Quiz 05 Review Session

COMP 210 / 2024 Summer Session I

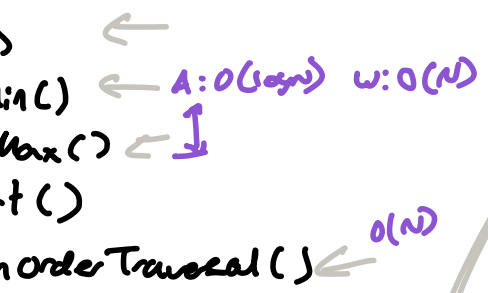
Ajay Gandecha

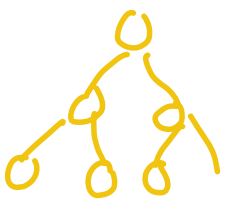
Quiz 05 Format

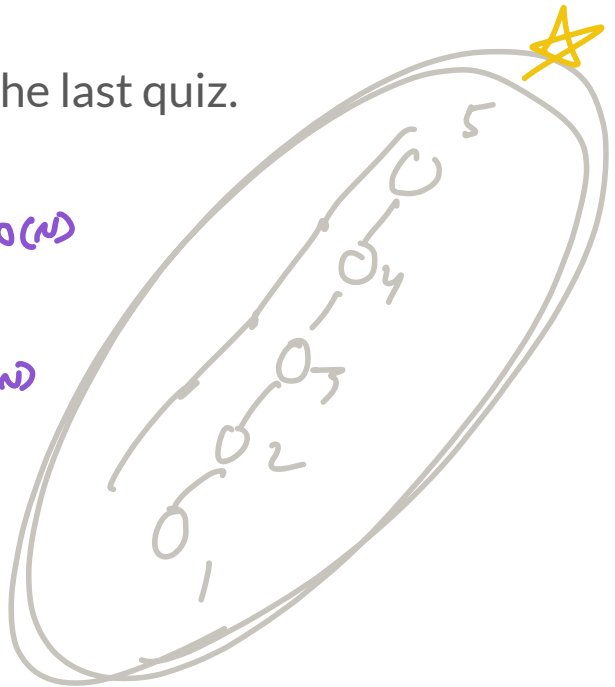
- 30 minutes at the start of class.
- *On paper* - bring a pencil!
- **Question Types:**
  - Multiple choice, T/F, select all that apply, fill in the blank, (drawing trees) # unsure

Exercise Check-In Question

- Similar format to the exercise question on the last quiz.
- Review `Ex08`.
- *Questions?*







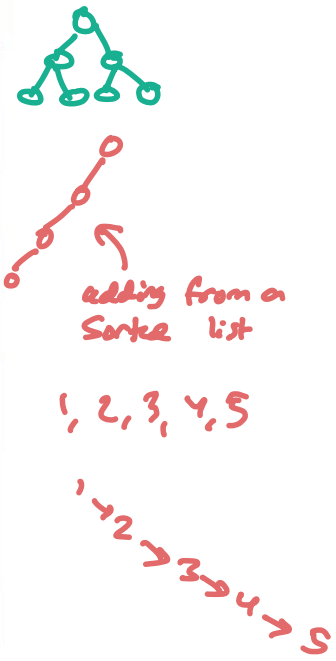
On Quiz 05

- Trees, Binary Trees, Binary Search Trees
- Time Complexity of Binary Search Trees
- Priority Queues, Min. Binary Heaps

BST Operations - Time Complexities

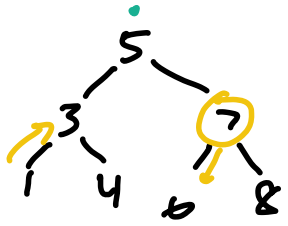
Operation	Worst Case	Average Case
Insert	O(n)	O(logn)
Remove	O(n)	O(logn)
findMin	O(n)	O(logn)
findMax	O(n)	O(logn)
Contains	O(n)	O(logn)
isEmpty	O(1)	
size	O(n) -- If we calculate on demand O(1) – if we maintain a counter at each tree	
getValue	O(1)	

getRoot *



Sorting with a BST

traverse (left)
print (root)
traverse (right)



1, 3, 4, 5, 6, 7, 8 ← sorted!

- Insert all values
- Read in an ***in-order*** traversal.
- **Time Complexity**
  - Average Sort: $O(n \log n) + O(n) = O(n \log n)$ ⇐
  - Worst Case Sort: $O(n^2) + O(N) = O(n^2)$

in-order + N = Time complexity of BSTsort

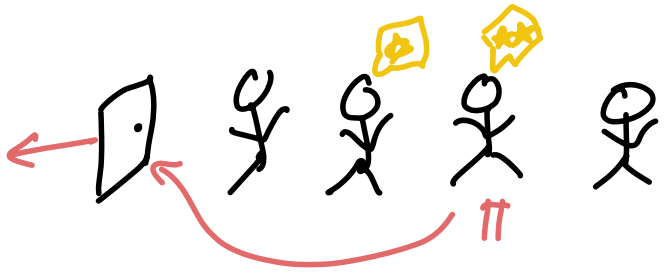
AVG: $n + O(\log n) = O(n \log N)$ (insert)

WC: $n * O(N) = O(n^2)$ (items, insert)

① Add items
+
② In order traversal

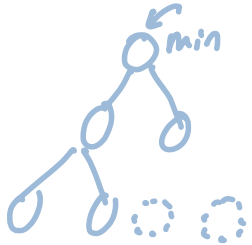
Priority Queue

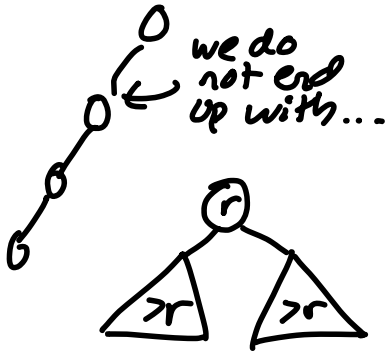
- Queue, except value dequeued should have the lowest priority value.



alt: max

Minimum Binary Tree

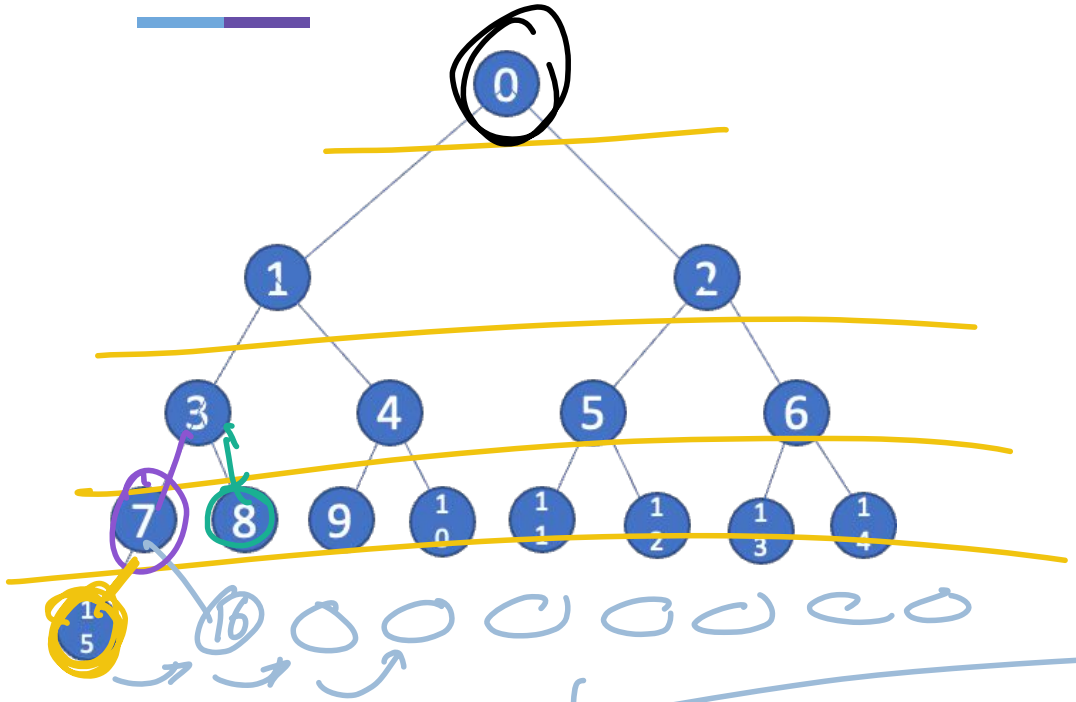


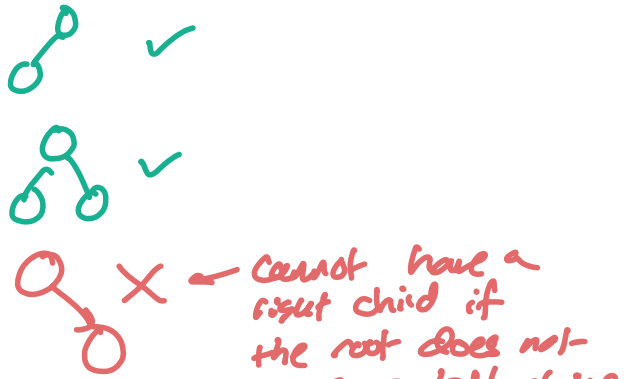


- Binary Trees where given ANY root r in the tree:
  - Every value in the left and right subtree > root's value.
  - The height of the left and right subtree differ by a maximum of 1.
- **Enqueue:** O(logN)
- **Dequeue:** O(logN)

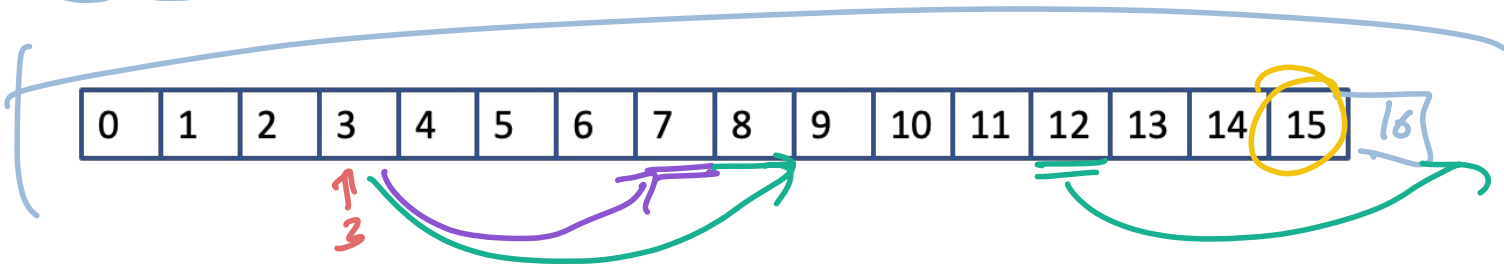
Minimum Binary Heap

- Since our trees always grow along the bottom, our trees are always full.
- Only missing spaces are on the bottom right-hand side.
- So, we can represent our tree as a ***list***.





left: 3*2+1=7
right 3*2+2=8



Minimum Binary Heap (MBT as a List) Invariants

- Indexes
  - “left” at (i*2)+1
  - “right” at (i*2)+2
- If left index is >= size of list, then parent is a leaf.
- If left = size-1, then parent has left, but no right.

Minimum Binary Heap (MBT as a List) Invariants

peek() → findMin()

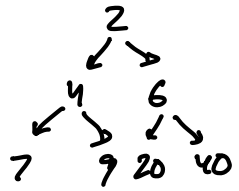
- Get (worst case): O(1)
- Insert (avg case): O(1) – worst case: *O(logN)*
- Remove (avg + worst case): *O(logN)*

C++ not includes memory reallocation

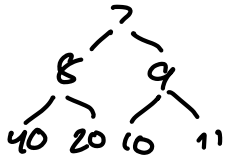
Very efficient!

Insert Practice

① Add 8 to the following:



② Add 1 to the following

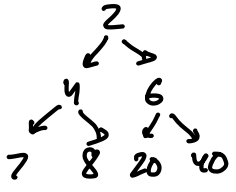


③ Add 3 to the following:

[4, 6, 8, 9, 11, 10, 12]

Delete Practice

① Delete 2 from the following:

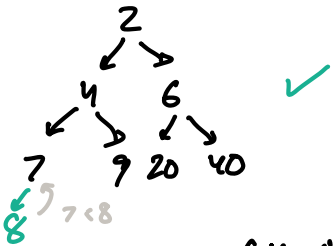


② Delete 4 from the following:

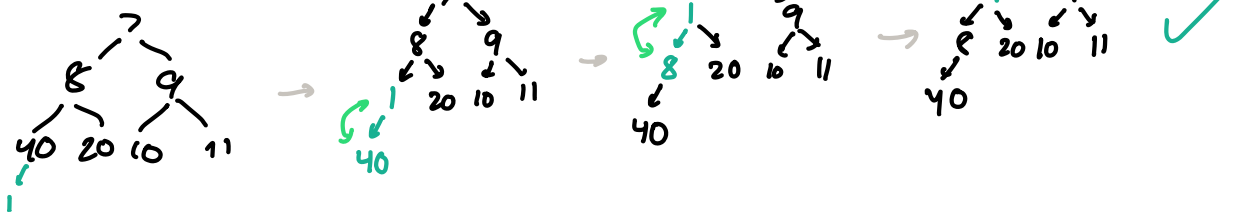
[4, 6, 8, 9, 11, 10, 12]

Insert Practice

① Add 8 to the following:

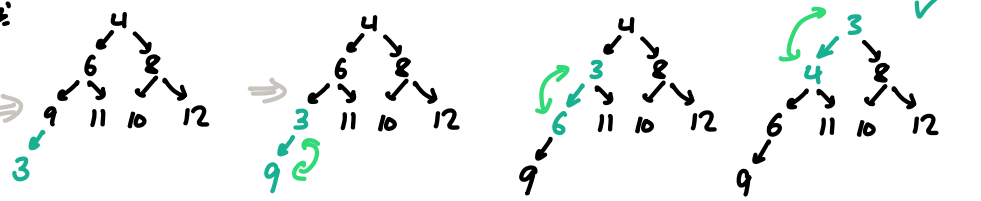


② Add 1 to the following



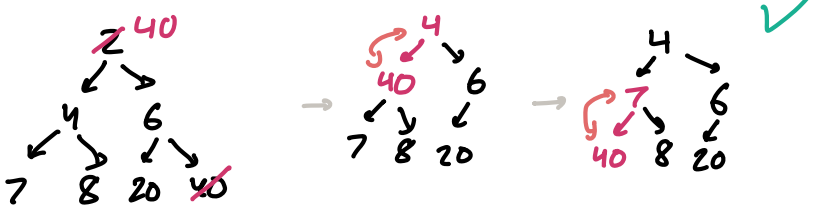
③ Add 3 to the following:

[4, 6, 8, 9, 11, 10, 12]



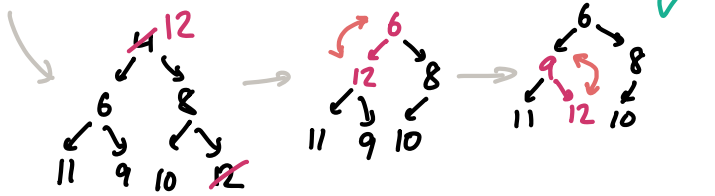
Delete Practice

① Delete 2 from the following:



② Delete 4 from the following:

[4, 6, 8, 11, 9, 10, 12]

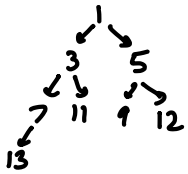


Enqueue to MBH

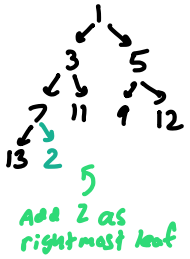
① Add item as the right-most leaf in the tree.

② "Bubble up"
- Keep swapping up the enqueued item with its parent node if the parent node is larger than the enqueued item (which breaks the invariant)

Ex Enqueue 2 to:

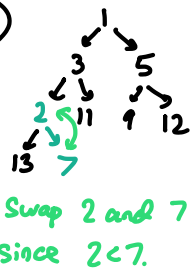


1)



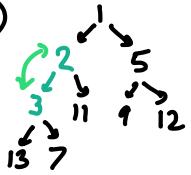
Add 2 as rightmost leaf

2)

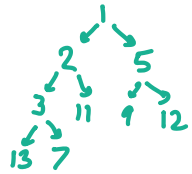


Swap 2 and 7 since 2<7.

3)



Swap 2 and 3 since 2<3.



Tree is valid, so we are done!

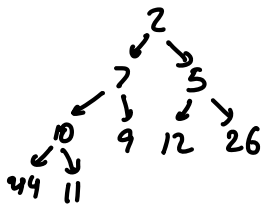
Dequeue from MBH

* we only ever dequeue the root (element with smallest priority value).

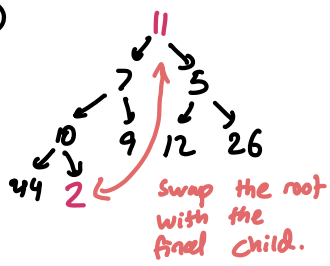
① Swap the final item in the tree (leaf) with the root. Then, delete the minimum (now at the end of the tree).

② "Bubble Down"
- Swap the root with the minimum of its two children. Repeat until the root is no longer larger than either of its children.

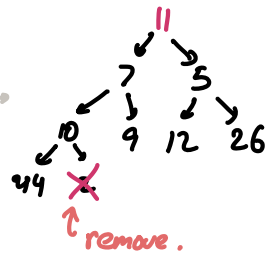
Ex Dequeue from



1)

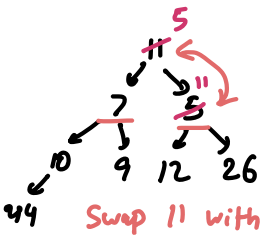


Swap the root with the final child.

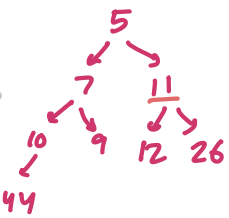


remove.

2)



Swap 11 with 5 (the smallest child node)



11<12 and 11<26, so we are done.