CSE 373

OCTOBER 27TH – PRIORITY QUEUES

TODAY

- **HW 2 grades went out yesterday**
  - Many of the problems seemed to be a problem with rigor
  - In proofs, justify what you’re saying and make as much explicit as you can

TODAY

- **HW 2 grades went out yesterday**
  - Many of the problems seemed to be a problem with rigor
  - In proofs, justify what you're saying and make as much explicit as you can
  - https://catalyst.uw.edu/webq/survey/ejmcc/340863

TODAY

- **B+-Trees**
  - Conclusion and important info
- **New ADT – Priority Queue**

B-TREE EXAMPLE

- Let M (the number of children from a signpost) be 3 and let L (the number of k,v pairs in a leaf) be 1
- This is a 3-1 tree (uncommon, but useful for demonstration).

B-TREES

- **B-Trees do not receive a benefit unless their nodes are page aligned**
  - If the nodes overlap a page boundary, we are doubling the number of potential disk accesses

B-TREES

- **B-Trees do not receive a benefit unless their nodes are page aligned**
  - If the nodes overlap a page boundary, we are doubling the number of potential disk accesses
- **Because of this, B-trees are not implemented in Java.**

B-TREES

- **Designed based on our knowledge of memory architecture**
    - If a disk access brings a whole page into memory (or cache), make sure that we get the maximum amount of information.
- **When we bring in a signpost, we can use fast in-memory binary search to find the correct child**

B-TREES

- **Important things to remember**
  - Signposts v. Leaves
  - Performing a find
  - Runtime analysis
  - Inserting in simple cases
  - Calculating M and L

B-TREE

- Conclusion

B-TREE

- **Conclusion**
  - Good data structure for working with and understanding memory and the disk

B-TREE

- **Conclusion**
  - Good data structure for working with and understanding memory and the disk
  - More complicated analysis, but comes after recognizing that bigO assumes equal memory access

B-TREE

- **Conclusion**
  - Good data structure for working with and understanding memory and the disk
  - More complicated analysis, but comes after recognizing that bigO assumes equal memory access
  - Computer architecture constraints have real-world impacts that can be corrected for

B-TREE

- **Conclusion**
  - Good data structure for working with and understanding memory and the disk
  - More complicated analysis, but comes after recognizing that bigO assumes equal memory access
  - Computer architecture constraints have real-world impacts that can be corrected for
  - Theory is great, but it has its limitations

PRIORITY QUEUE

- New ADT

PRIORITY QUEUE

- **New ADT**
- **Objects in the priority queue have:**
  - Data
  - Priority

PRIORITY QUEUE

- **New ADT**
- **Objects in the priority queue have:**
  - Data
  - Priority
- **Conditions**
  - Lower priority items should dequeue first
  - Should be able to change priority of an item
  - FIFO for equal priority?

PRIORITY QUEUE

- **insert(K key, int priority)**
  - Insert the key into the PQ with given priority
- **findMin()**
  - Return the key that currently has lowest priority in the PQ (min-heap)
- **deleteMin()**
  - Return and remove the key with lowest priority
- **changePriority(K key, int newPri)**
  - Assign a new priority to the object key

PRIORITY QUEUE

- Applications?

PRIORITY QUEUE

- **Applications?**
  - Hospitals
  - CSE course overloads
  - Etc…

PRIORITY QUEUE

- **How to implement?**

PRIORITY QUEUE

- How to implement?
- Array?

PRIORITY QUEUE

- **How to implement?**
- **Array?**
  - Must keep sorted
  - Inserting into the middle is costly (must move other items)

PRIORITY QUEUE

- **How to implement?**
  - Keep data sorted (somehow)
- **Array?**
  - Inserting into the middle is costly (must move other items)
- **Linked list?**
  - Must iterate through entire list to find place
  - Cannot move backward if priority changes

PRIORITY QUEUE

- These data structures will all give us the behavior we want as far as the ADT, but they may be poor design decisions
- Any other data structures to try?

PRIORITY QUEUE

- **Priority queue implementations?**

PRIORITY QUEUE

- **Priority queue implementations?**
  - Binary search tree?

PRIORITY QUEUE

- **Priority queue implementations?**
  - Binary search tree?
    - Faster insert

PRIORITY QUEUE

- **Priority queue implementations?**
  - Binary search tree?
    - Faster insert
    - Find? Always deleting the smallest (left-most) element

PRIORITY QUEUE

- **Priority queue implementations?**
  - Binary search tree?
    - Faster insert
    - Find? Always deleting the smallest (left-most) element
    - Changing priority?

PRIORITY QUEUE

- Want the speed of trees (but not BST)
- Priority Queue has unique demands

PRIORITY QUEUE

- Want the speed of trees (but not BST)
- Priority Queue has unique demands
- Other types of trees?

PRIORITY QUEUE

- Want the speed of trees (but not BST)
- Priority Queue has unique demands
- Other types of trees?
- Review BST first

PROPERTIES (BST)

- **Tree**

PROPERTIES (BST)

- **Tree (Binary)**
  - Root
  - (Two) Children
  - No cycles

PROPERTIES (BST)

- **Tree (Binary)**
  - Root
  - (Two) Children
  - No cycles
- **Search**

PROPERTIES (BST)

- **Tree (Binary)**
  - Root
  - (Two) Children
  - No cycles
- **Search**
  - Comparable data
  - Left child data < parent data

PROPERTIES (BST)

- **Tree (Binary)**
  - Root
  - (Two) Children
  - No cycles
- **Search**
  - Comparable data
  - Left child data < parent data
  - Smallest child is at the left most node

PROPERTIES (BST)

- **Binary tree may be useful**
- **Search property doesn’t help**

PROPERTIES (BST)

- **Binary tree may be useful**
- **Search property doesn't help**
  - Always deleting min

PROPERTIES (BST)

- **Binary tree may be useful**
- **Search property doesn’t help**
  - Always deleting min
  - Put min on top!

HEAP-ORDER PROPERTY

- Still a binary tree

HEAP-ORDER PROPERTY

- Still a binary tree
- Instead of search (left < parent),

HEAP-ORDER PROPERTY

- Still a binary tree
- Instead of search (left < parent),
  parent should be less than children

HEAP-ORDER PROPERTY

- Still a binary tree
- Instead of search (left < parent),
  parent should be less than children
- How to implement?
- Insert and delete are different than BST

HEAP-ORDER PROPERTY

- Still a binary tree
- Instead of search (left < parent),
  parent should be less than children
- How to implement?
- Insert and delete are different than BST

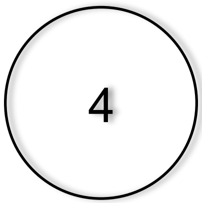
HEAPS

- The Priority Queue is the ADT
- The Heap is the Data Structure

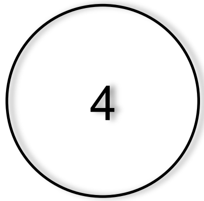
HEAP EXAMPLE

- Only looking at priorities
- Insert something priority 4

HEAP EXAMPLE

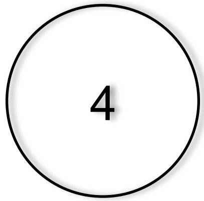


HEAP EXAMPLE



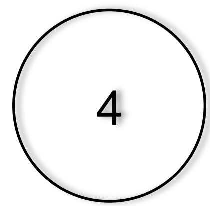
- Now insert priority 6?

HEAP EXAMPLE



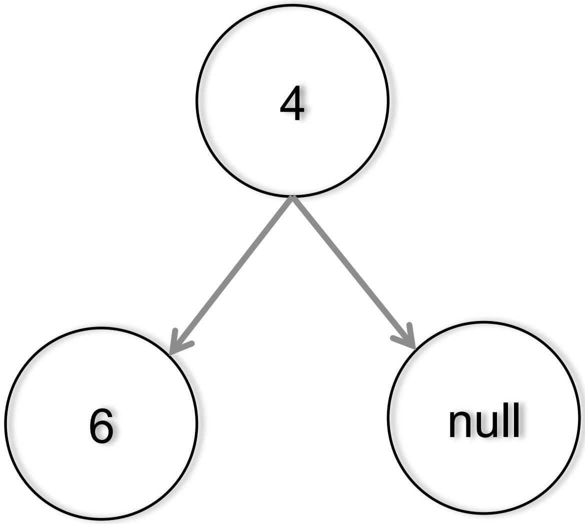
- Now insert priority 6?
- Should come after 4, but no preference right over left?

HEAP EXAMPLE



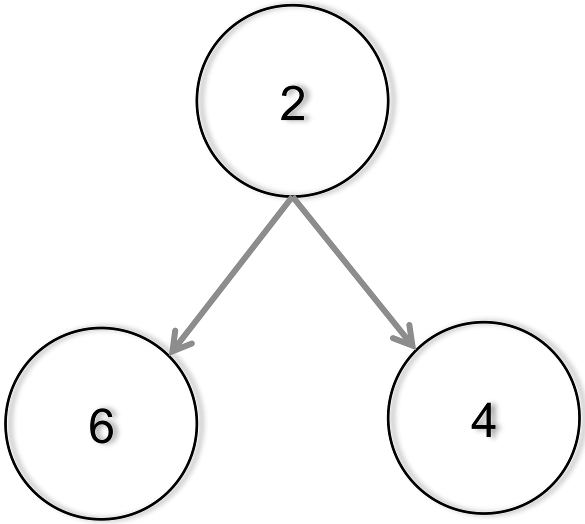
- **Now insert priority 6?**
- **Should come after 4, but no preference right over left?**
- **Solution: fill the tree from top to bottom left to right.**

HEAP EXAMPLE



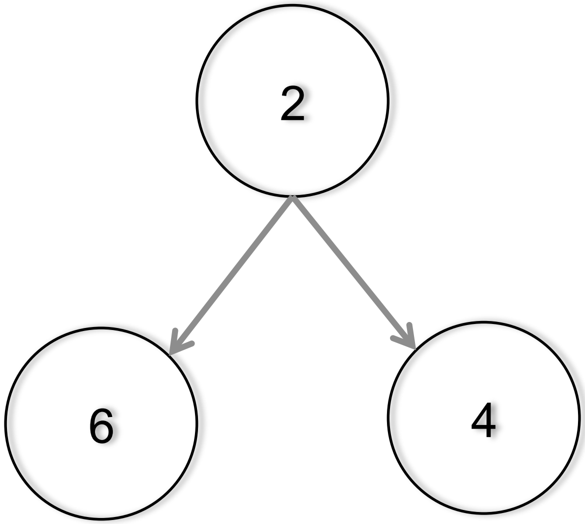
Now insert 2.

HEAP EXAMPLE



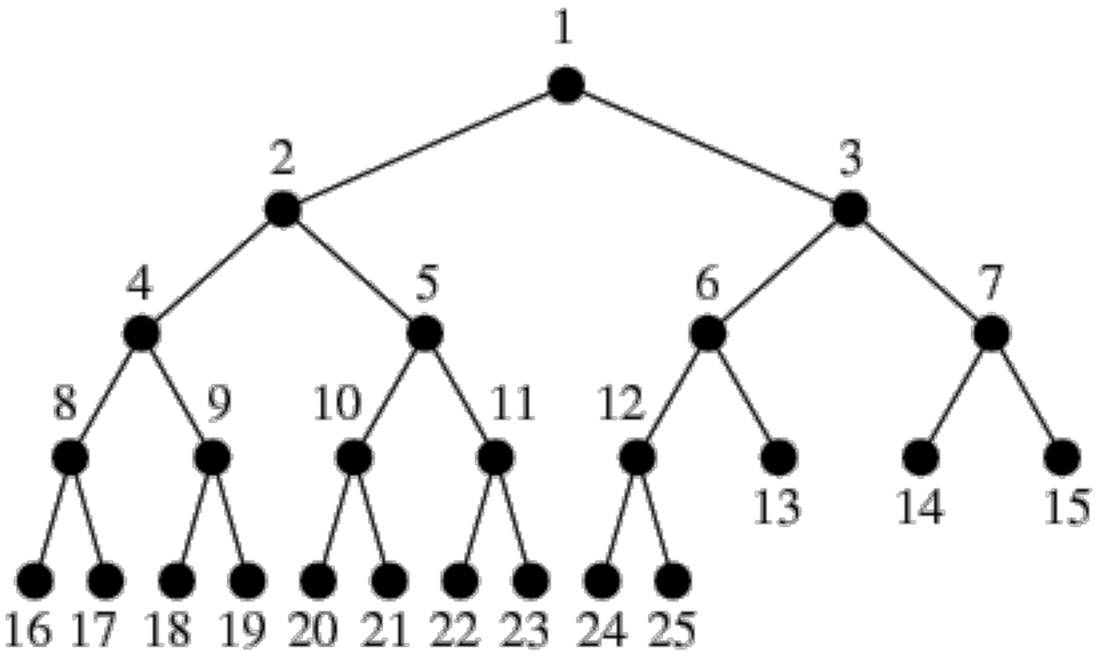
Now insert 2.

HEAP EXAMPLE

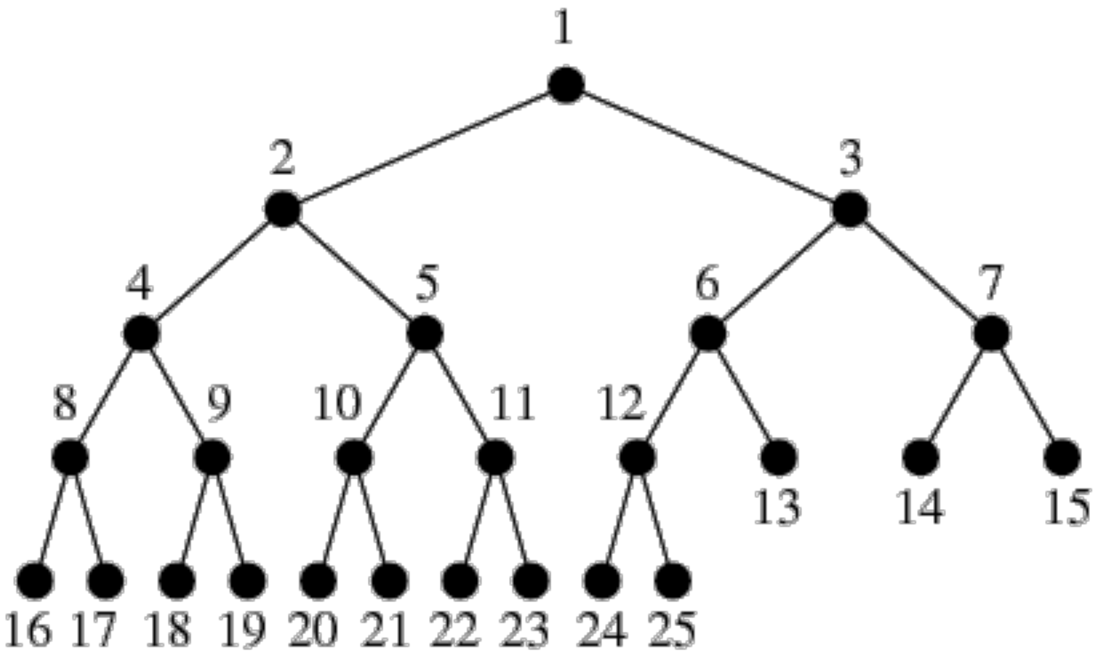


Could easily have been 4 on the left, but our left to right top to bottom strategy determines this solution

COMPLETENESS



COMPLETENESS



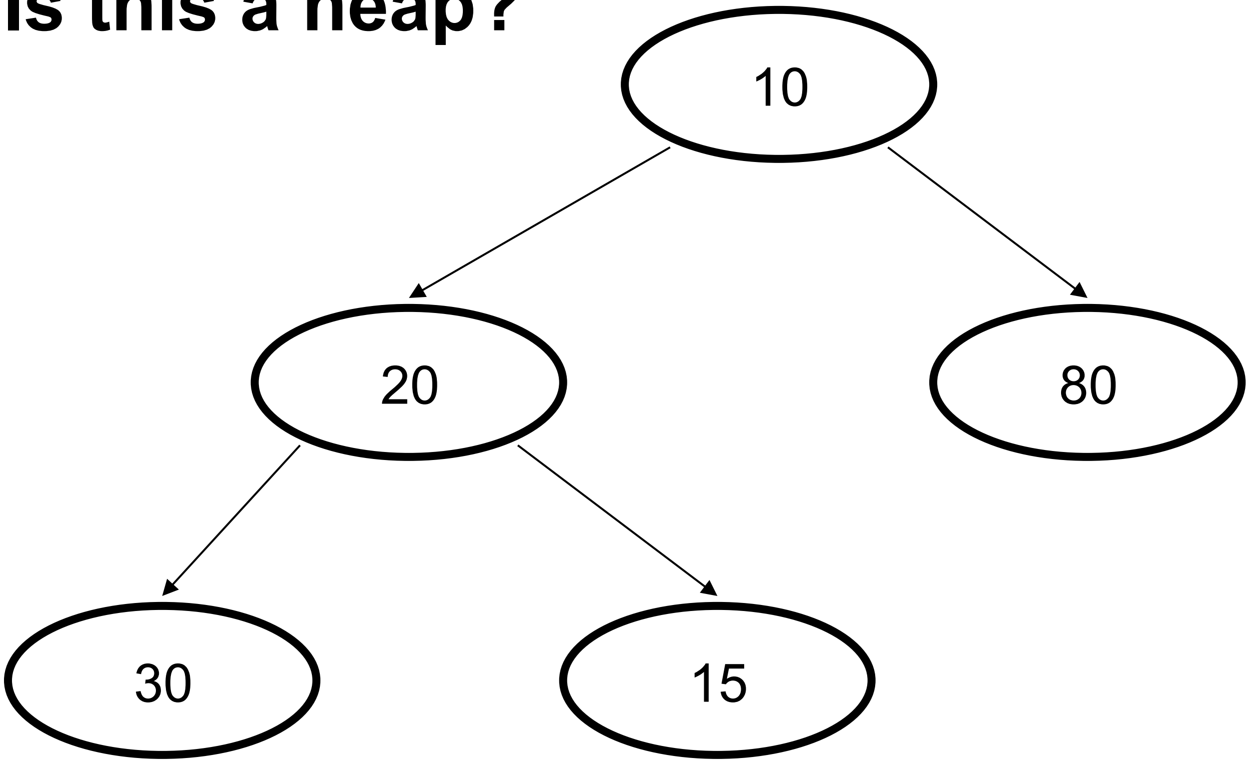
Filling left to right and top to bottom is another property - completeness

HEAPS

- Heap property (parents < children)
- Complete tree property (left to right, bottom to top)

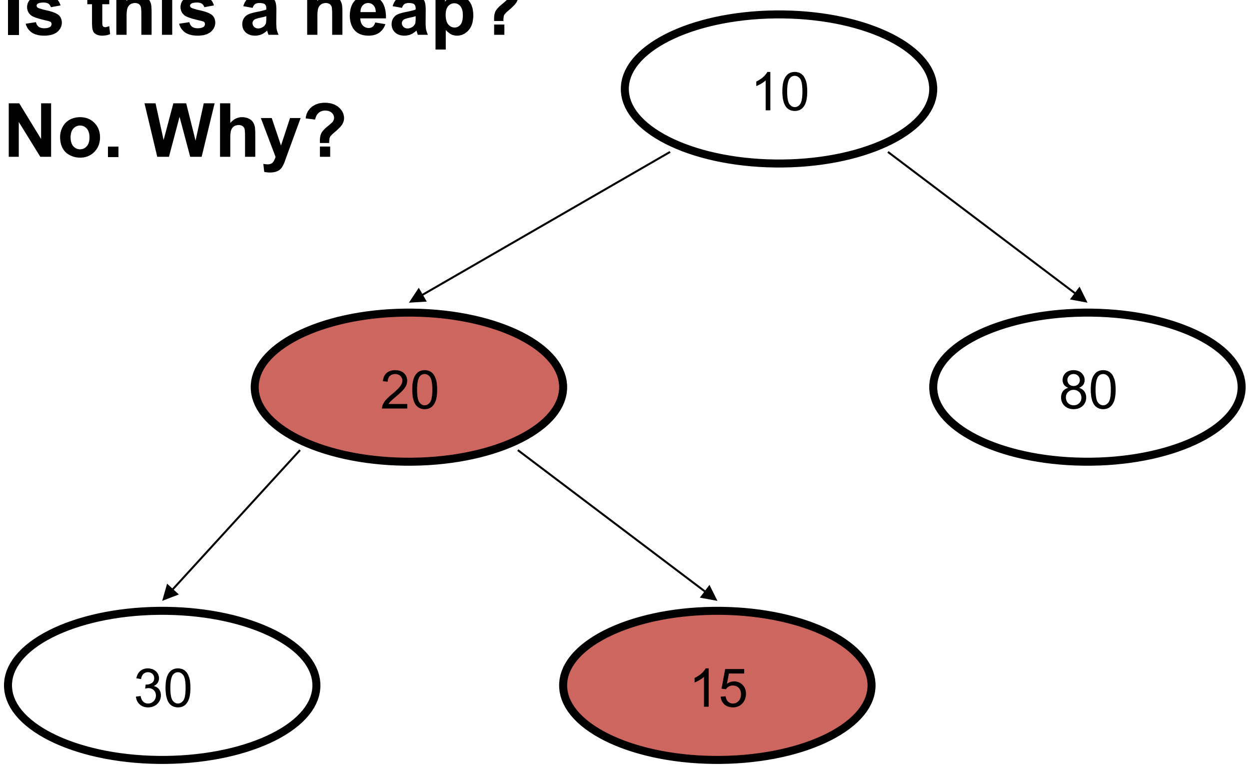
REVIEW

- **Is this a heap?**



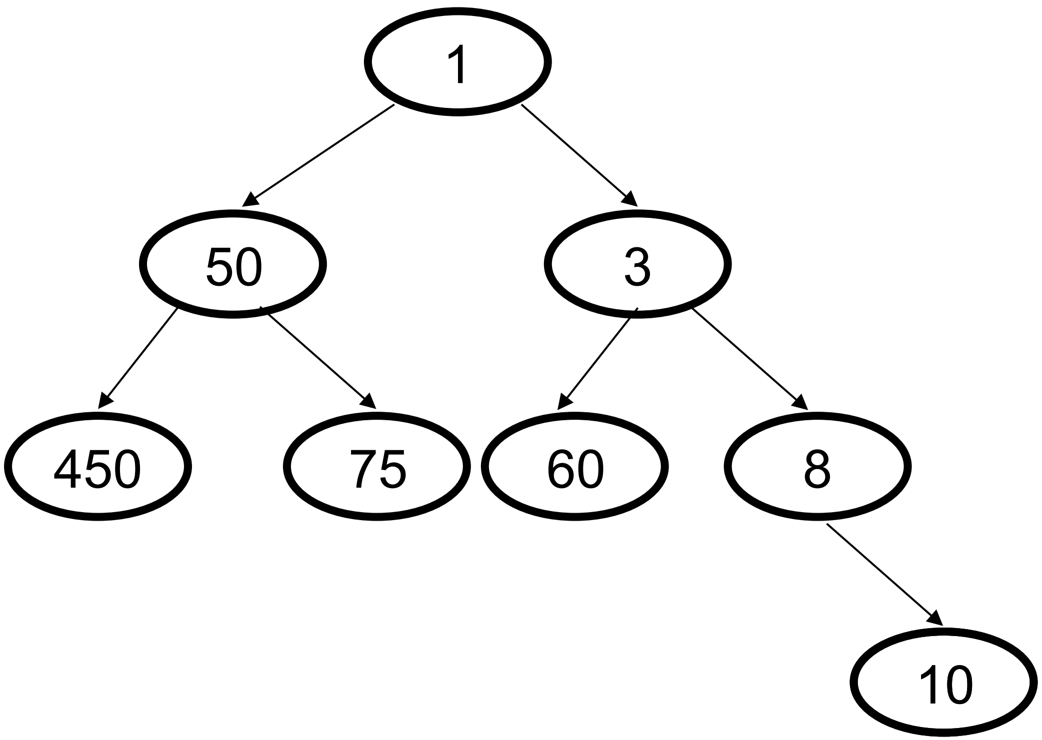
REVIEW

- **Is this a heap?**
- **No. Why?**



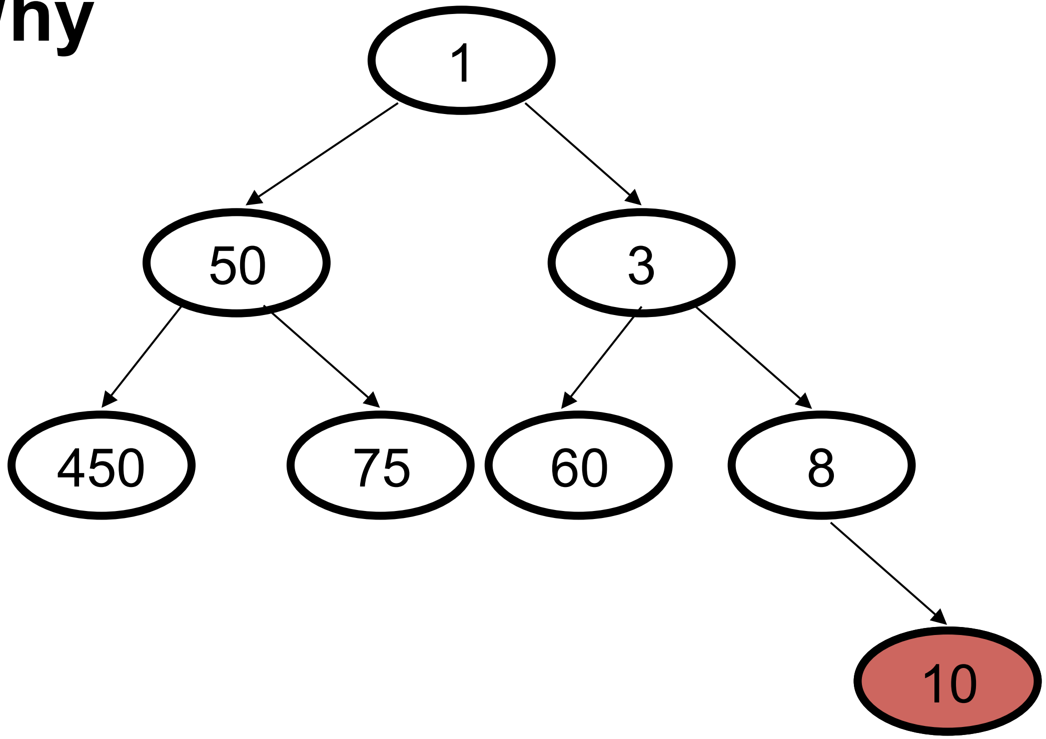
REVIEW

- **Is this a heap?**



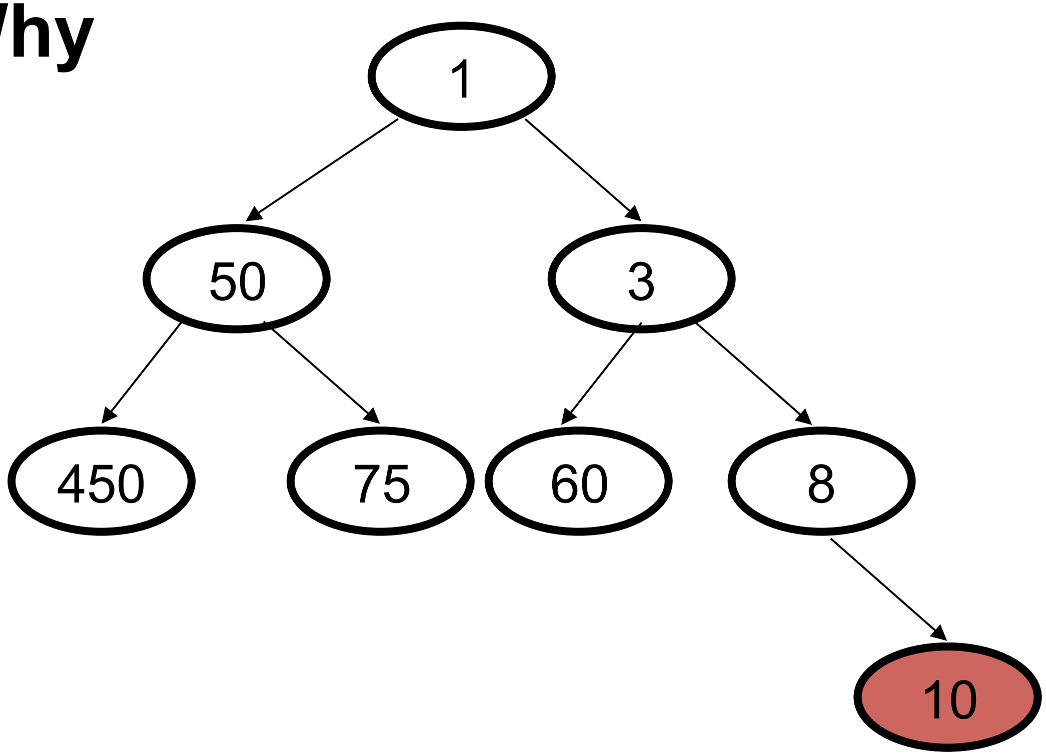
REVIEW

- Is this a heap?
- No. Why



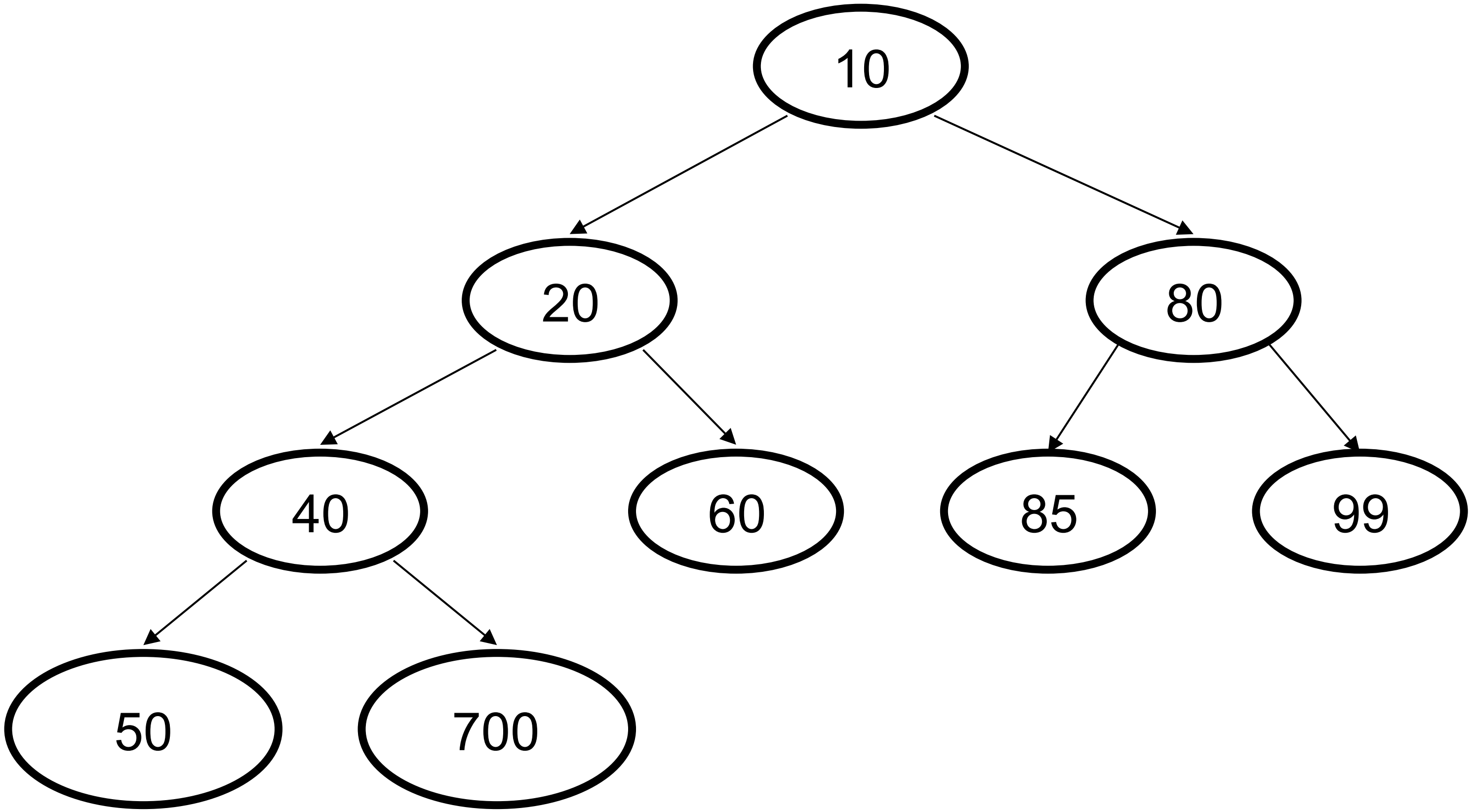
REVIEW

- Is this a heap?
- No. Why



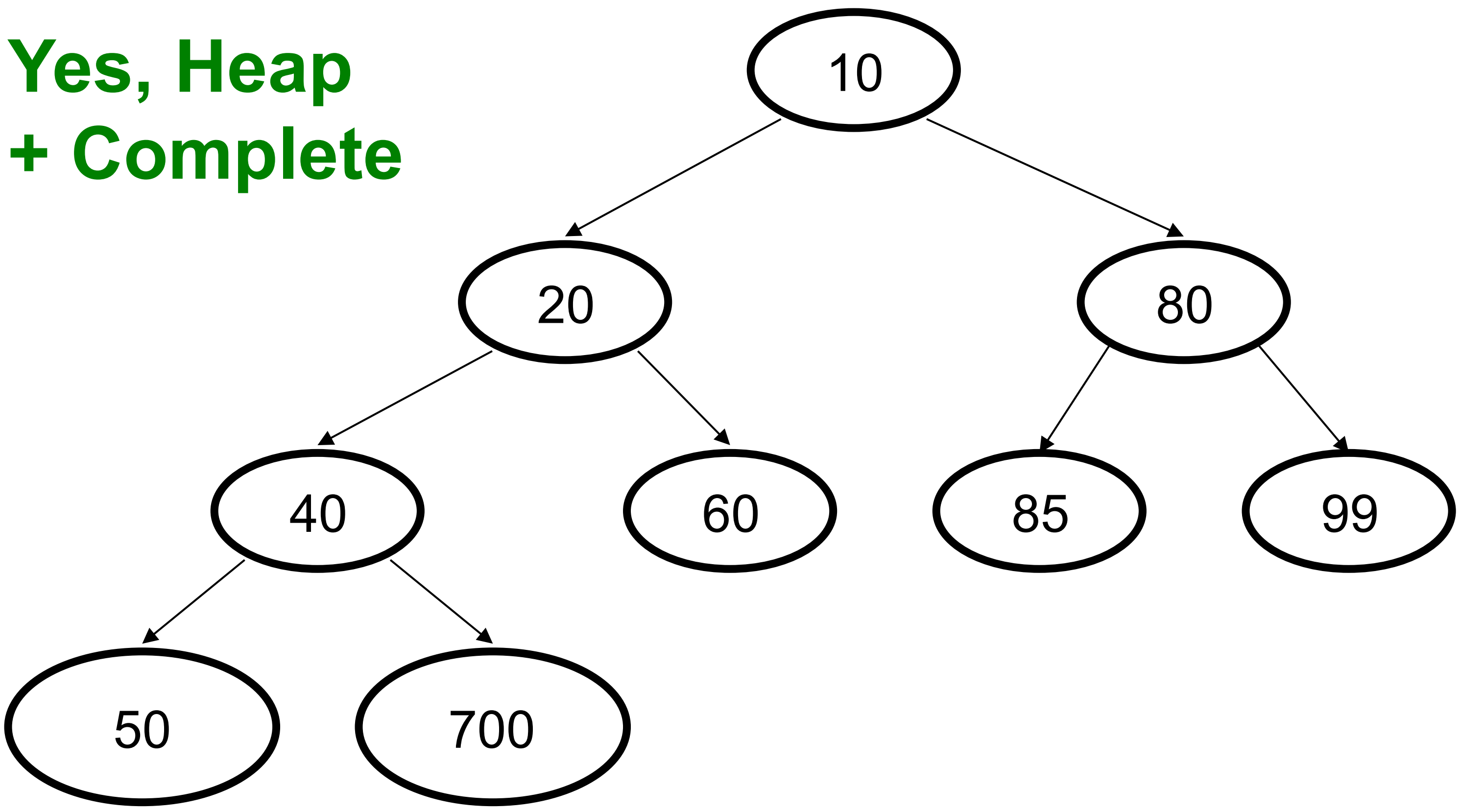
REVIEW

- **Is this a heap?**



REVIEW

- **Is this a heap?**
- **Yes, Heap**
  + Complete

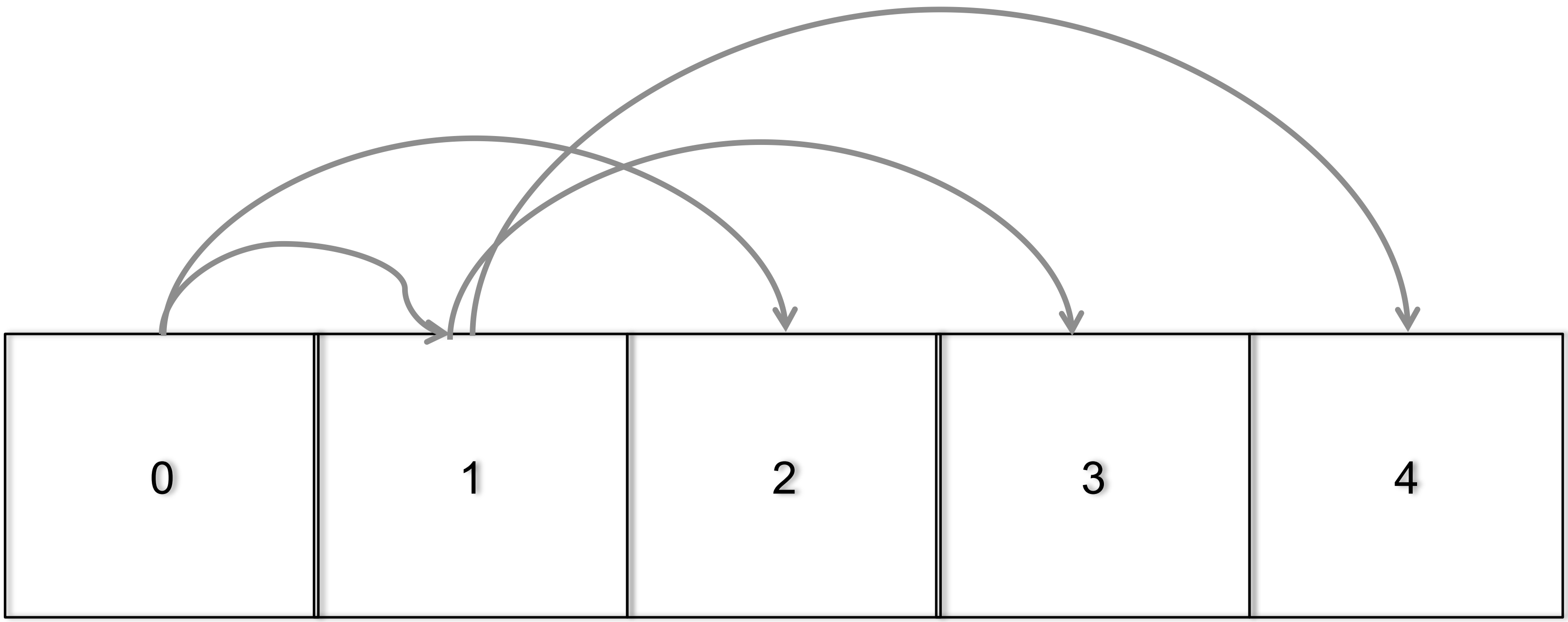


HEAPS

- **Heap property (parents < children)**
- **Complete tree property (left to right, bottom to top)**
- **How does this help?**
  - Array implementation

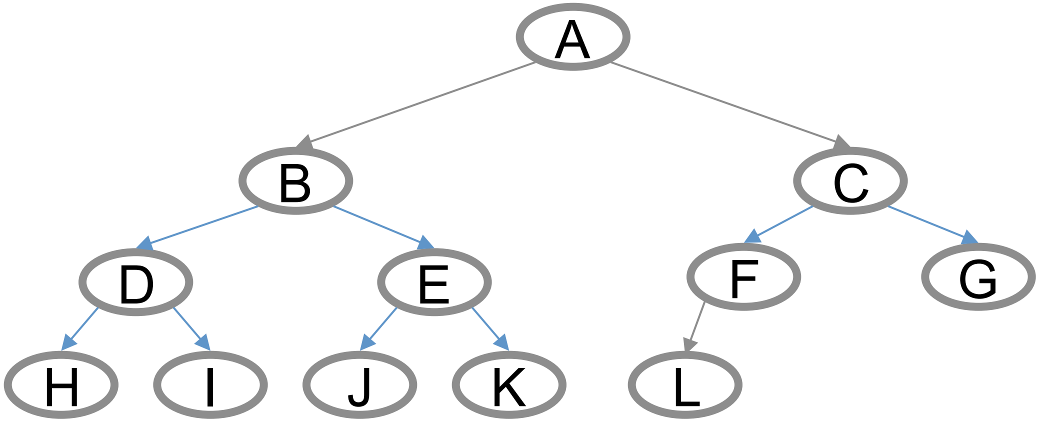
HEAPS

- Insert into array from left to right
- For any parent at index i,
  children at 2*i+1 and 2*i+2



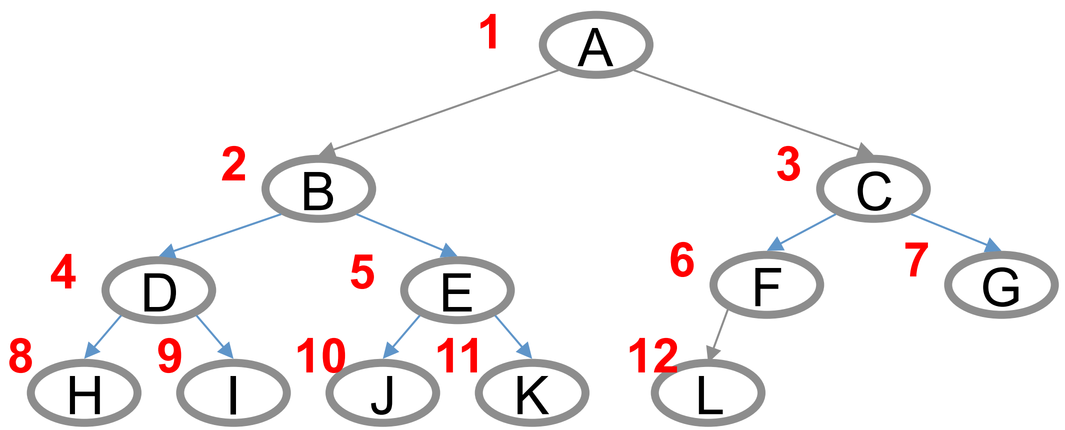
REVIEW

- **Array property (with 1 indexing)**



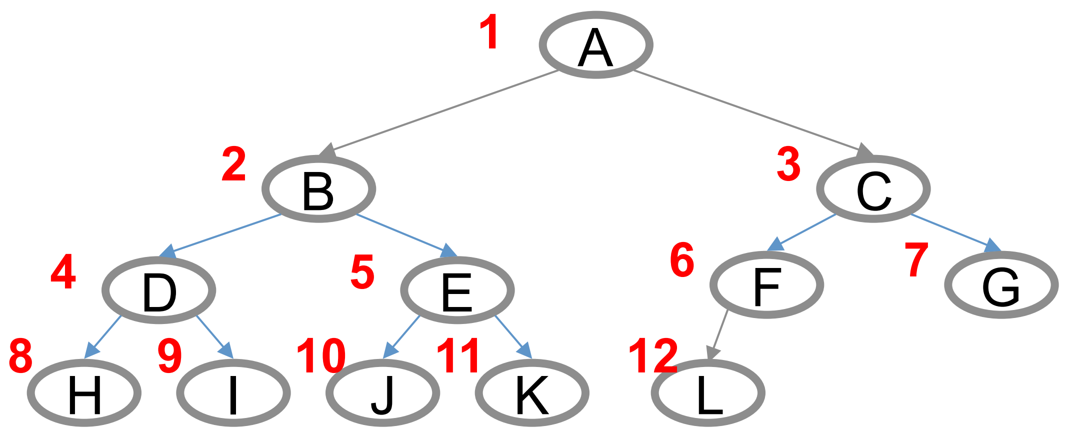
REVIEW

- **Array property**



REVIEW

- **Array property**



	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

HEAPS

- https://www.cs.usfca.edu/~galles/visualization/Heap.html
- **Another visualizer**

HEAPS

- **How to maintain heap property then?**

HEAPS

- **How to maintain heap property then?**
  - Parent must be higher priority than children

HEAPS

- **How to maintain heap property then?**
  - Parent must be higher priority than children
- **Two functions – percolate up and percolate down**

HEAP FUNCTIONS

- **Percolate up**
  - When a new item is inserted:
    - Place the item at the next position to preserve completeness
    - Swap the item up the tree until it is larger than its parent

HEAP FUNCTIONS

- **Percolate down**
  - When an item is deleted:
    - Remove the root of the tree (to be returned)
    - Move the last object in the tree to the root
    - Swap the moved piece down while it is larger than it's smallest child
    - Only swap with the smallest child

HEAPS AS ARRAYS

- **Because heaps are complete, they can be represented as arrays without any gaps in them.**
- **Naïve implementation:**
  - Left child: 2*i+1
  - Right child: 2*i + 2
  - Parent: (i-1)/2

HEAPS AS ARRAYS

- **Alternate (common) implementation:**
  - Put the root of the array at index 1
  - Leave index 0 blank
  - Calculating children/parent becomes:
    - Left child: 2*i
    - Right child: 2*i + 1
    - Parent: i/2

HEAPS AS ARRAYS

- **Why do an array at all?**

HEAPS AS ARRAYS

- **Why do an array at all?**
  - + Memory efficiency
  - + Fast accesses to data
  - + Forces log n depth
  - - Needs to resize
  - - Can waste space

HEAPS AS ARRAYS

- **Why do an array at all?**
  - \+ Memory efficiency
  - \+ Fast accesses to data
  - \+ Forces log n depth
  - \- Needs to resize
  - \- Can waste space
- **Almost always done through an array**

NEXT WEEK

- **Analysis of the heap**
- **buildHeap()—a unique case and analysis**