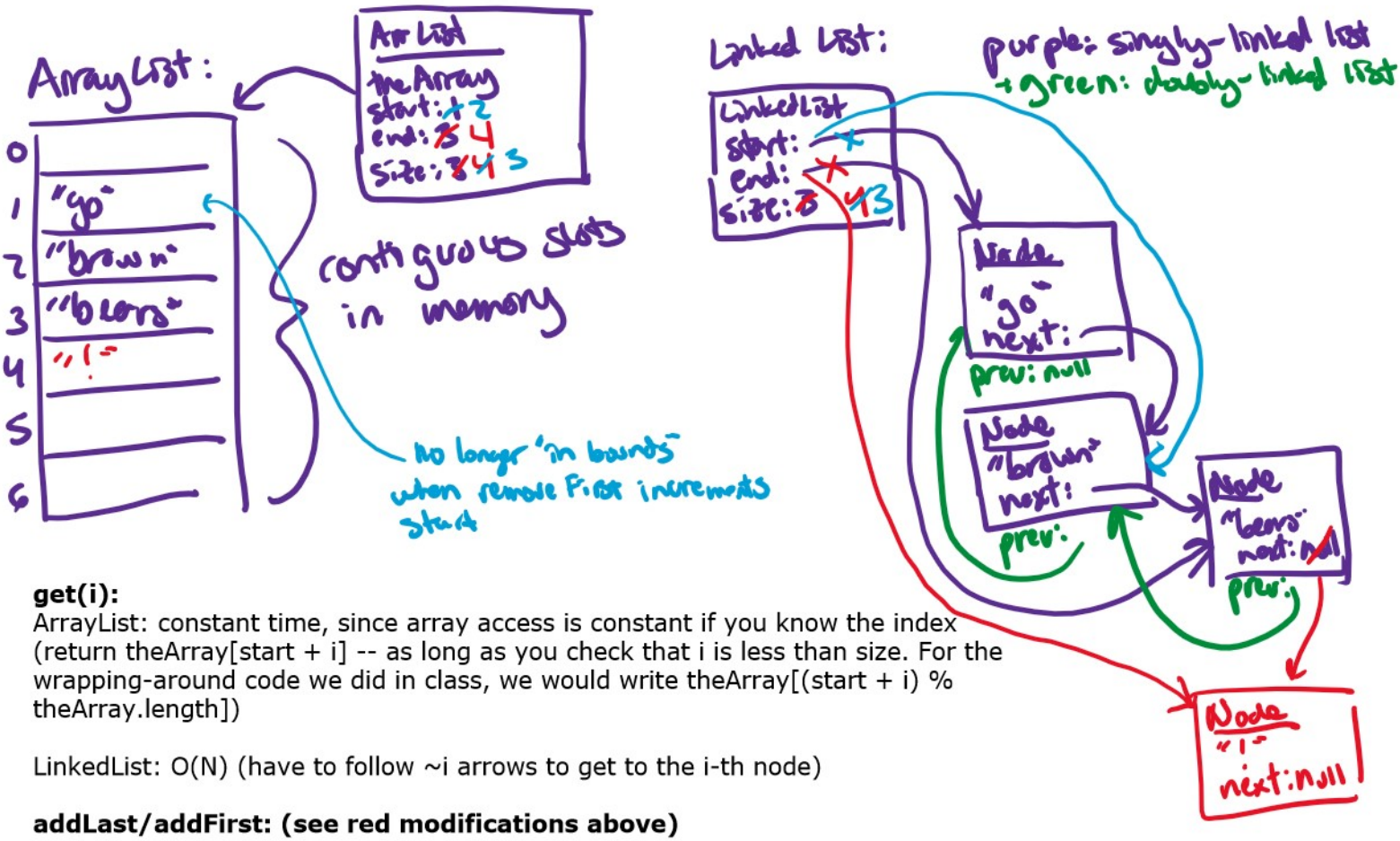
ArrayLists vs. Linked Lists runtime

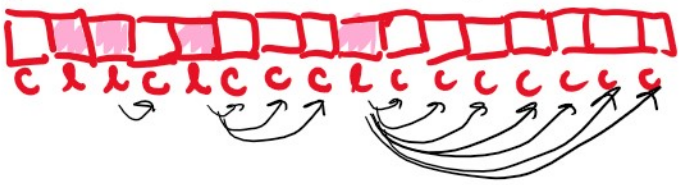
Tuesday, December 13, 2022 3:01 PM



get(i):
ArrayList: constant time, since array access is constant if you know the index (return theArray[start + i] -- as long as you check that i is less than size. For the wrapping-around code we did in class, we would write theArray[(start + i) % theArray.length])

LinkedList: O(N) (have to follow ~i arrows to get to the i-th node)

addLast/addFirst: (see red modifications above)
ArrayList: O(1) if we don't have to resize, O(N) if we do (to copy old array contents over into the new, bigger array)
If we double the array size every time we have to resize, this becomes an O(1) "amortized" (average/typical) operation, because the cost of resizing the array gains us about as many constant operations as it cost us copy operations



LinkedList: O(1) as long as we have start/end fields

removeLast/removeFirst: (see light blue modifications above for removeFirst)
ArrayList: O(1), since we can just decrement the end field (removeLast) or increment the start field (removeFirst). We could reset theArray contents at that index to some default value, but do not have to because that index is now considered "out of bounds" until some other addLast/addFirst operation overwrites it

LinkedList: removeFirst is O(1). The code looks something like: this.start = this.start.getNext(); removeLast is O(1) for a doubly-linked list (since we can use the prev field to write this.end = this.end.getPrev()), but O(N) for a singly-linked list since we have to follow the nodes to find the second-to-last element and make it the new end.

Different kinds of trees

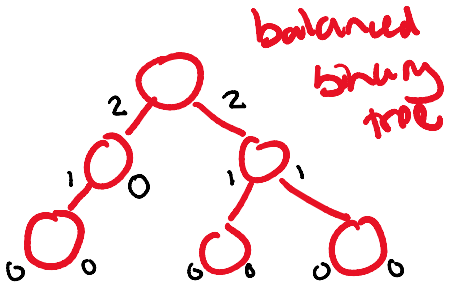
Wednesday, December 14, 2022 1:55 PM

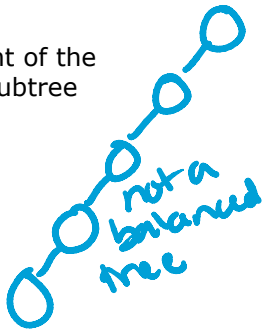
Tree: hierarchical data structure
Each node has zero or more subtrees and is either the root has exactly one parent

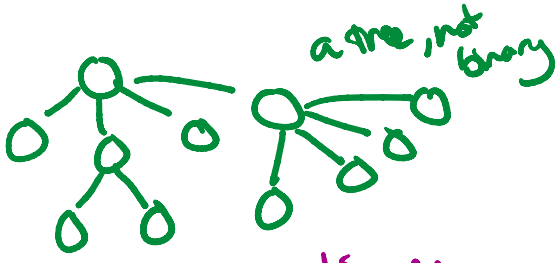
Binary trees: Each node has at most two subtrees

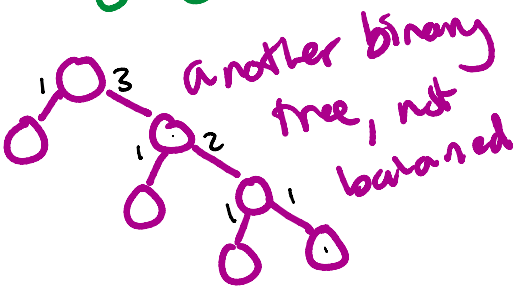
Balanced: is the height of the tree roughly log N (N is the number of nodes/elts in the tree)

> AVL balance condition: for every node, the height of the left subtree differs from the height of the right subtree by at most 1





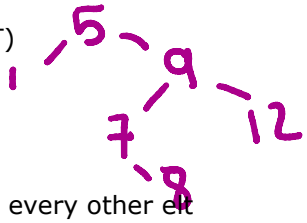




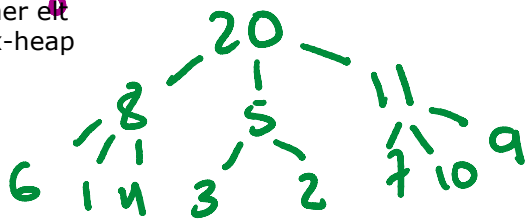
Constraints on the data a tree holds

Binary Search Tree: for every node, every elt. in the left subtree is smaller than the node and every elt. in the right subtree is bigger than the node

(hasElt is logN for a balanced BST)

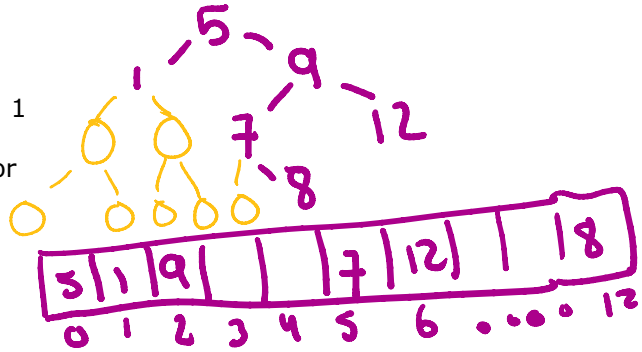


max-heap: the root is larger than every other elt in the tree, and every subtree is also a max-heap



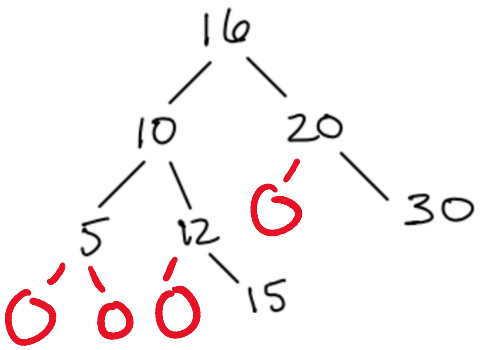
Embedding binary trees in arrays:
root is at elt 0
for every elt, if it is at index i, its left child will be at index 2*i + 1 and its right child will be at index 2*i + 2

> "storing the tree top-to-bottom, left-to-right, leaving holes for empty spots)



Spring 2022 Exam, Heaps question

Wednesday, December 14, 2022 3:46 PM

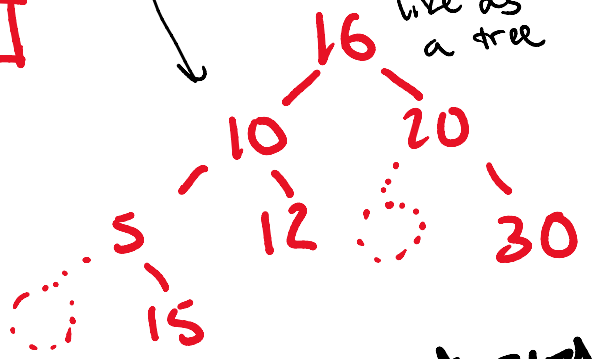


16	10	5	12	15	20	30			
16	10	20	5	12		30	15		
5	10	12	15	16	20	30			
☆ 16	10	20	5	12		30		15	

what this tree looks like as an array:

16	10	20	5	12		30			15

what this array looks like as a tree



From the next question: not all arrays can represent valid trees:

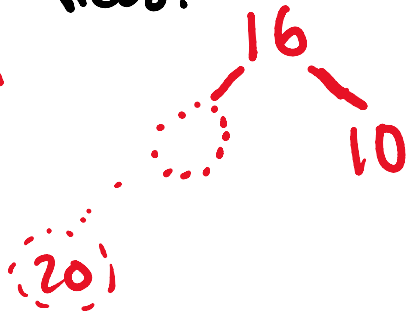
☆ 16		10	20	5		12	30	15	
16	20	5	12		30		15		
16		20	5	12		30	15		
16	12	20	5	15		30			

☐ X

☐

☐

☐



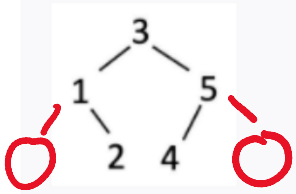
2021 final Q3.1-3.2

Wednesday, December 14, 2022 2:06 PM

Q3 Tree-Based Data Structures

24 Points

Consider the following binary tree:



Q3.1 which DS

4 Points

Which of the following data structures is this binary tree an example of?

- [x] binary search tree (whether balanced or not)
- [ ] heap (whether balanced or not)
- [ ] neither one

Q3.2 Trees via arrays

6 Points

Which of the following arrays capture the tree (from the figure above) in a way that would let us navigate between parent and child nodes based solely on arithmetic on array indicies (as we did in lecture)?

Array A	3	1	5	2	4			
Array B	3	1	5		2	4		
Array C	3	1	2			5	4	
Array D		3	1		5	4	2	

2021 final Q3.3

Wednesday, December 16, 2022 2:06 PM

Q3.3 Examples of Heaps

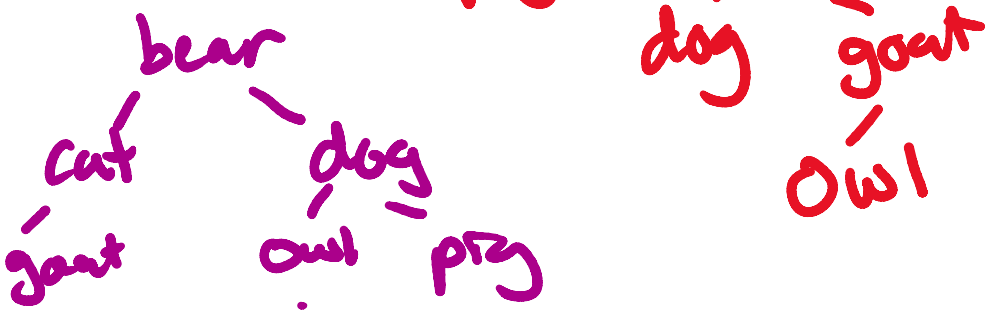
6 Points

Heaps can be used to sort any type of data on which there is an ordering. Draw **TWO** different heaps on the following set of words, where priority is based on alphabetical order. Upload a single picture showing both of your heaps.

cat, dog, bear, pig, owl, goat

No files uploaded

min heap or max heap?
balanced or imbalanced?
(more free to put elements below other elements, as long as they follow the heap property)



2021 final Q3.4

Wednesday, December 14, 2022 2:06 PM

Q3.4 Arrays vs Classes

8 Points

We have seen two ways to represent trees as data: recursive classes and arrays. As a reminder, the recursive class might look like:

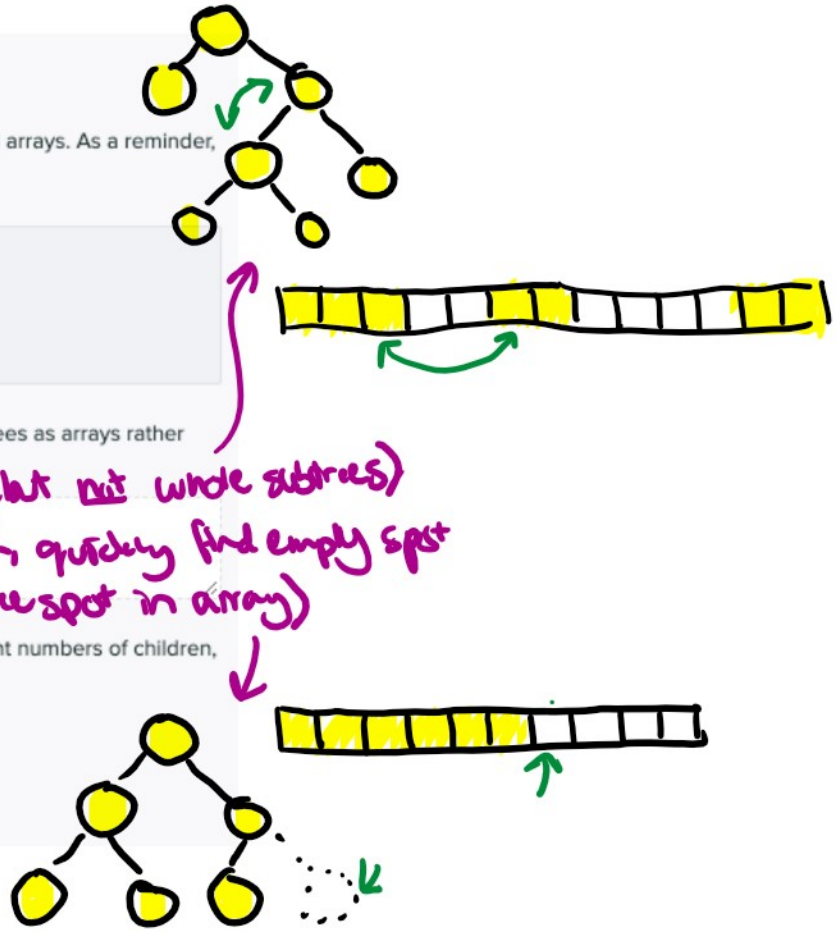
```
class BinaryTree[T] {
  T data;
  BinaryTree left;
  BinaryTree right:
}
```

Briefly explain (1-2 sentences) **TWO** advantages to representing binary trees as arrays rather than as classes.

- quickly swap parents & children (but not whole subtrees)
- if you're filling in a heap L-to-R, quickly find empty spot (first free spot in array)

If you were asked to represent trees in which different nodes had different numbers of children, which approach would you prefer?

- ○ Arrays
- ◉ A Recursive Class (perhaps modified from the above)
- ○ No preference

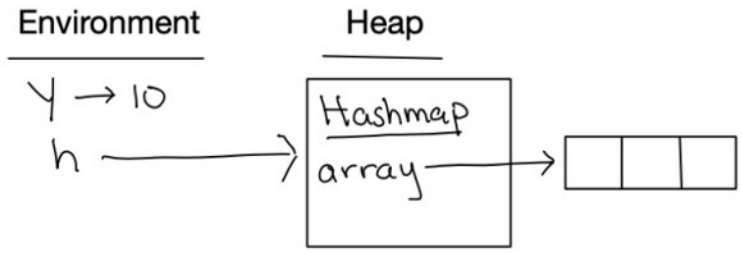


Hashmap Memory Diagram

Wednesday, December 14, 2022 1:54 PM

Finish the memory diagram for the following code fragment, which creates a HashMap with three slots. Assume the `hashcode` method on the string key returns the length of the string (e.g., the hashcode of "cs200" is 5).

```
h = new HashMap<String, Number>(3) // 3 is the size of the array
y = 10
h.insert("red", y)
h.insert("brown", 15)
h.insert("yellow", h.get("red") + 2)
```



We did not have time to go over this problem in the review session, but here is what you would do:

- figure out which array index each key hashes to by computing the hashcode and taking the modulus/remainder (%) with the array length, which is 3:
  - index for "red" is 3 % 3 = 0
  - index for "brown" is 5 % 3 = 2
  - index for "yellow" is 6 % 3 = 0
- Fill in the linked lists at each array index with the corresponding KV pairs. Note that "red" and "yellow" both hash to the same index, but yellow is added after red, so the KV pair with "yellow" should come second in the linked list at array index 0 (see the solution for this)

This exam question was also graded by checking that:

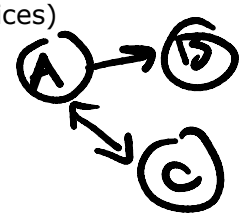
- no other names (besides y and h) appear in the environment
- names from the environment (such as "y") do not appear in the heap

Graph representations

Wednesday, December 14, 2022 1:56 PM

We did not have time to go over this in the review session, but here are some ways we have seen to represent graphs and their tradeoffs:

Example of very basic representation: a hashmap from Strings (representing vertex names) to lists or sets of Strings (the neighboring vertices)



E.g.
{"A": ["B", "C"],
"B": [],
"C": ["A"]}

Pros: simple, easy to find neighbors
Cons: doesn't work for weighted graphs, no data associated with vertices except for name, not very "OO-friendly" (vertices cannot perform their own operations because there is no vertex class)

Designs that contain a Vertex class:

- Vertex class contains a list/set of neighboring vertices
  (e.g. object for vertex "A" above would contain a list/set with the object for vertex "B" as the element)
  Cons: doesn't work for weighted graphs, no easy way to ask if Vertex has a neighbor of a certain name
- Vertex class contains a dictionary from neighboring vertex names to objects
  (e.g. object for vertex "A" above would contain a dictionary that maps from "B" to the object for vertex "B")
  Cons: doesn't work for weighted graphs, coding it up takes a little more awareness of how underlying data should look
- Vertex class contains a list/set of outgoing Edge objects, where each Edge object may have a weight and fields for the start and end vertices
  Cons: depending on the implementation, may not need this level of complexity

We might also have a Graph class that holds vertex objects. We would choose our data structures based on the operations we want to perform on the graph (for example, for TravelPlanner, you needed to have a way to associate a city name with a vertex object, so you likely chose to keep a dictionary from city names to vertices).

There are other variations here (for example, instead of having an edge object, you could maintain a dictionary from neighbor vertices to edge weights inside of the edge class). It all boils down to the operations you want to do on the graph (easily store edge weights? Find neighbors quickly? Look up a vertex with a given name?).

Note that we put "list/set" above - the choice of which one depends on what operations you want to do on the graph. For example, if you wanted a fast Vertex method like "hasNeighbor", you would probably choose a set for its constant-time lookup.