2018-03-08 Priority Queues

Thursday, March 8, 2018 8:54 AM

Hackathon: lumberhacks.org

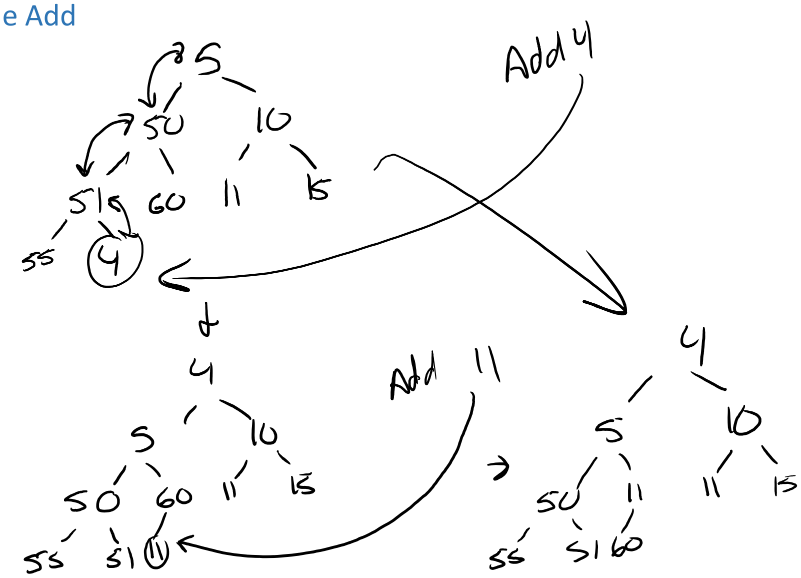
Binary Heaps

- Rules:
  - A complete binary tree
  - All things below a current node are "less important" than that node

Adding an element to a binary heap

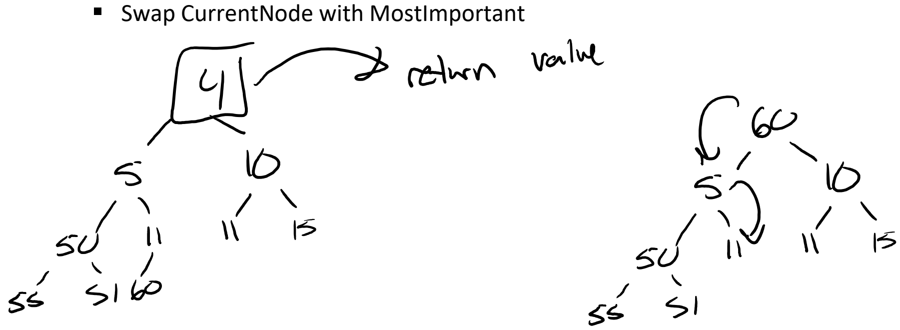
- The new item goes to the bottom-right most empty slot
  - Maintains completeness rule
- Let CurrentNode = this newly inserted node
- Now make 2nd rule true:
  - WHILE CurrentNode is more important than its parent, swap with parent

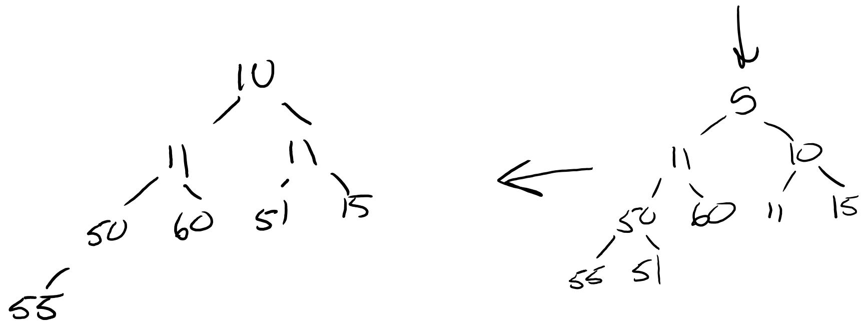
Example Add



Removing (dequeue) from a binary heap

- The item to remove is the top of the tree.
- There is now a hole at the top of the tree. This needs replacing. Replace with bottom-right most element in tree.
  - Ensures adherence to completeness rule
- Now, the root must "roll down" into a valid place (maintain 2nd binary heap rule)
  - Let CurrentNode = root
  - WHILE root is less important than at least one child
    - Let MostImportant = More Important(current->left, current->right)
    - Swap CurrentNode with MostImportant

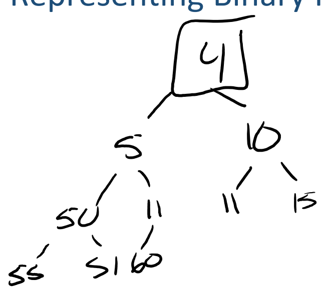




Algorithmic Efficiency of a Binary Heap

- Idea: maintain a sorted vector, dequeue from the front
  - Enqueue: Find correct place (binary search) + shift (N moves) O(N)
  - Dequeue: N shifts O(N)
  - FindTop: O(1)
- Better idea: maintain reverse sorted
  - Enqueue still O(N)
  - Dequeue O(1)
  - FindTop: O(1)
- Other idea: use a AVL Tree
  - Enqueue: Log(N)
  - Dequeue: Log(N)
  - FindTop: Log(N)
- Binary Heap
  - Enqueue: Log(N)
  - Dequeue: Log(N)
  - FindTop: O(1)

Representing Binary Heaps using a vector



0-based heap

4	5	10	50	11	11	15	55	51	60
0	1	2	3	4	5	6	7	8	9

Left child = 2 * i + 1
Right child = 2 * i + 2
Parent = floor((i - 1) / 2)

1-based heap

	4	5	10	50	11	11	15	55	51	60
0	1	2	3	4	5	6	7	8	9	10

Left child = 2 * i
Right child = 2 * i + 1
Parent = floor(i / 2)

Recap: why use an array over a linked list

- Allows us to quickly find parent
- Allows us to quickly find bottom-right most element for enqueue and dequeue
- Complete trees store very efficiently into an array
  - LL: 3 units of memory per node in tree

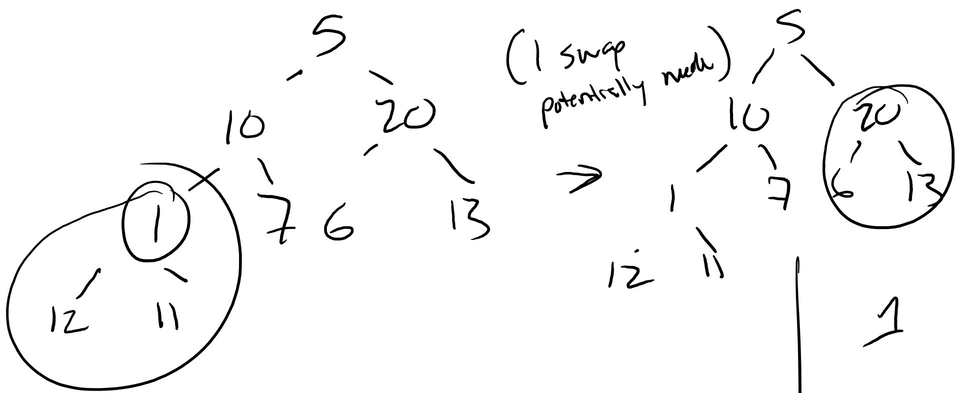
- ○ Complete tree in vector: 1 unit of memory per node in tree

Converting an array into a binary heap

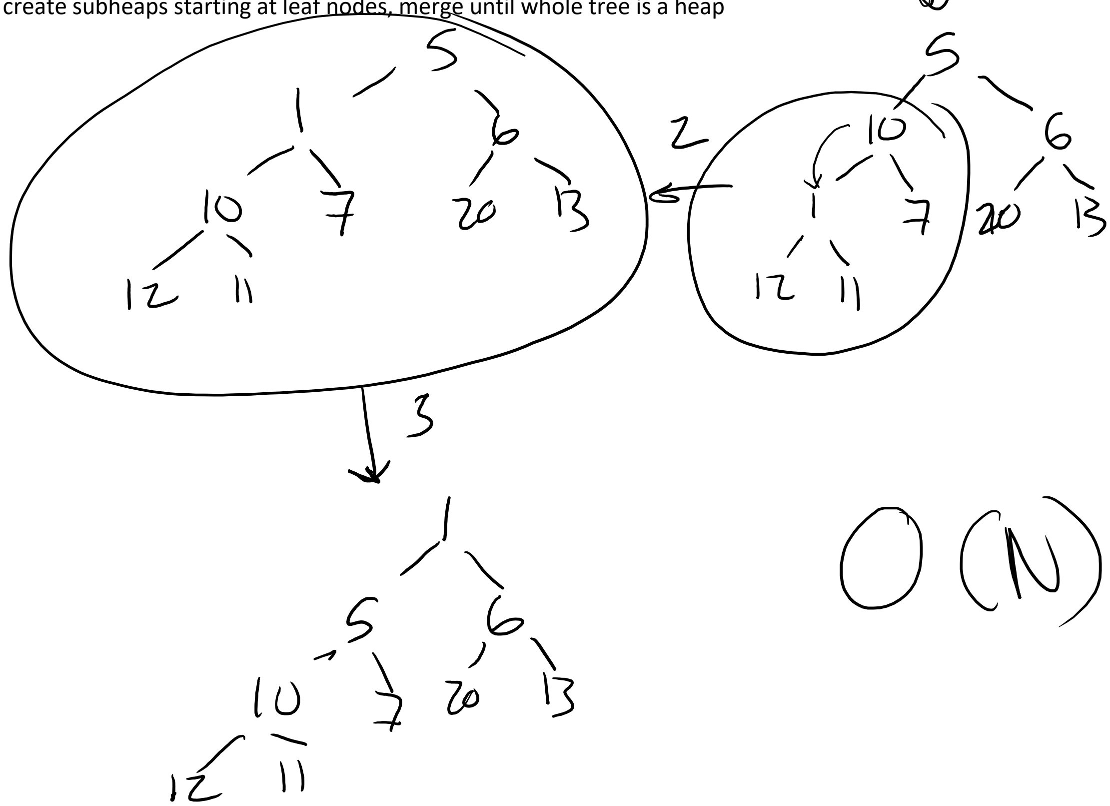
- Naïve: For each item in the array, add to the heap O(N*LogN)
- More clever implementation (build heap)

0	1	2	3	4	5	6	7	8
5	10	20	1	7	6	13	12	11

- Pretend that this is already a heap (but it's actually not)



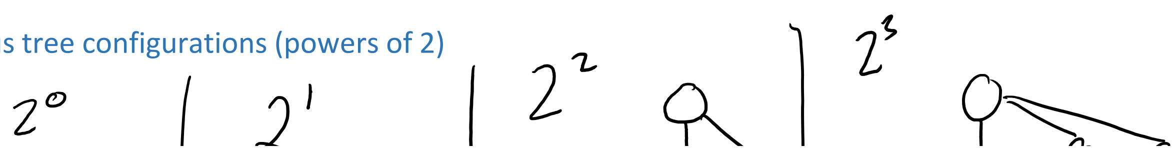
Idea: create subheaps starting at leaf nodes, merge until whole tree is a heap



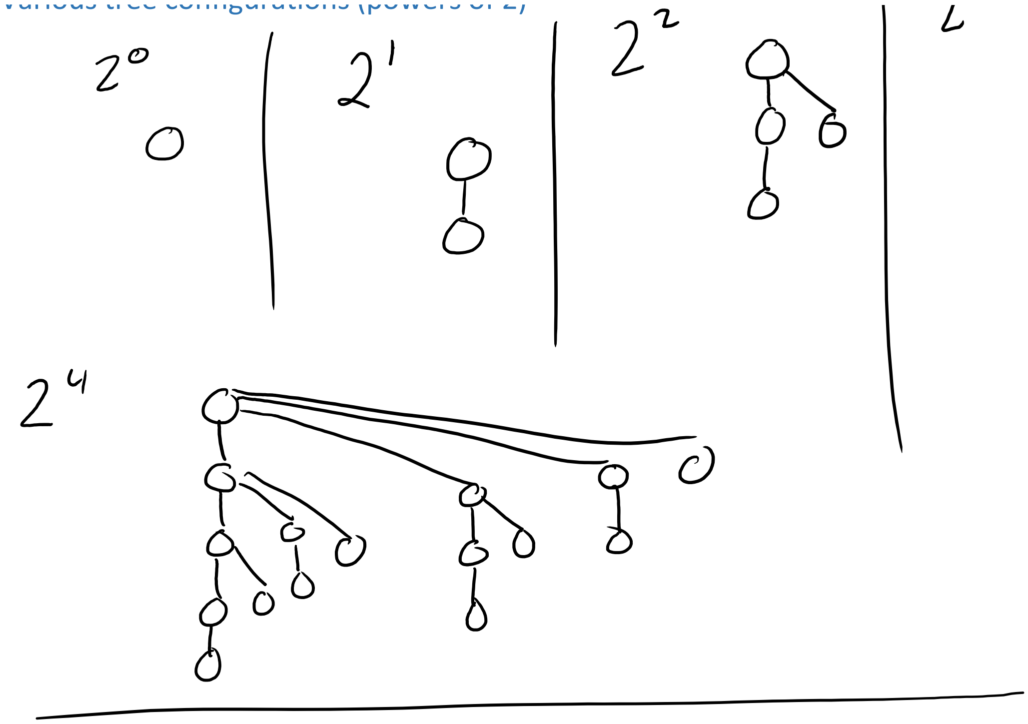
Binomial Heaps

- Binomial heaps are a forest of trees.
- Each tree in the forest has a unique size
- Sizes based on powers of 2

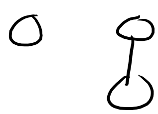
Various tree configurations (powers of 2)



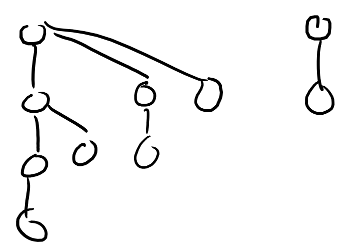
Various tree configurations (powers of 2)



Heap of size 3



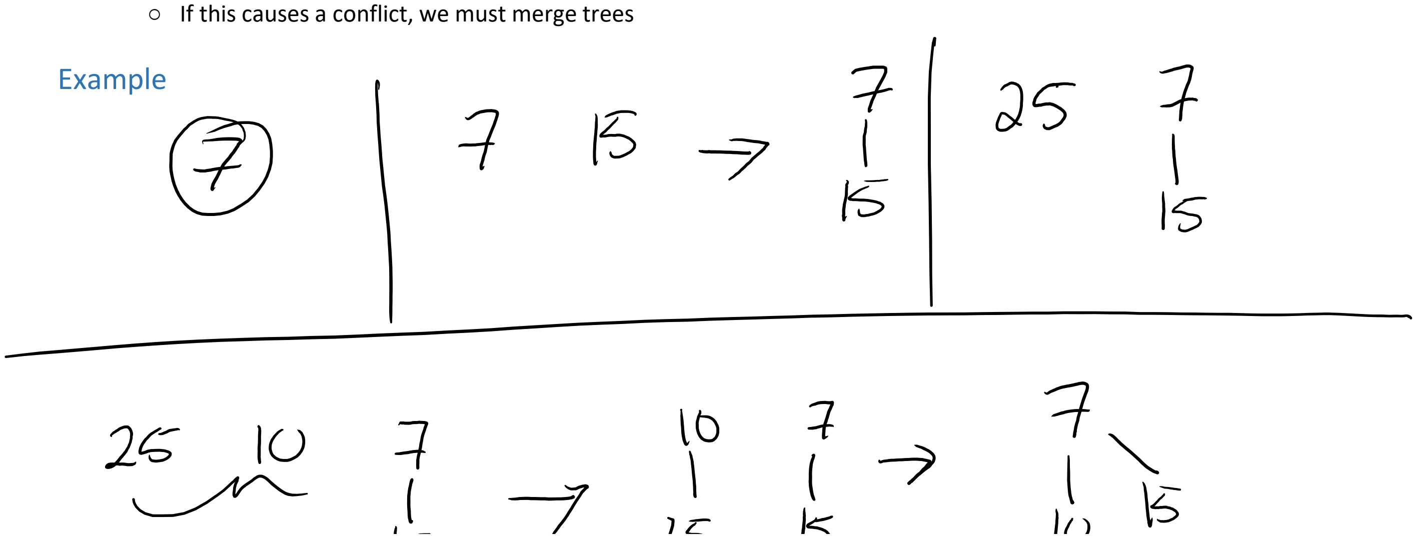
Heap of size 10

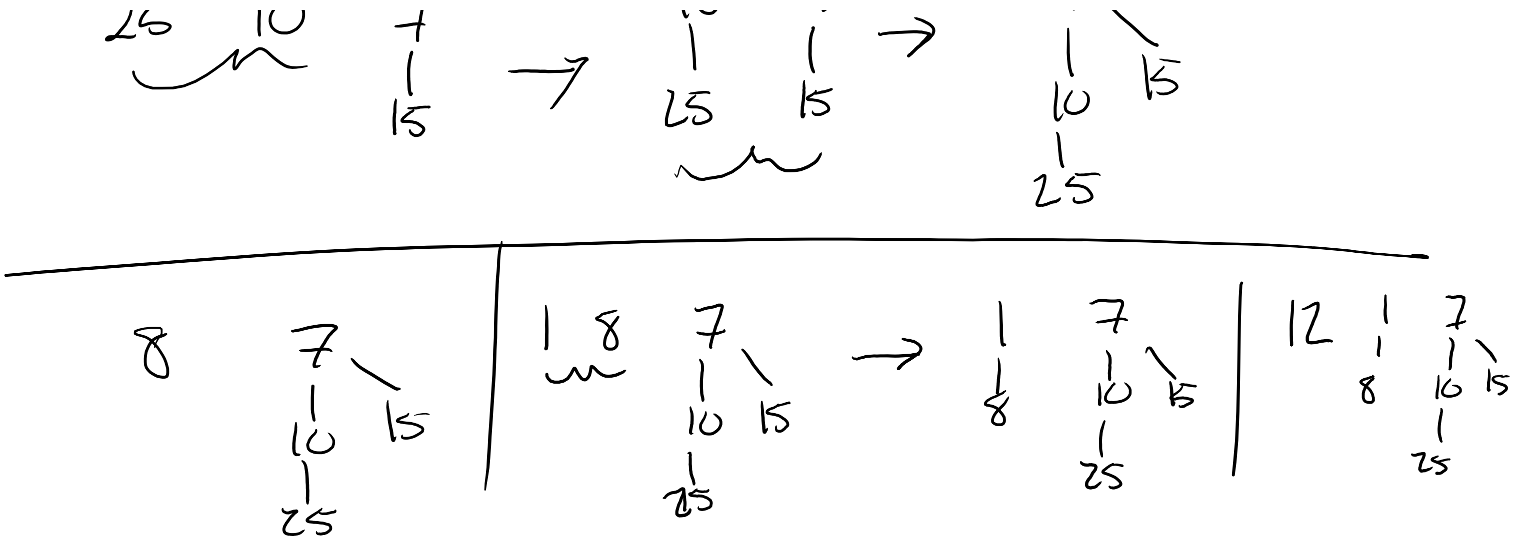


Rules for a binomial heap

- There can only be one tree of a size in a forest
- All items below a given node must be less important
- Enqueue a new value as a tree of size 1
  - If this causes a conflict, we must merge trees
- Dequeues remove the most important root node in the forest
  - If this causes a conflict, we must merge trees

Example





Dequeue