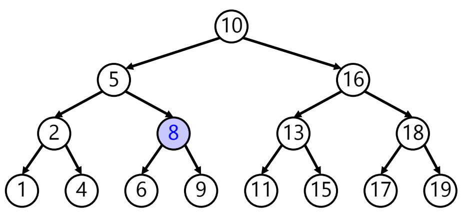
example: adding 7



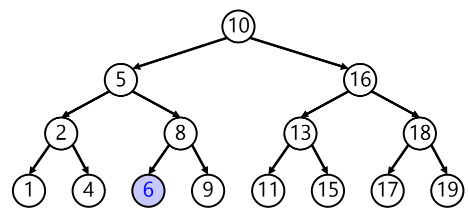
example: adding 7



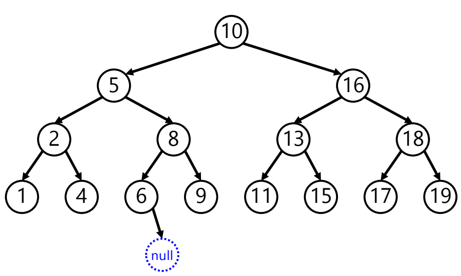
example: adding 7



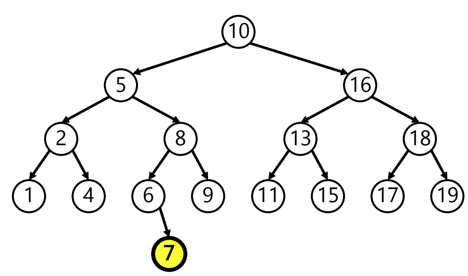
example: adding 7



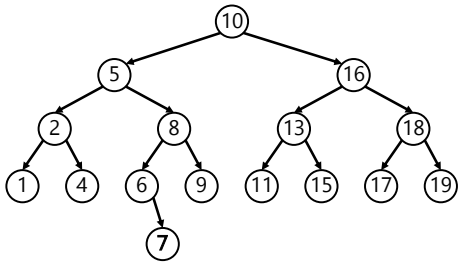
example: adding 7



example: adding 7



example: adding 7



is this approach to adding a new node good?

hint: no.

self-balancing
binary search trees

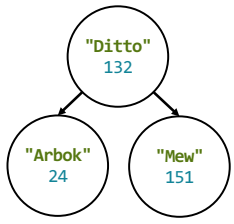
note: hard to implement

uses of binary search trees

tree map (implement a map)

- you can implement the map interface using a BST
  NOTE: this is NOT a hash map!—no hashing is involved!

```
class Node {
    String key; // NOTE: BST is sorted by key
    Integer value;
    Node leftChild;
    Node rightChild;
}

class TreeMap {
    Node root;
    ValueType get(KeyType) { ... }
    void put(ValueType, KeyType) { ... }
}
```

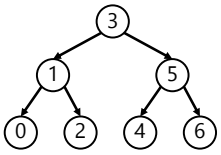


binary tree depth-first traversal orders

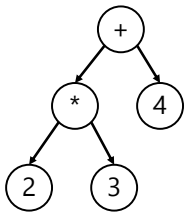
binary tree depth-first traversal orders

depth-first traversal orders (of a binary tree)

- **pre-order** = self, left, right
  - 3, 1, 0, 2, 5, 4, 6
- **in-order** = left, self, right
  - 0, 1, 2, 3, 4, 5, 6
- **post-order** = left, right, self
  - 0, 2, 1, 4, 6, 5, 3
- **reverse pre-order** = self, right, left
  - 3, 5, 6, 4, 1, 2, 0
- **reverse in-order** = right, self, left
  - 6, 5, 4, 3, 2, 1, 0
- **reverse post-order** = right, left, self
  - 6, 4, 5, 2, 0, 1, 3

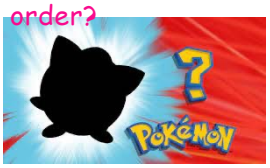


2 * 3 + 4



who's that traversal order?

in-order!




```
void recurse(Node self) {
    // NOTE: rearranging these 3 lines gives you all
    // 3! ("three factorial") = 6 traversal orders
    if (self.leftChild  != null) { recurse(self.leftChild);  }
    System.out.print(self.value + " ");
    if (self.rightChild != null) { recurse(self.rightChild); }
}
```

ANNOUNCEMENTS

today is See You Back In Person on Wednesday Wednesday!

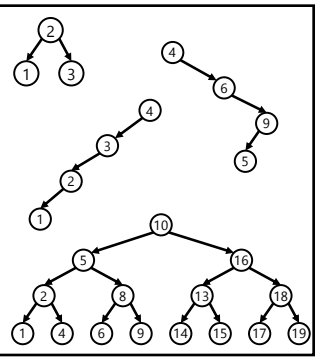
WARMUP

Is It A Binary Search Tree?

in a **binary search tree** (sorted binary tree) every node follows:
"a node's value is **greater than the values of all nodes in its left subtree** and **less than the values of all nodes in its right subtree**"

TODAY

traversal order demo; heaps



ANNOUNCEMENTS

today is See You Back In Person on Wednesday Wednesday!

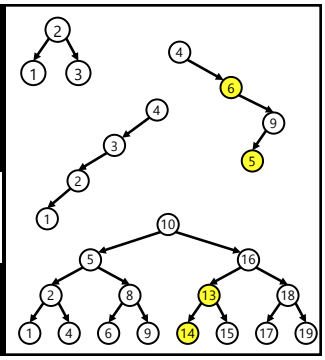
WARMUP

Is It A Binary Search Tree?

in a **binary search tree** (sorted binary tree) every node follows:
"a node's value is **greater than the values of all nodes in its left subtree** and **less than the values of all nodes in its right subtree**"

TODAY

traversal order demo; heaps



record LEC-02

something to ponder:
what will this pseducode do?

```
binarySearchTree = BinarySearchTree()
array = [ 1, 2, 5, 6, 7, 9, 12, 17 ]
for element in array:
    binarySearchTree.add(element)
```

TODO (Jim): Live-code traversal orders
(Feel free to follow along or race.)

heaps

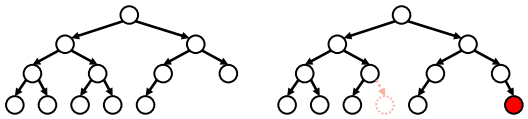
always-complete
max binary heap

always-complete max binary heap

- in this class, when we say "**heap**" or "**max heap**", we mean an "**always-complete max binary heap**"
  - we might occasionally mention a "**min heap**", which means an "**always-complete min binary heap**"

always-complete max binary heap

- in a **complete binary tree**, all levels (depths, rows) are "full of nodes", except for possibly the bottom level, in which all nodes are "as far to to the left as possible"

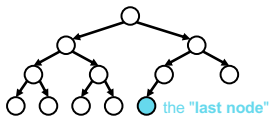


complete
(and balanced)

balanced but **NOT complete**

always-complete max binary heap

- in a **complete binary tree**, all levels (depths) are "full of nodes", except for possibly the bottom level, in which all nodes are "as far to to the left as possible"



complete
(and balanced)

always-complete max binary heap

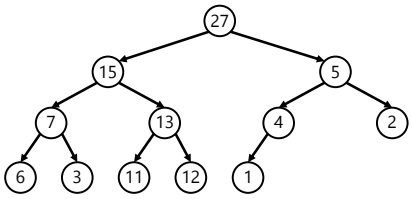
- "**always-complete**" means that every function in the Heap interface (`add(...)` & `remove()`) "preserves the completeness of the heap"
  - the heap **was complete** before calling `add`...
  - ...and the heap **is still complete** after `add` returns

always-complete max binary heap

- a **binary heap** is another special kind of binary tree
  - ☠️ **a binary heap is NOT, in general, a binary search tree**

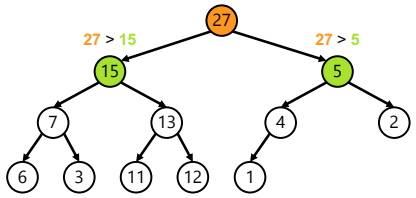
always-complete max binary heap

- in a **max binary heap**, **every node** follows: "a node's value is greater than the value of its left child and the value of its right child"



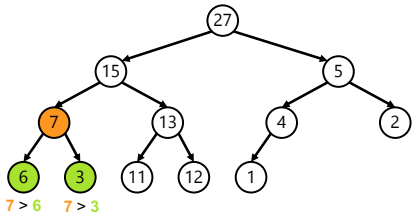
always-complete max binary heap

- in a **max binary heap**, **every node** follows: "a node's value is greater than the value of its left child and the value of its right child"



always-complete max binary heap

- in a **max binary heap**, **every node** follows: "a node's value is greater than the value of its left child and the value of its right child"

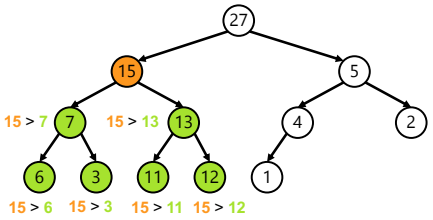


always-complete max binary heap

- in a **max binary heap**, **every node** follows: "a node's value is greater than the value of its left child and the value of its right child"
- 🧠 is the "**max heap property**" above equivalent to...
  - in a **max binary heap**, **every node** follows: "a node's value is greater than the values of all its descendants"
    - yes.
      - **idea:** apply definition recursively
        - node's value is greater than the values of its children...
          ...which are greater than the values of their children...
- 🧠 which node always has the max value of all nodes in the heap?
  - the root

always-complete max binary heap

- in a **max binary heap**, **every node** follows: "a node's value is greater than the values of all its descendants"



heap interface

`add(...)` & `remove()`

heap interface

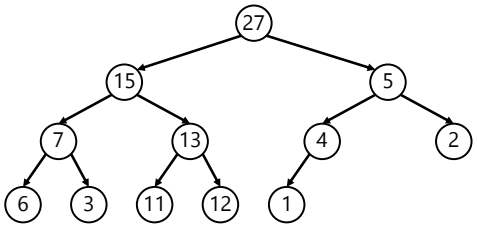
```
// Add this value to the heap.
void add(ValueType value) { ... }

// Remove the max value from the heap, and return it.
ValueType remove() { ... }
```

add(...)

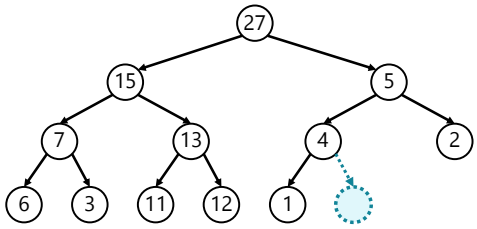
`void add(ValueType value);`

- to **add** a new node with a given value to a max binary heap...
  - add the new node so that the heap is still complete (add into "the next empty slot")
  - while that node violates the max heap property...
    - swap it with its parent
- the node **"swims up"**
  - "sifts up"
  - "heap up"?

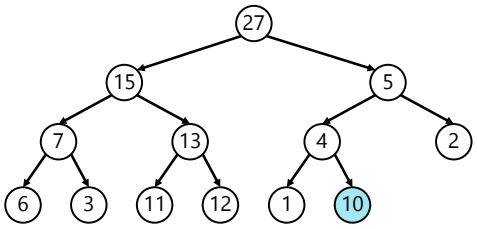
example: adding 10



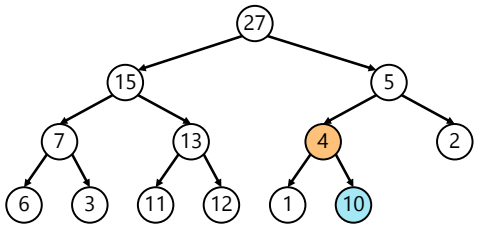
example: adding 10



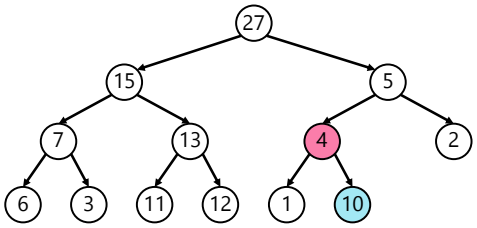
example: adding 10



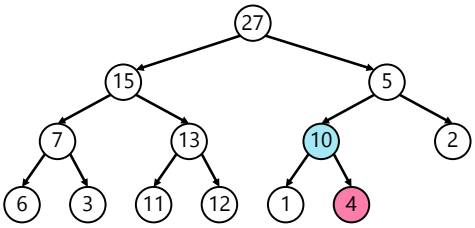
example: adding 10



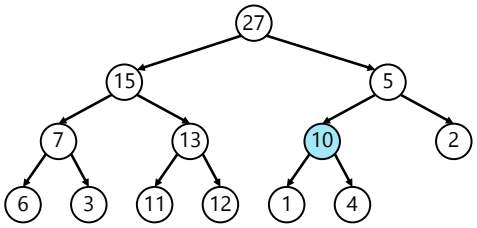
example: adding 10



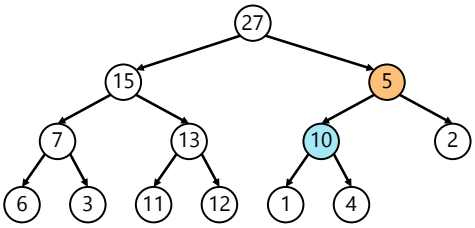
example: adding 10



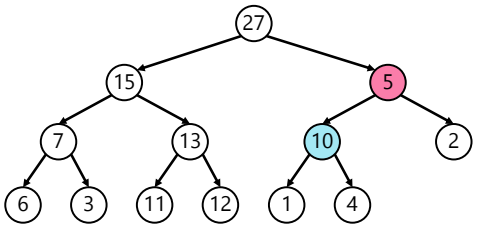
example: adding 10



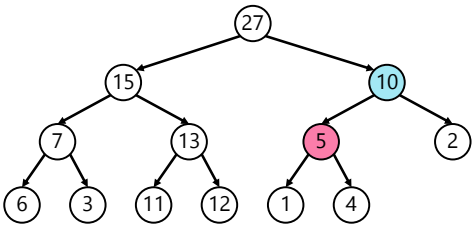
example: adding 10



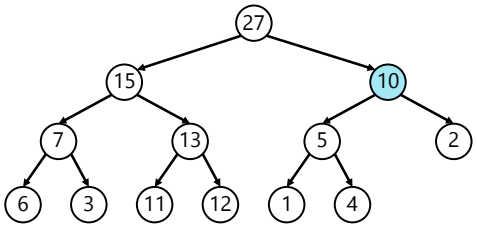
example: adding 10



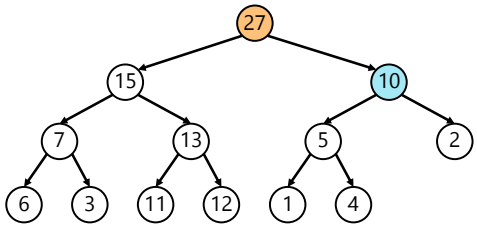
example: adding 10



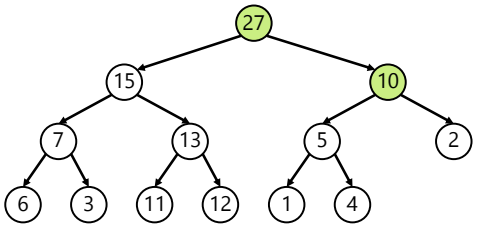
example: adding 10



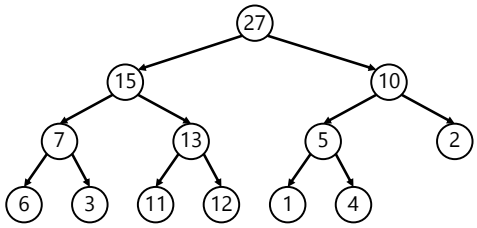
example: adding 10



example: adding 10



example: adding 10



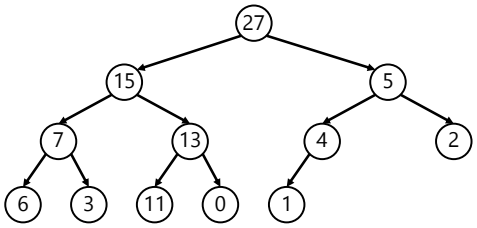
remove()

`ValueType remove();`

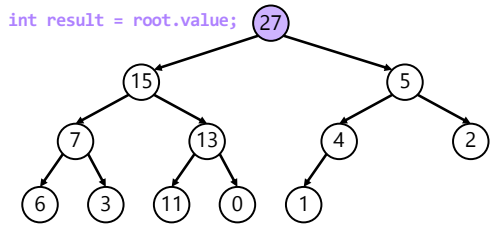
- to **remove** the node with max value (the root) from a max binary heap...
  - save the root's value in a temporary variable called `result`
  - replace the root with the **last node** (rightmost node in the bottom level)
    (the old root is now "dead" and ready to be garbage collected 🚛)
  - while that node violates the max heap property...
    - swap it with its larger child
  - `return result;`

- the node "**sinks down**" ⚓
  - "**sifts down**"
  - "heap down"?

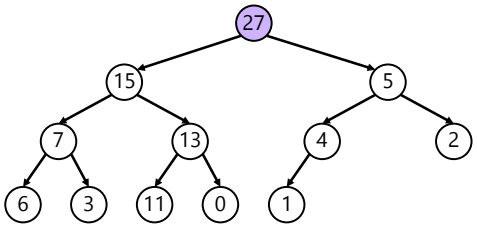
example: removing max node



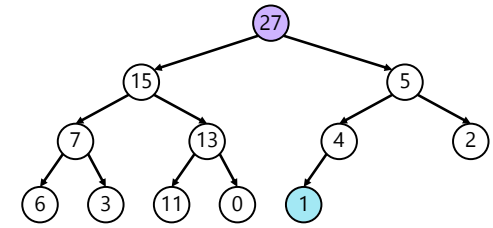
example: removing max node



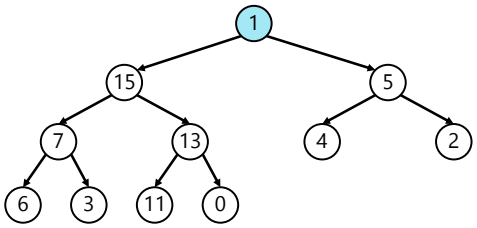
example: removing max node



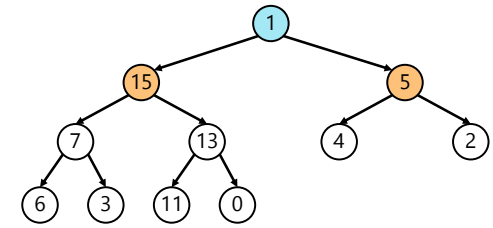
example: removing max node



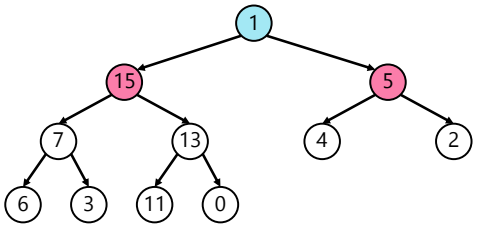
example: removing max node



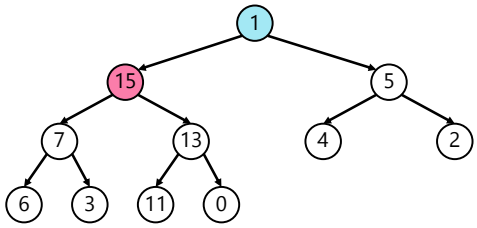
example: removing max node



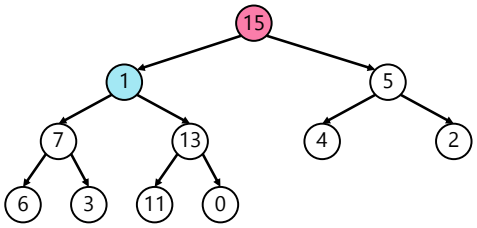
example: removing max node



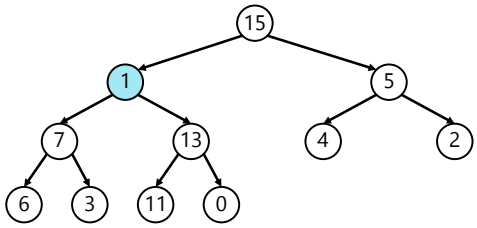
example: removing max node



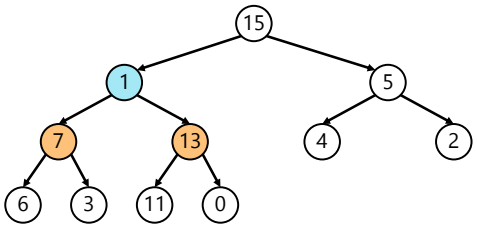
example: removing max node



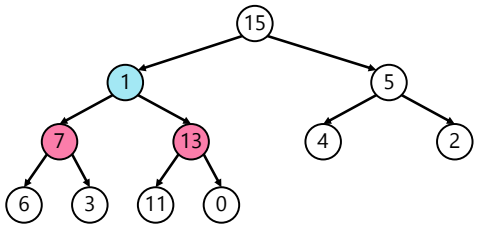
example: removing max node



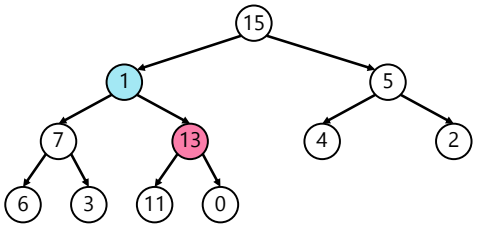
example: removing max node



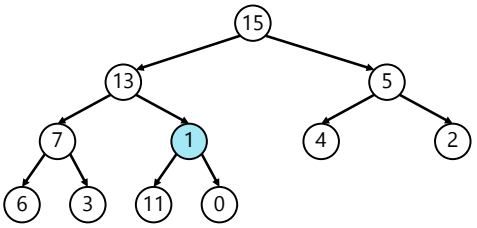
example: removing max node



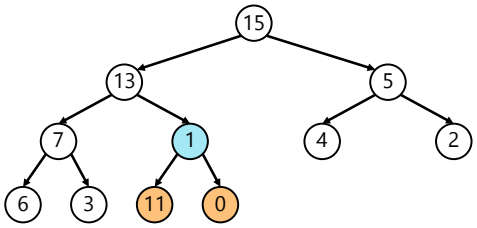
example: removing max node



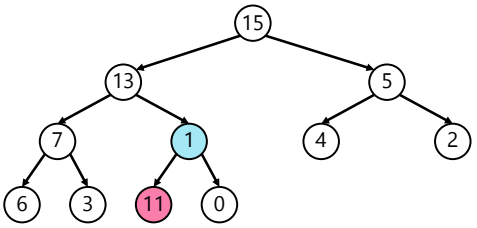
example: removing max node



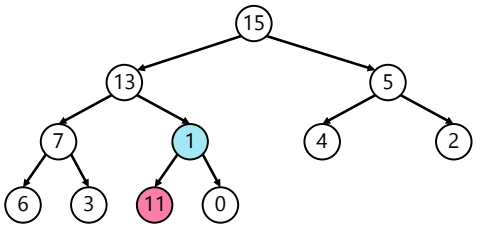
example: removing max node



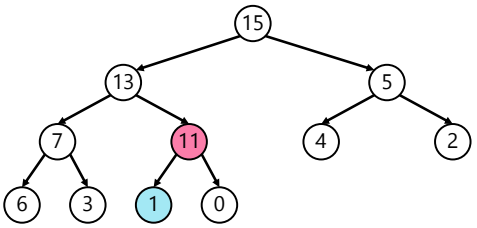
example: removing max node



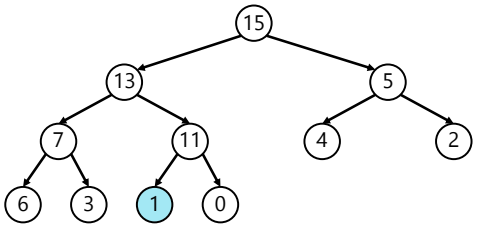
example: removing max node



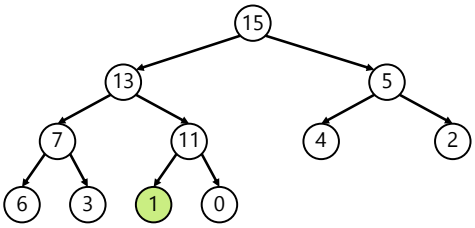
example: removing max node



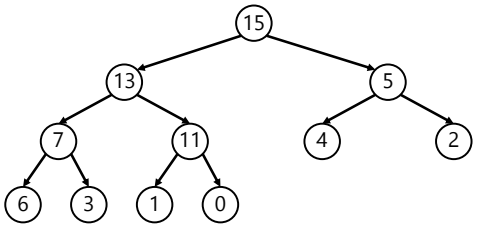
example: removing max node



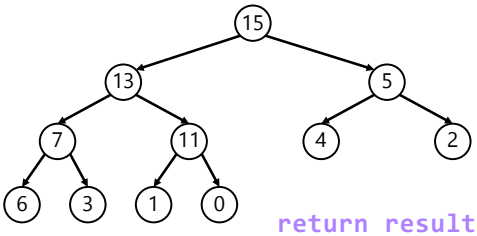
example: removing max node



example: removing max node



example: removing max node



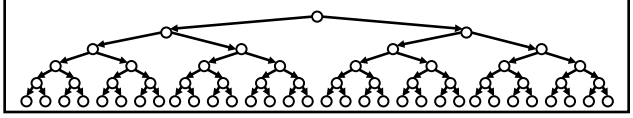

```
return result;
```

ANNOUNCEMENTS today is Fun Friday with DJ Microsoft Excel
also Prof. Katie Keith is Visiting Friday

WARMUP

How **tall** is a **perfect** (totally full) **binary tree** with *n* nodes?
Give your answer in big O. Is it O(1), O(log n), O(n), O(n^2), ...?

TODAY binary search tree and heap wrap-up



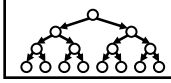
record LEC-02

1 node → height 0

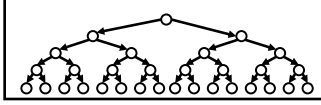
3 nodes → height 1

7 nodes → height 2

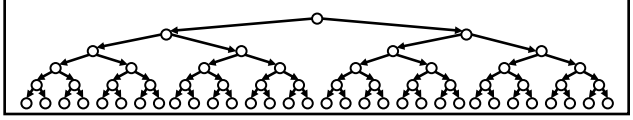
15 nodes → height 3



31 nodes → height 4



63 nodes → height 5



summary

- (1 node, height 0)
- (3 nodes, height 1)
- (7 nodes, height 2)
- (15 nodes, height 3)
- (31 nodes, height 4)
- (63 nodes, height 5)
- ...

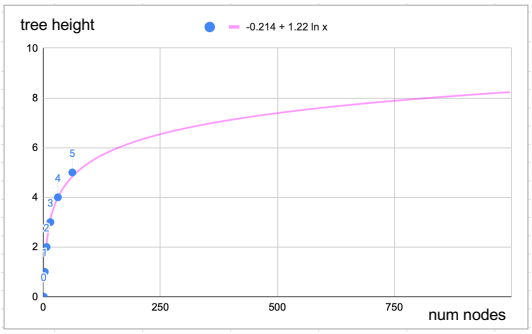
summary

(1, 0)
(3, 1)
(7, 2)
(15, 3)
(31, 4)
(63, 5)
...

summary

1	0
3	1
7	2
15	3
31	4
63	5

TODO (Jim): Let's make a plot.

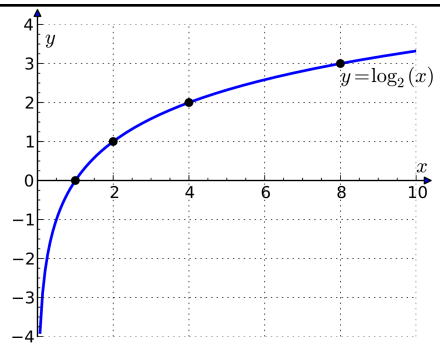


log n

a **balanced binary tree** has O(log n) height

note: the example we just did showed this for a "perfectly balanced" binary tree, but it is also true for just plain ol' balanced binary search trees

what does log look like?



log is the inverse of exponential growth

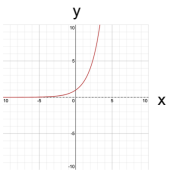
$y = 2^x$

🧠 solve for x.

$y = 2^x$

$x = \log_2 y$

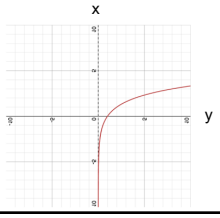
$y = 2^x$



$x = \log_2 y$

$y = 2^x$

$x = \log_2 y$



the change of base formula

implies that $O(\log_2 n) = O(\log_{10} n) = \ldots$

$\log_b n = \log_d n / \log_d b$

$\log_2 n = \log_{10} n / \log_{10} 2$

$\log_2 n = \log_{10} n / \log_{10} 2$

this is a constant.

$\log_2 n = c \log_{10} n$

$O(\log_2 n) = O(c \log_{10} n)$

$O(\log_2 n) = O(\log_{10} n)$

$O(\log_2 n)$ and $O(\log_{10} n)$ are the exact same thing

so you can just say O(log n) and not worry about it 🙄 👍

binary search tree details

self-balancing
binary search trees

life lesson: it is important that your binary search tree is ✨ balanced ✨

(otherwise things starts to look a lot like linear search on a linked list 😩)

self-balancing binary search trees are very cool but painful to implement

heap details

heap application:
priority queue

review: queue

remove();

review: queue

remove();

review: queue

add();

review: queue

add();

review: queue

remove();

review: queue

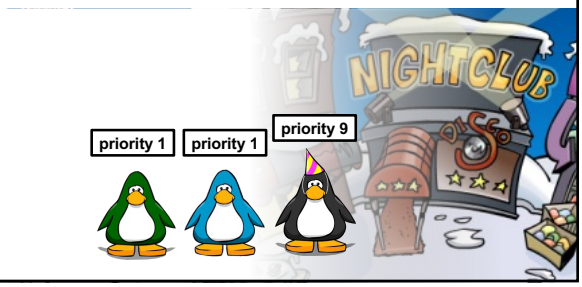
extension: priority queue

extension: priority queue

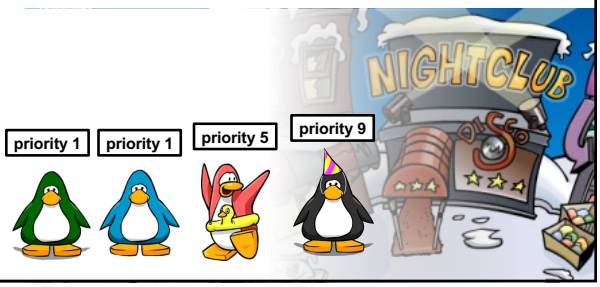
extension: priority queue

extension: priority queue

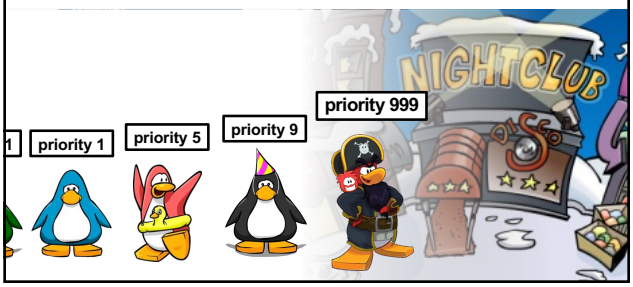
extension: priority queue



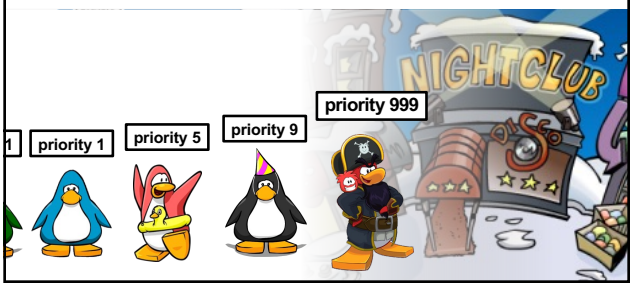
extension: priority queue



extension: priority queue



extension: priority queue



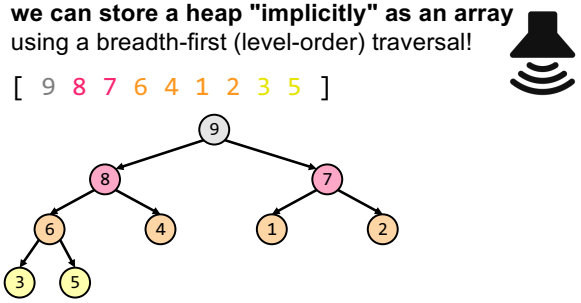
a **heap**'s remove() function removes the node with **maximum value**

a **priority queue**'s remove() function removes the element with **highest priority**

TODO: club penguin meme

heap application:
(implicit) heapsort

because a heap is an always-complete binary tree,
we can store a heap "implicitly" as an array
using a breadth-first (level-order) traversal!



this lets us do **in-place heapsort**!

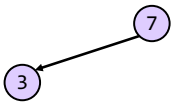
(using only swaps)

1. build a heap by calling add(...) over and over
2. deconstruct the heap by calling remove() over and over

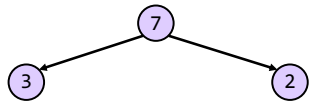
[7 3 2 4 6 1 8 5 9]

[7 3 2 4 6 1 8 5 9]

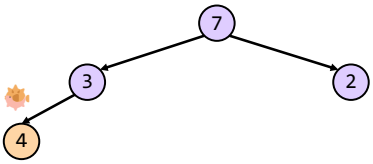
[7 3 2 4 6 1 8 5 9]



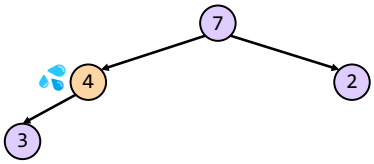
[7 3 2 4 6 1 8 5 9]



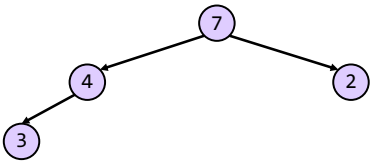
[7 3 2 4 6 1 8 5 9]



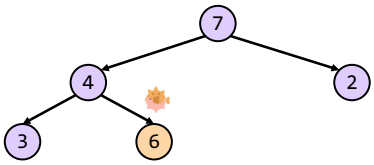
[7 4 2 3 6 1 8 5 9]



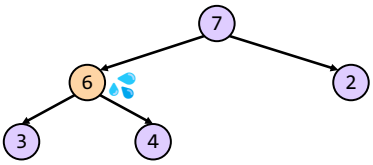
[7 4 2 3 6 1 8 5 9]



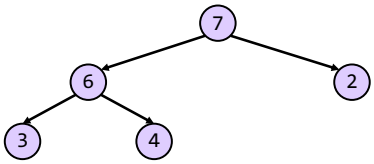
[7 4 2 3 6 1 8 5 9]



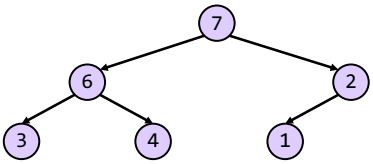
[7 6 2 3 4 1 8 5 9]



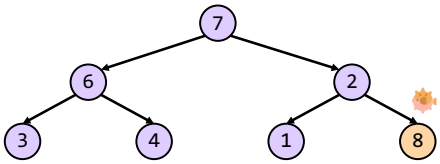
[7 6 2 3 4 1 8 5 9]



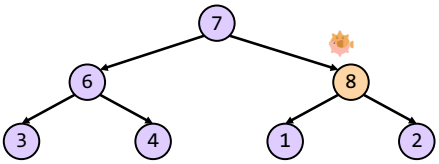
[7 6 2 3 4 1 8 5 9]



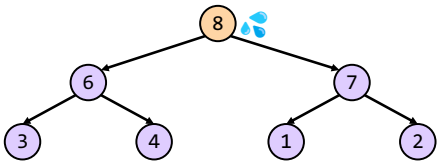
[7 6 2 3 4 1 8 5 9]



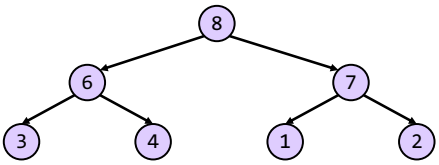
[7 6 8 3 4 1 2 5 9]



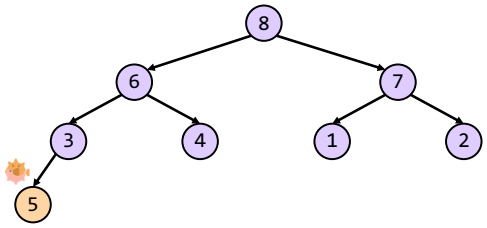
[8 6 7 3 4 1 2 5 9]



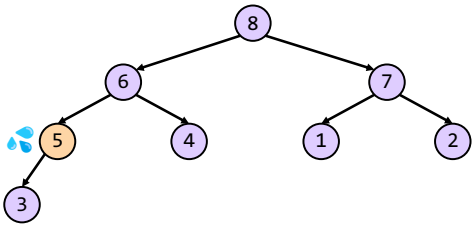
[8 6 7 3 4 1 2 5 9]



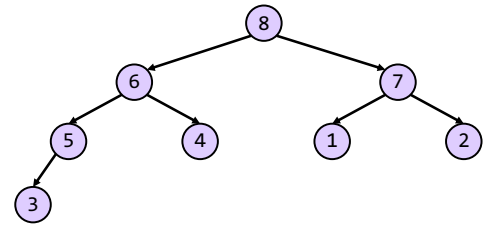
[8 6 7 3 4 1 2 5 9]



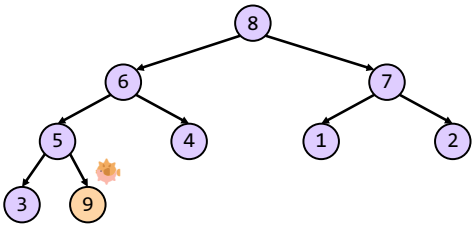
[8 6 7 5 4 1 2 3 9]



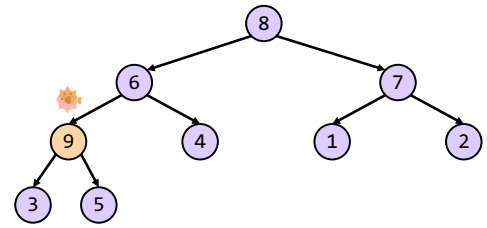
[8 6 7 5 4 1 2 3 9]



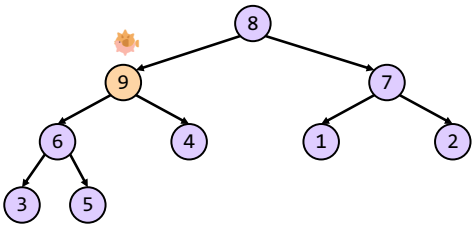
[8 6 7 5 4 1 2 3 9]



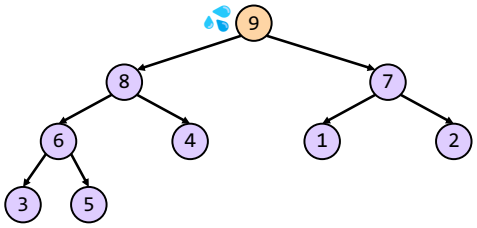
[8 6 7 9 4 1 2 3 5]



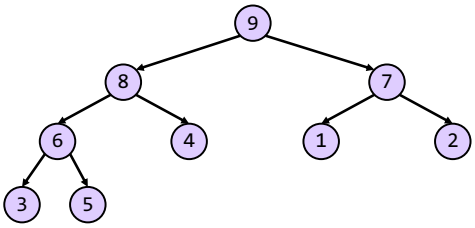
[8 9 7 6 4 1 2 3 5]



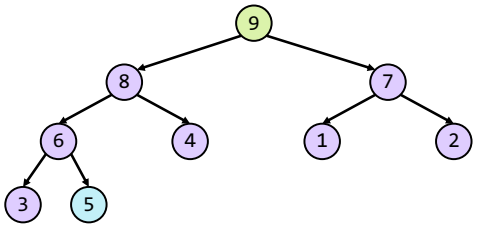
[9 8 7 6 4 1 2 3 5]



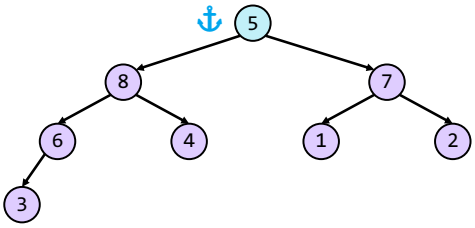
[9 8 7 6 4 1 2 3 5]



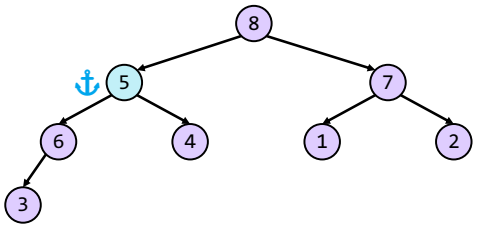
[9 8 7 6 4 1 2 3 5]



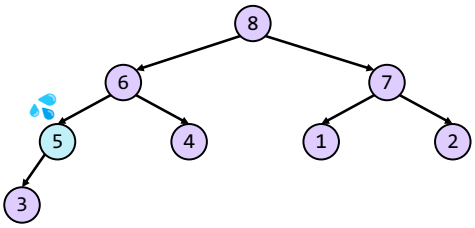
[5 8 7 6 4 1 2 3 9]



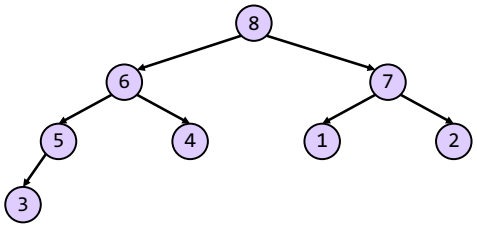
[8 5 7 6 4 1 2 3 9]



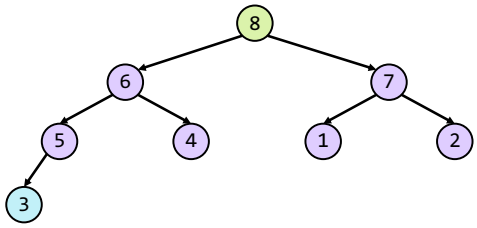
[8 6 7 5 4 1 2 3 9]



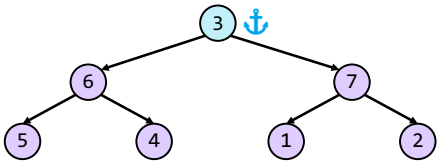
[8 6 7 5 4 1 2 3 9]



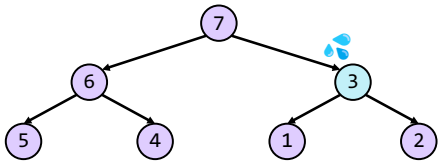
[8 6 7 5 4 1 2 3 9]



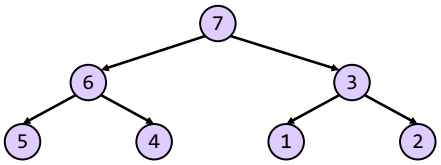
[3 6 7 5 4 1 2 8 9]



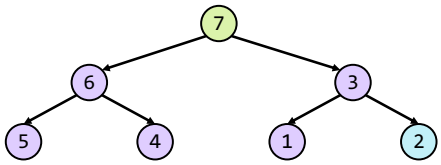
[7 6 3 5 4 1 2 8 9]



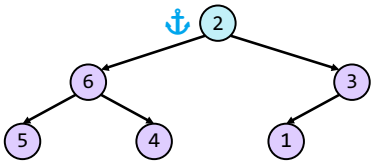
[7 6 3 5 4 1 2 8 9]



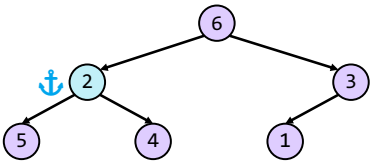
[7 6 3 5 4 1 2 8 9]



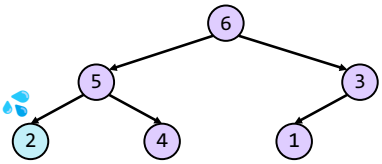
[2 6 3 5 4 1 7 8 9]



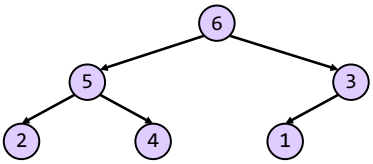
[6 2 3 5 4 1 7 8 9]



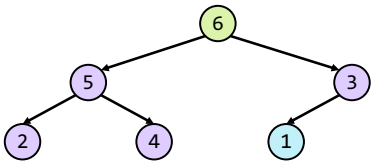
[6 5 3 2 4 1 7 8 9]



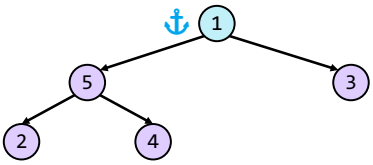
[6 5 3 2 4 1 7 8 9]



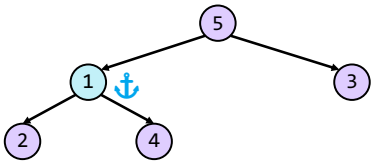
[6 5 3 2 4 1 7 8 9]



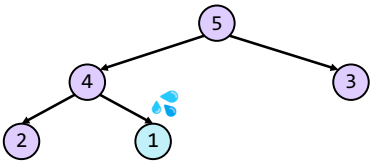
[1 5 3 2 4 6 7 8 9]



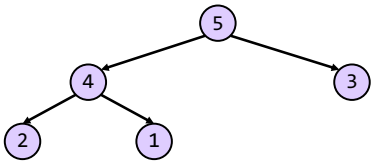
[5 1 3 2 4 6 7 8 9]



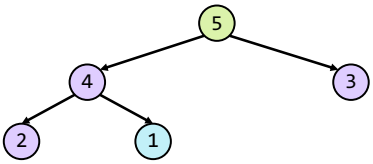
[5 4 3 2 1 6 7 8 9]



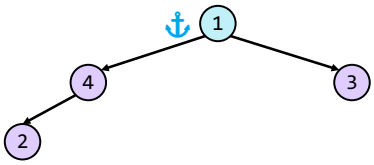
[5 4 3 2 1 6 7 8 9]



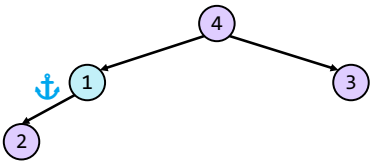
[5 4 3 2 1 6 7 8 9]



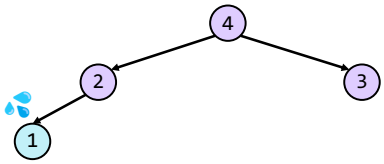
[1 4 3 2 5 6 7 8 9]



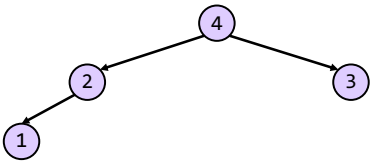
[4 1 3 2 5 6 7 8 9]



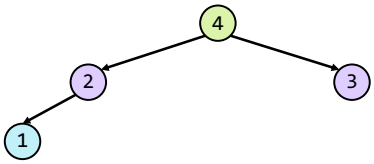
[4 2 3 1 5 6 7 8 9]



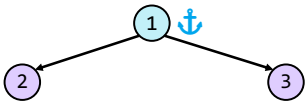
[4 2 3 1 5 6 7 8 9]



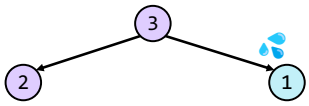
[4 2 3 1 5 6 7 8 9]



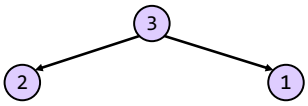
[1 2 3 4 5 6 7 8 9]



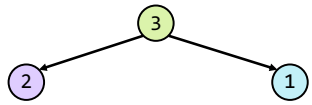
[3 2 1 4 5 6 7 8 9]



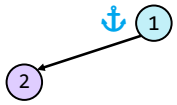
[3 2 1 4 5 6 7 8 9]



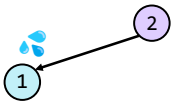
[3 2 1 4 5 6 7 8 9]



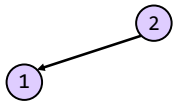
[1 2 3 4 5 6 7 8 9]



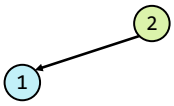
[2 1 3 4 5 6 7 8 9]



[2 1 3 4 5 6 7 8 9]



[2 1 3 4 5 6 7 8 9]



[1 2 3 4 5 6 7 8 9]

[1 2 3 4 5 6 7 8 9]

1

[1 2 3 4 5 6 7 8 9]

1

[1 2 3 4 5 6 7 8 9]

gamedev update

(switch to other laptop)

avl tree
red black tree
anchor

n vs. log n
priority queue (club analogy)
log(n) time
implicit heap

robots/games

space complexity

john's robots tree algorithm

ANNOUNCEMENTS
today is **No Laptop Monday**!

Week08a

WARMUP
"a chain is only as strong as its weakest link"
- what does this expression mean?
- what is a chain?
- what is a link?
- is this true for real-world metal chains? why or why not?

TODAY linked lists

linked lists

record lecture

(p)review: list interface

list interface

```
// Get the element with this index.
ElementType get(int index);

// Add (append) an element to the back of the list.
void add();

// Add (insert) an element into the list so it has this index
void add(int index, ElementType element);

// Remove (delete) the element in the list at this index.
void remove(int index);

// Get the number of elements currently in the list.
int size();
```

list interface (cont.)

```
// NOTE: Many other functions could be included
//       in this interface.
  void sort(); // Sort the list.
  void reverse(); // Reverse the list.

  List<ElementType> sorted(); // Get sorted copy of the list.
  List<ElementType> reversed(); // Get reversed copy of list.

  // Get index of first element with this value.
  int find(ElementType element);

  ...
```

a few weeks ago,
we implemented the list interface
using an array

the *array list*

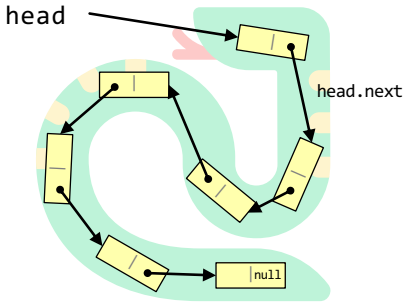
this week, we will implement
the list interface using nodes with
"links" (references) to other nodes

this will be called a **linked list**

note

today we will be discussing the
simplest possible linked list

(LinkedList literally just has a
reference to Node head.)



some other implementations are possible.
some will be faster than this one.

for linked lists, do NOT memorize
big O runtimes out of context

why are we doing this?

A: it will be cool to see two very different implementations of the same interface ☃️

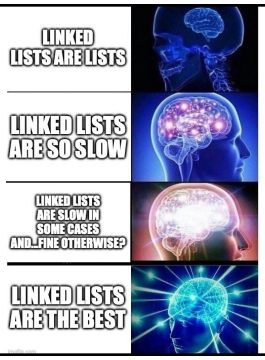
B: linked lists will prepare us for trees and graphs 🌳

C: linked lists are incredibly FUNdaMENTAL 🥳

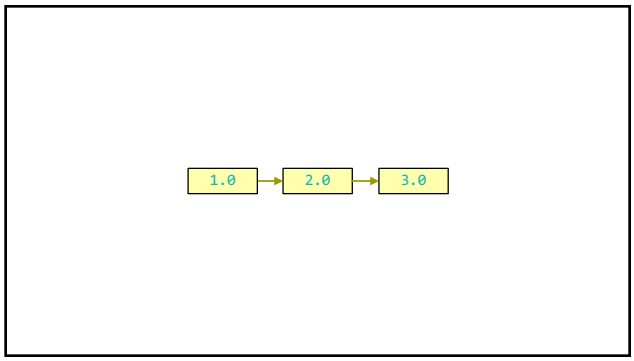
(for us, as fundamental as arrays)

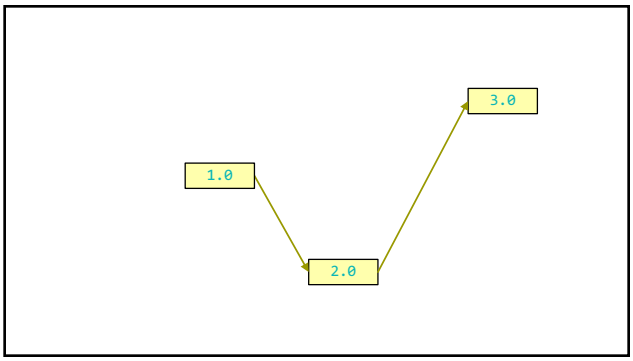
D: linked lists are actually big O better (than array lists) in very specific cases

E: linked lists are actually really, really important
(especially in the C programming language)



linked list





linked list

```
class LinkedList {
  Node head;
}
```

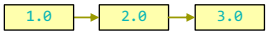
```
class Node {
  Value value;
  Node next;

  Node(Value value) {
    this.value = value;
  }
}
```

```
LinkedList list = new LinkedList();
```

```
list.head = new Node(1.0);
```


```
list.head.next = new Node(2.0);
```


```
list.head.next.next = new Node(3.0);
```



linked list

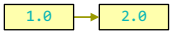
```
class LinkedList {
  Node head;

  void add(Value value) {
    ...
  }
}
```

```
class Node {
  Value value;
  Node next;

  Node(Value value) {
```

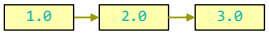
```
LinkedList list = new LinkedList();
```

```
list.add(1.0);
```


```
list.add(2.0);
```

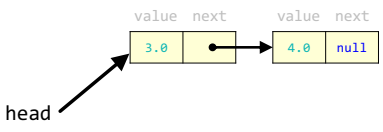


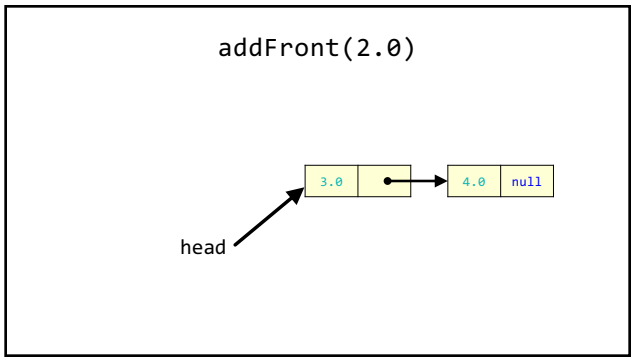

```
list.add(3.0);
```

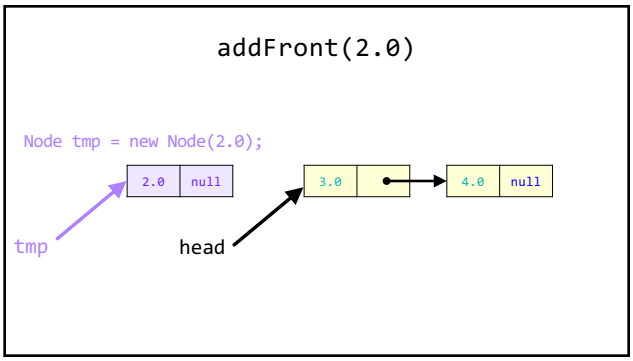


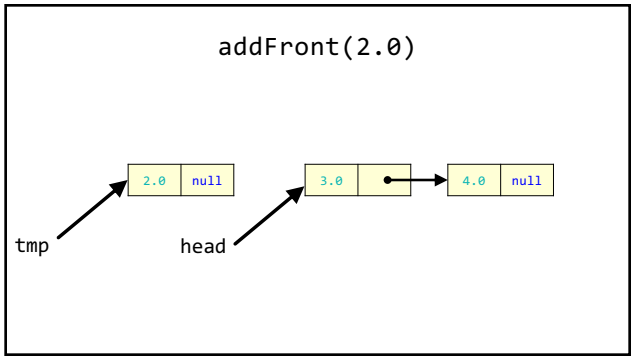
example: `addFront(value)`

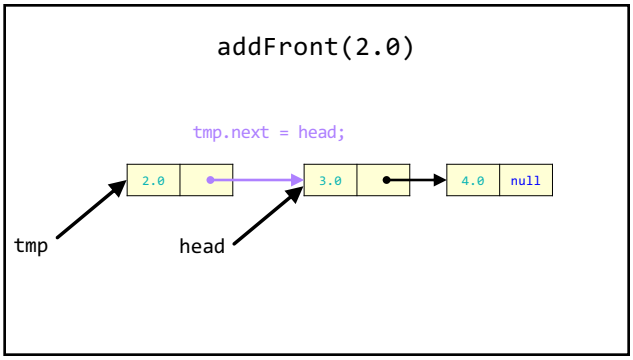
`addFront(2.0)`

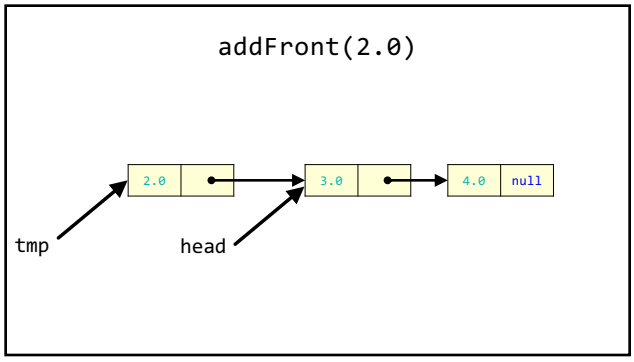


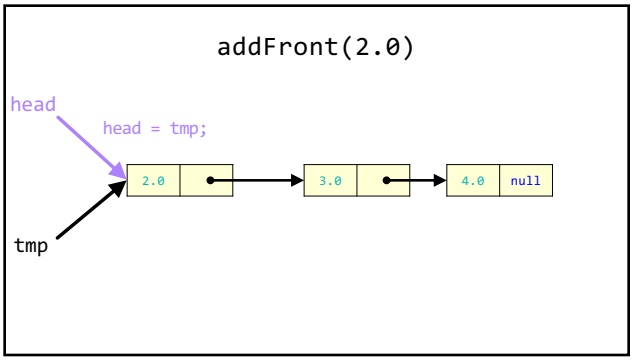

addFront(2.0)
3.0
4.0
null
head


addFront(2.0)
Node tmp = new Node(2.0);
2.0
null
3.0
4.0
null
tmp
head

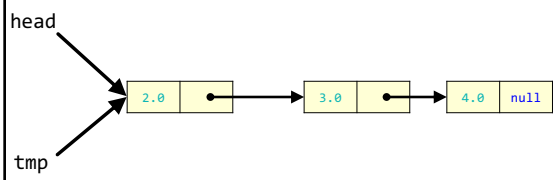

addFront(2.0)
2.0
null
3.0
4.0
null
tmp
head


addFront(2.0)
tmp.next = head;
2.0
3.0
4.0
null
tmp
head

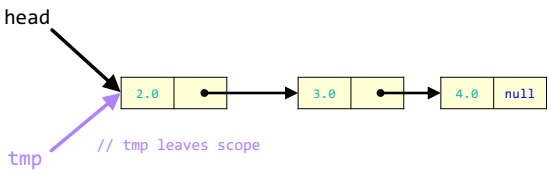

addFront(2.0)
2.0
3.0
4.0
null
tmp
head


addFront(2.0)
head
head = tmp;
2.0
3.0
4.0
null
tmp

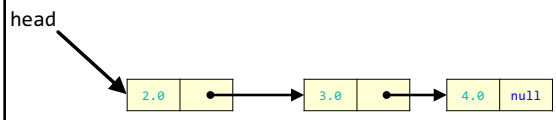
addFront(2.0)



addFront(2.0)

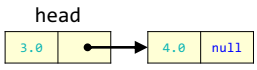


addFront(2.0)



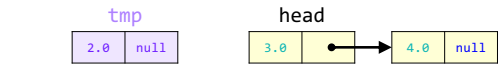
same thing but with labels instead of arrows for `head` and `tmp`

addFront(2.0)

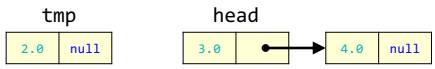


addFront(2.0)

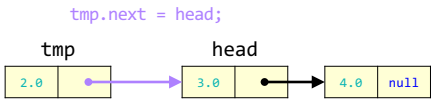
```
Node tmp = new Node(2.0);
```



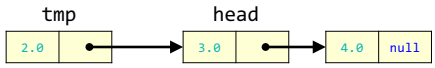
addFront(2.0)



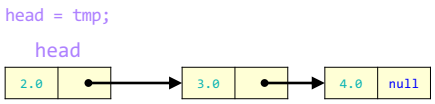
addFront(2.0)



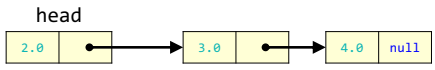
addFront(2.0)

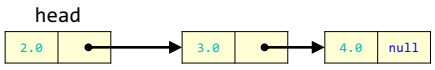


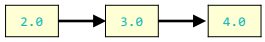
addFront(2.0)



addFront(2.0)



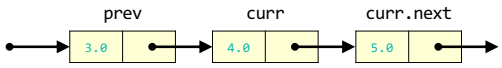




(even) more abstract diagram

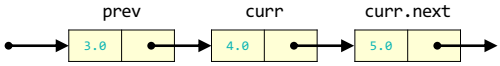
example: `remove`

remove

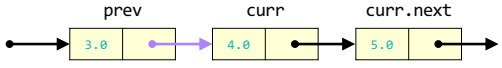



```
Node prev = null; // the previous node
Node curr = head; // the current node
while (...) {
    ...
    prev = curr;
    curr = curr.next;
}
```

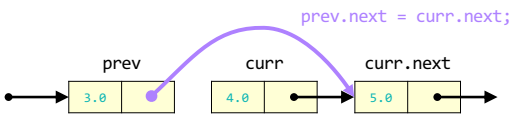
remove



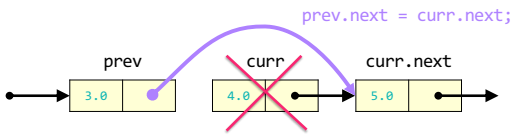
remove



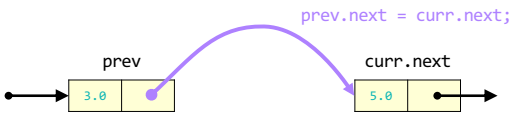
remove



remove

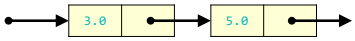


remove



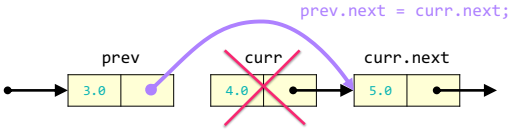
remove

remove



where did the middle node go?

- way back when...we created it using `new`
- then we just like..."stopped referring to it"
- is it gone now? 😳



🚚 it has been garbage collected

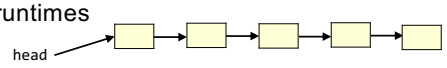
[board discussion of
"no directed path from stack to the node"]

big O runtimes

what is the big O runtime of size()?
[pointing activity]

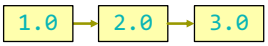
O(n) 😥

singly-linked list
worst case runtimes

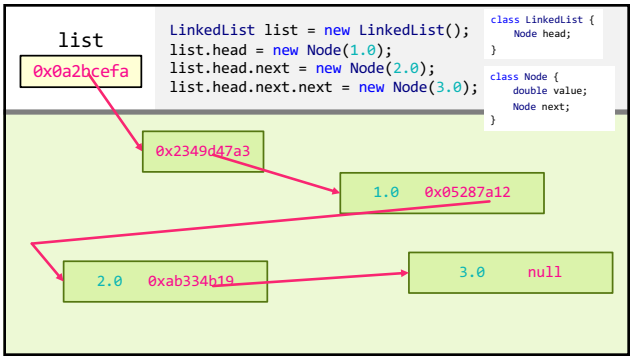


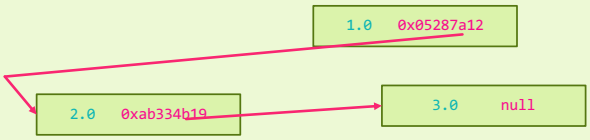
- list.add(index, value)
  - // O(n)
- list.removeByIndex(index)
  - // O(n)
- list.size()
  - // O(n)
- list.addFront() // list.add(0, value)
  - // O(1)
- list.removeFront() // list.removeByIndex(0)
  - // O(1)
- list.addBack()
  - // O(n)
- list.removeBack()
  - // O(n)

beyond big O runtime



what does this actually look like in memory?





what does this *mean*?

cons? 😟

pros? 😳
(how is this very different than an array list?)

https://x.com/_kzr/status/1672497446705037312

ANNOUNCEMENTS

friday's colloquium will be cool!

my friend Pascal makes ✨**underwater robotic spheres**✨!

midterms will be returned Fri

Week08b

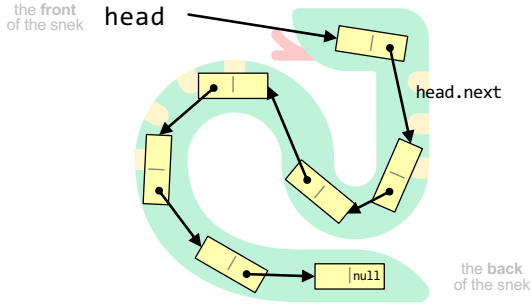
WARMUP

review: for a linked list, `size()` is O(n) runtime

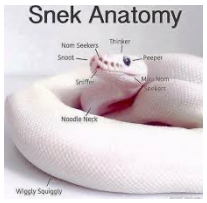
- how could you make `size()` (or perhaps... *`size`*) O(1)?

-- is this at all...*spooky*? 👻

TODAY `size()` and riffs on linked lists



linked lissssssst

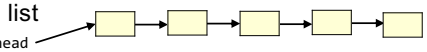


[record lecture]

LinkedList review

runtimes

worst case singly-linked list runtimes

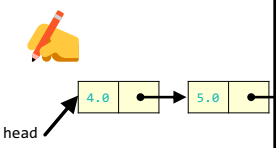


- `list.add(index, value)`
  - `// O(n)`
- `list.removeByIndex(index)`
  - `// O(n)`
- `list.size()`
  - `// O(n)`
- `list.addFront()`
  - `// O(1)`
- `list.removeFront()`
  - `// O(1)`
- `list.addBack()`
  - `// O(n)`
- `list.removeBack()`
  - `// O(n)`

additional warmup: prepending a list

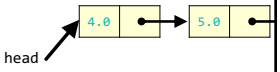
example

```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```



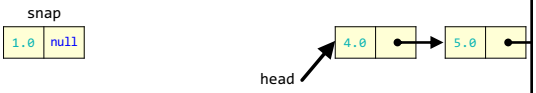
example

```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```



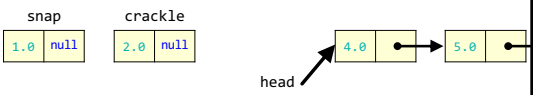
example

```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```



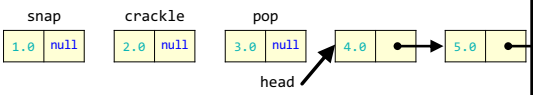
example

```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```



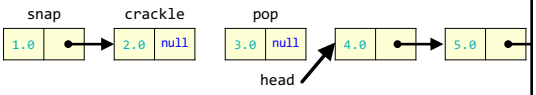
example

```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```



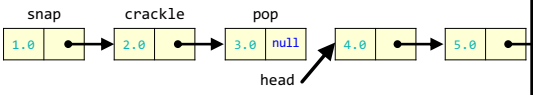
example

```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```



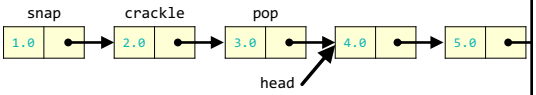
example

```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```



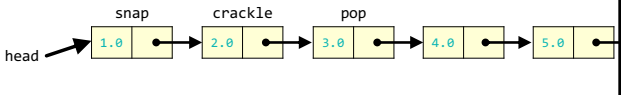
example

```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```



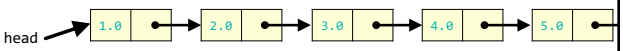
example

```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```



example

```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```




```
Node snap = new Node(1.0);
Node crackle = new Node(2.0);
Node pop = new Node(3.0);
snap.next = crackle;
crackle.next = pop;
pop.next = head;
head = snap;
```

vs.

```
addFront(3.0);
addFront(2.0);
addFront(1.0);
```

🧠 was that faster than calling `addFront()` n times?

🧠 same big O or different big O?

(if we were to call it many times)

🧠 was that faster than calling `addFront()` n times?

yes

(only updated head once)

🧠 same big O or different big O?

same

(still have to "hook up" O(n) references)

[implement LinkedList]

[implement LinkedList]

`size()`

[implement `size()` poorly]

[implement `size()` poorly]

joyful implementation of size

```
static class LinkedList {
    Node head;

    int size() {
        int result = 0;
        Node curr = head;
        while (curr != null) {
            ++result;
            curr = curr.next;
        }
        return result;
    }
}
```

`size()`

- what is the big O runtime of this method?
  - O(n) 😞
- this seems like a pretty steep cost to pay just to know the list's size... what would be a more efficient approach?
  - store size as an instance variable
    - update it every time you change the number of elements in the list (inside of `add`, `remove`, etc.)
  - what is the runtime of this approach?
    - O(1) 😀

while way more efficient, this approach is perhaps a bit spooky 👻

multiple functions are now also responsible for carefully modifying an instance variable
(mess up, and any code that depends on size will be very weirdly broken)

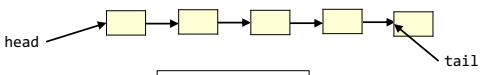
note: the A homework doesn't use size at all

but if you *were* going to implement/use the list's size...
i would start with size() as a function,
get everything working perfectly,
and only then carefully turn it into an instance variable
😳👍

(it's this thing again)

tail reference

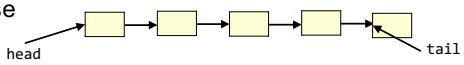
singly-linked list with reference to tail




```
class LinkedList {
    Node head;
    Node tail;
}
```

```
class Node {
    Value value;
    Node next;
}
```

singly-linked list with reference to tail worst case runtimes



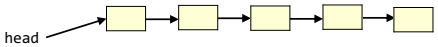
- `list.add(index, value)`
  - `// O(n)`
- `list.removeByIndex(index)`
  - `// O(n)`
- `list.size()`
  - `// O(n)`

- `list.addFront()`
  - `// O(1)`
- `list.removeFront()`
  - `// O(1)`
- `list.addBack()`
  - **`// O(1)`** 😜
- `list.removeBack()`
  - `// O(n)` 😡

doubly-linked list

singly- vs. doubly-linked

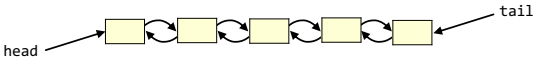
singly-linked list




```
class LinkedList {
    Node head;
}
```

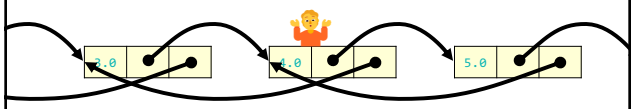
```
class Node {
    Value value;
    Node next;
}
```

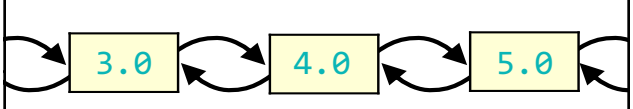
doubly-linked list




```
class LinkedList2 {
    Node2 head;
    Node2 tail;
}
```

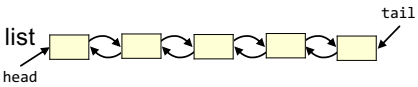
```
class Node2 {
    Value value;
    Node2 next;
    Node2 prev;
}
```





runtimes

worst case doubly-linked list runtimes



- `list.add(index, value)`
  - `// O(n)`
- `list.removeByIndex(index)`
  - `// O(n)`
- `list.size()`
  - `// O(n)`
- `list.addFront()`
  - `// O(1)`
- `list.removeFront()`
  - `// O(1)`
- `list.addBack()`
  - **`// O(1)`** 😃
- `list.removeBack()`
  - **`// O(1)`** 😃

a doubly-linked list is a great way to implement a **deque** (**double-ended queue**)

- `O(1) addFront()`
  `O(1) removeFront()`
  `O(1) addBack()`
  `O(1) removeBack()`

- 🧠 could you pull this off with an **array list**?
  - no.
    - addFront() is O(n)

- 🧠 could you pull this off with an array?
  - sort of!—the **array deque** (amortized O(1) add)

warmup: have you pre-registered?
what classes are you taking?

Week08c

underwater robotic spheres!

colloquium today in Wege (TCL123)
@ 2:30pm

pre-reg

237

```c
int main(void)
{
    // List of size 0
    node *list = NULL;

    // Add number to list
    node *n = malloc(sizeof(node));
    if (n == NULL)
    {
        return 1;
    }
    n->number = 1;
    n->next = NULL;
    list = n;

    // Add number to list
    n = malloc(sizeof(node));
    if (n == NULL)
    {
        return 1;
    }
    n->number = 2;
    n->next = NULL;
    list->next = n;
```

"the stack"

local variable primitives & **references** to Objects

undefined *by default (will NOT compile if used)*

"the heap"

the actual **Objects** (arrays and String's count as Objects)

cleared to 0 or null by default

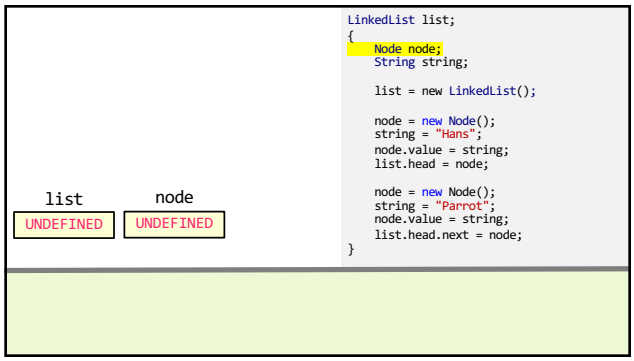
```
LinkedList list =
    new LinkedList("Hans -> Parrot");
```

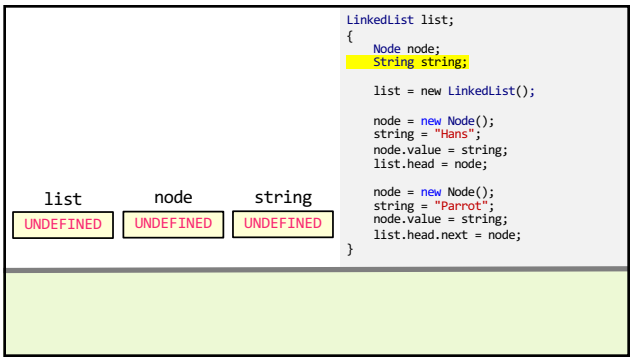
```
LinkedList list;
{
    Node node;
    String string;

    list = new LinkedList();

    node = new Node();
    string = "Hans";
    node.value = string;
    list.head = node;

    node = new Node();
    string = "Parrot";
    node.value = string;
    list.head.next = node;
}
```

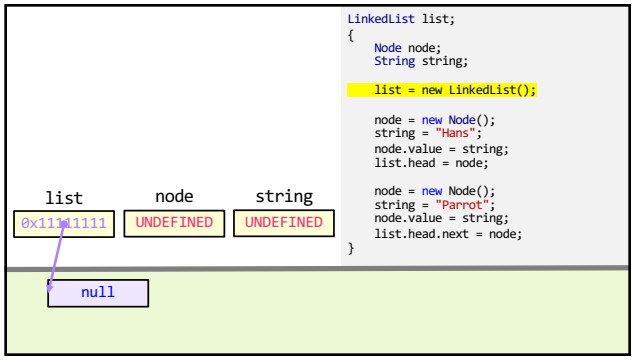
```
LinkedList list;
{
    Node node;
    String string;

    list = new LinkedList();

    node = new Node();
    string = "Hans";
    node.value = string;
    list.head = node;

    node = new Node();
    string = "Parrot";
    node.value = string;
    list.head.next = node;
}
```

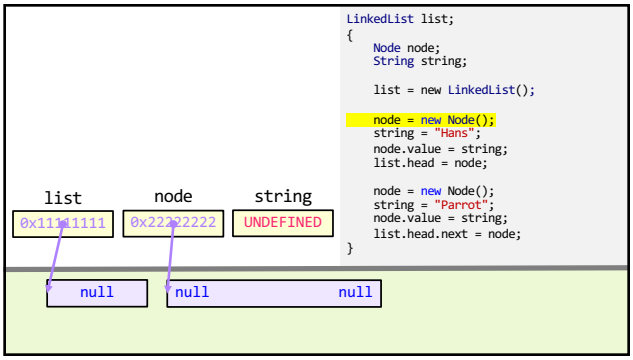
list

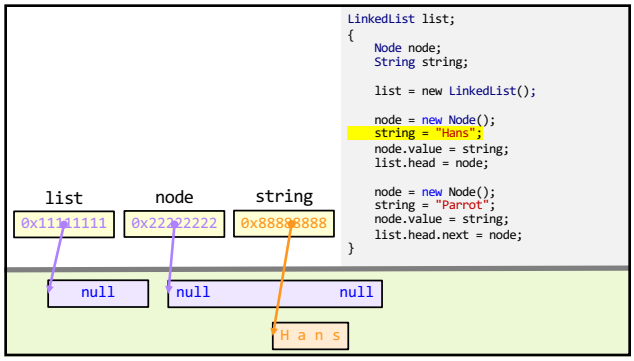
UNDEFINED

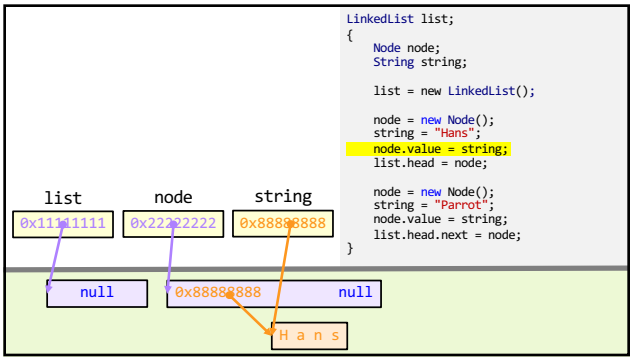

list
node
UNDEFINED
UNDEFINED
LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}

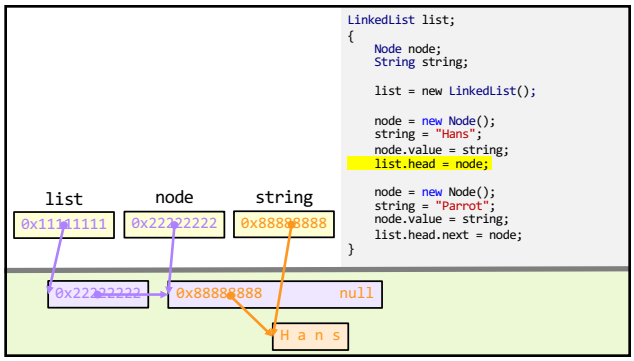

list
node
string
UNDEFINED
UNDEFINED
UNDEFINED
LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}

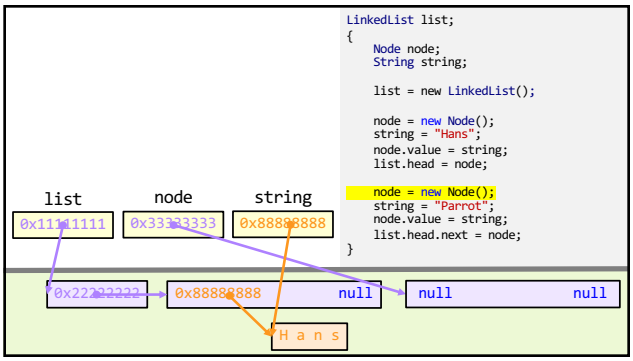

list
node
string
0x11111111
UNDEFINED
UNDEFINED
null
LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}

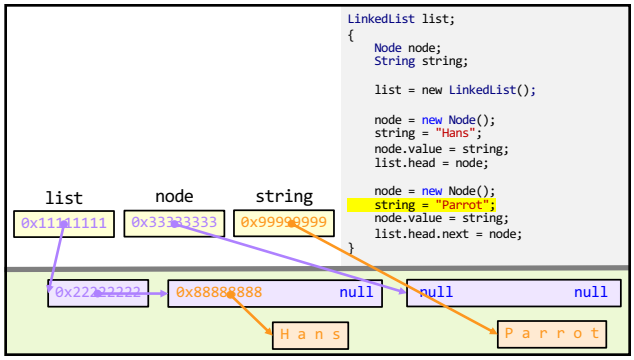

list
node
string
0x11111111
0x22222222
UNDEFINED
null
null
LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}

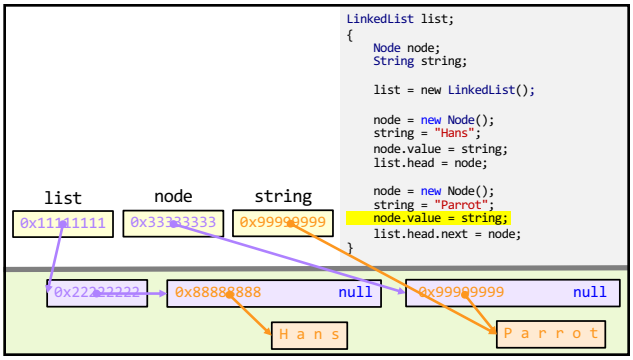

list
node
string
0x11111111
0x22222222
0x88888888
null
null
null
H a n s
LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}

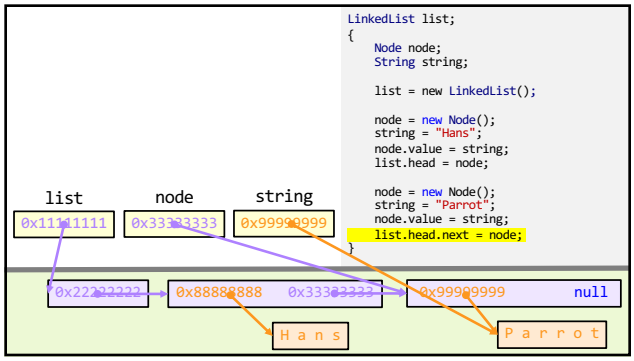

list
node
string
0x11111111
0x22222222
0x88888888
null
0x88888888
null
H a n s
LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}

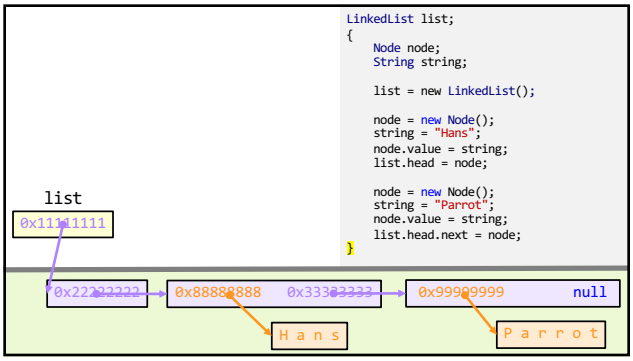

LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}
list
node
string
0x11111111
0x22222222
0x88888888
0x88888888
null
H a n s

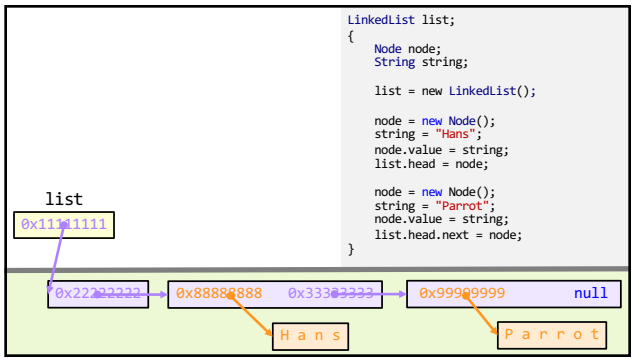

LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}
list
node
string
0x11111111
0x33333333
0x88888888
0x88888888
null
null
null
H a n s


LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}
list
node
string
0x11111111
0x33333333
0x99999999
0x88888888
null
null
null
H a n s
P a r r o t


LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}
list
node
string
0x11111111
0x33333333
0x99999999
0x88888888
null
0x99999999
null
H a n s
P a r r o t


LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}
list
node
string
0x11111111
0x33333333
0x99999999
0x88888888
0x33333333
0x99999999
null
H a n s
P a r r o t


LinkedList list;
{
Node node;
String string;
list = new LinkedList();
node = new Node();
string = "Hans";
node.value = string;
list.head = node;
node = new Node();
string = "Parrot";
node.value = string;
list.head.next = node;
}
list
0x11111111
0x88888888
0x33333333
0x99999999
null
H a n s
P a r r o t



memory in C

in C, we often get to choose whether to allocate memory on the stack or the heap

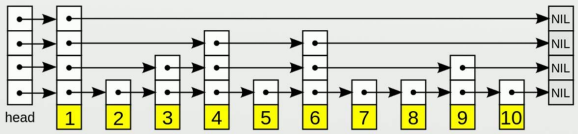
memory allocation

```java
// Java
int[] heapAllocatedArray = new int[10]
heapAllocatedArray[0] = 3;
```

```c
// C
int stackAllocatedArray[10];
int *heapAllocatedArray = malloc(10000000 * sizeof(int));

stackAllocatedArray[0] = 3;
heapAllocatedArray[0] = 3;

free(heapAllocatedArray);
```

256

Skip list



https://en.wikipedia.org/wiki/File:Skip_list.svg

334

Lists have no special operational treatment in Haskell. They are defined just like:

```
data List a = Nil | Cons a (List a)
```

Just with some special notation: `[a]` for `List a`, `[]` for `Nil` and `(:)` for `Cons`. If you defined the same and redefined all the operations, you would get the exact same performance.

Thus, Haskell lists are singly-linked. Because of laziness, they are often used as iterators. `sum [1..n]` runs in constant space, because the unused prefixes of this list are garbage collected as the sum progresses, and the tails aren't generated until they are needed.

```
; 4) Create some recursive functions
; Example create a power function
(defun power (x e)
    (if (eq e 0)
    1
    (* x (power x (- e 1)))
    )
)
; should print 125
(print (power 5 3))
(terpri)

(defun factor (x)
    nil
)
; Should print 120
(print (factor 5))
```

something fun?

fun friday menu

- jim rants about comments and syntax highlighting
- jim continues codeing (some of) microsoft paint from scratch
- jim updates you on his CAD software
- jim show and tells about his favorite element
- jim reads aloud from **grugbrain.dev**
- jim reads aloud from **vim creep**
- jim reads aloud from **hexing the technical interview**
- jim reads aloud from **the truth about lisp**
- jim reads aloud from **the worst API ever made**

midterm

common areas of struggle

question 1

- syntax
  - `list.add(int 3); // void add(int element);`
  - `True`
  - `}`

}

```
generally speaking {
    you want your curly braces {
    }
    to look {
        something like this {
        }
    }
}
```

}

```
generally speaking
{
    you want your curly braces
    {
    }
    to look
    {
        something like this
        {
        }
    }
}
```

}

```
this is quite spooky {
   and hard {
      to read }
   it's hard to tell {
      where one scope starts {
         and the next ends
         }}
do i have enough braces?
```

question 3

- control flow
  - returning too early (inside for loop)
  - handling special case too late (after creating array)
  - `if...if` vs. `if...else if` vs. `if...else`
- syntax
  - `list[i]`
  - `null`
  - `boolean 7Found;`
  - `;`

question 5

- complete understanding of app
  - pool
  - single enemy vs. multiple enemies

question 6

- complete understanding of app
  - `buffer` is 8 characters long, but only `length` many characters are drawn
- last question

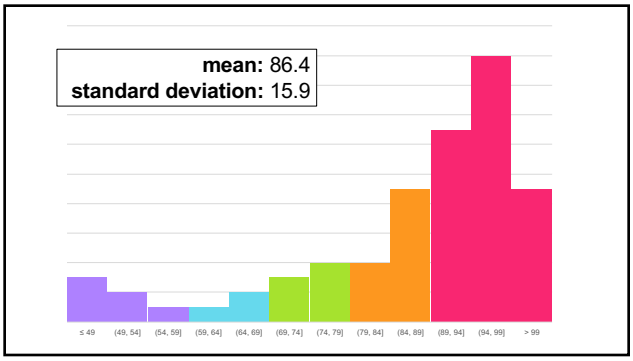
results

summay

- distribution is very wide (this is normal)
- some grades are low (< ~50)
  - this does **not** mean you will fail the coure

"**if you do all the homework**,
and the exams just aren't working for you,
you will pass CS136"
– James Bern, Fri Nov 1

- my comments are very sparse
  - if you don't know why you lost points,
    or you think i may have made a mistake
    please ask



thoughts