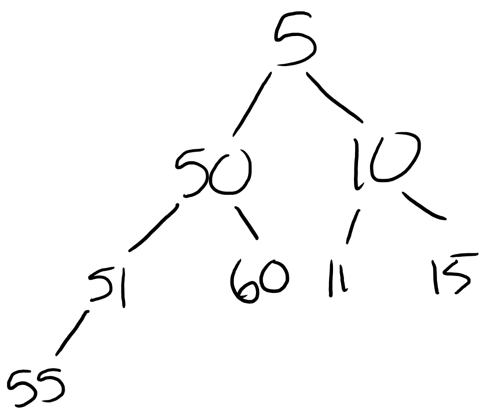
2018-10-02 Priority Queues

Tuesday, October 2, 2018   2:59 PM

- Unlike normal queues, items don't come out based on when they are inserted.
- It's possible for an item to never come out of a PQ
- The most important item always comes out first in a PQ
  - [min queue; class default] - The smallest thing comes out first
  - [max queue] - The biggest thing comes out first
- A priority queue is represented using a tree-like structure
- First PQ we will learn is called a binary heap
  - A binary heap is a binary tree with two rules:
    - The tree must be complete
    - (recursive) A node's parent is more "important" than the node

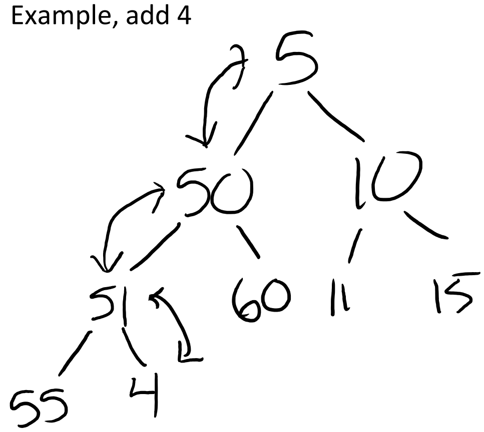
Example Min-Heap

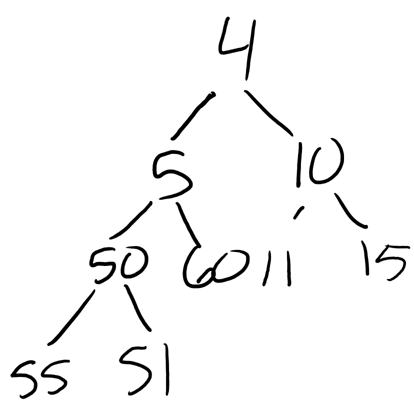


Inserting an item into a binary min-heap

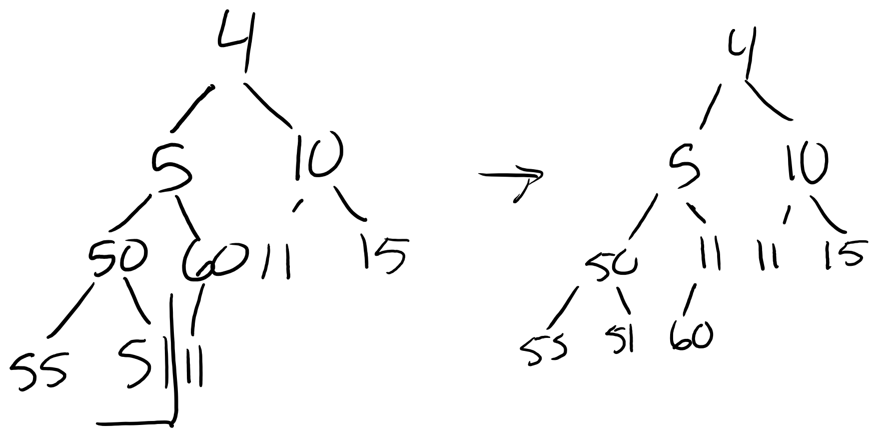
1. Insert the new item at the bottom of the tree such that completeness is maintained.
2. While the new value is more important than its parent, swap value with parent (recursive)

Example, add 4



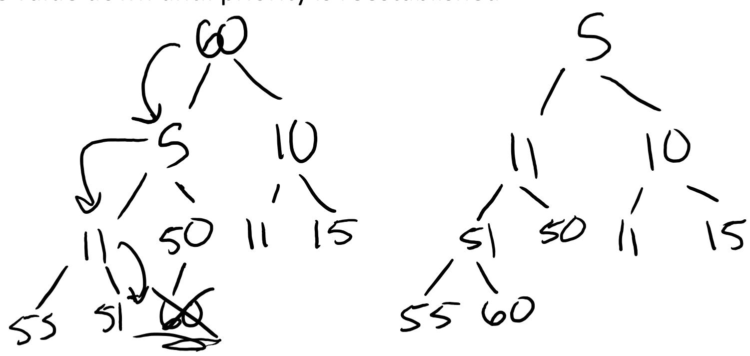


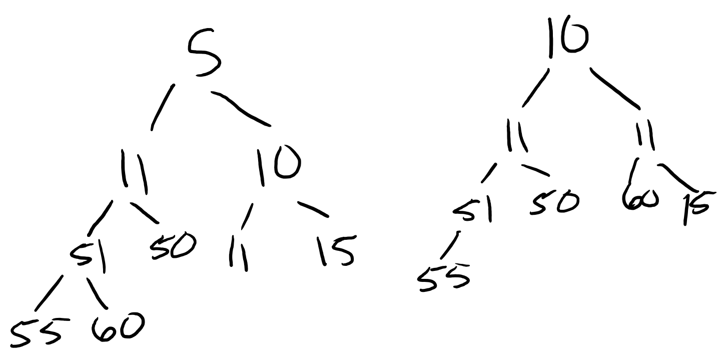
Add 11 to this tree



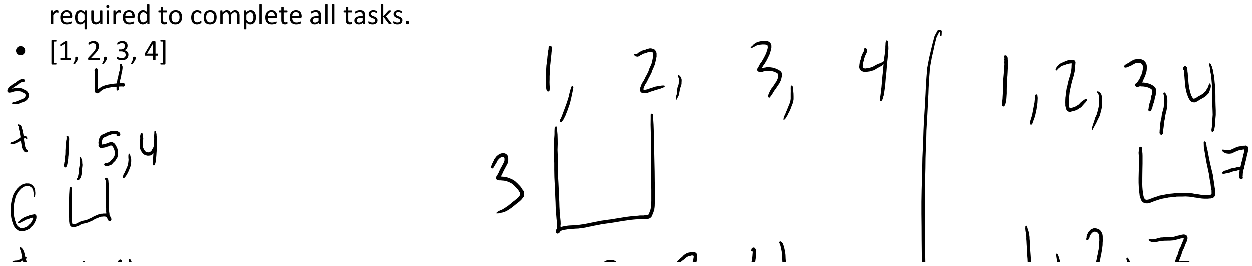
Removing (dequeue) from a binary heap

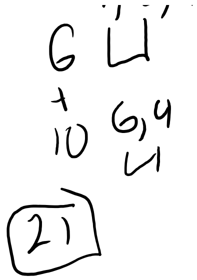
- The item to remove is the root.
- Conceptually, we have a hole at the top of our tree.
- Replace with value in tree such that completeness is maintained
- Percolate value down until priority is reestablished

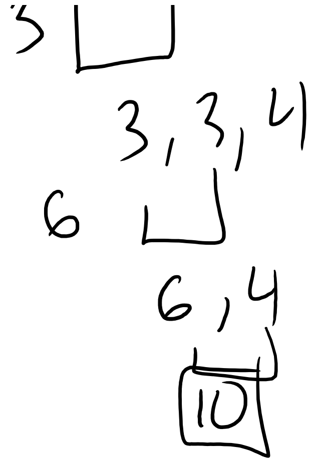


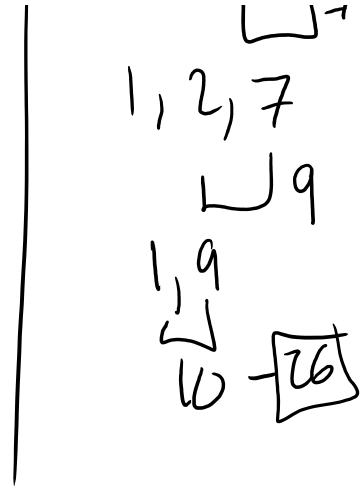


- Assume we have a list of tasks (array of integers). The number represents the time it takes to complete the task. Given a set of tasks, determine the least amount of time required to complete all tasks.
- [1, 2, 3, 4]



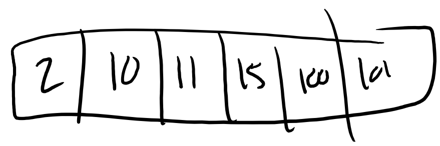






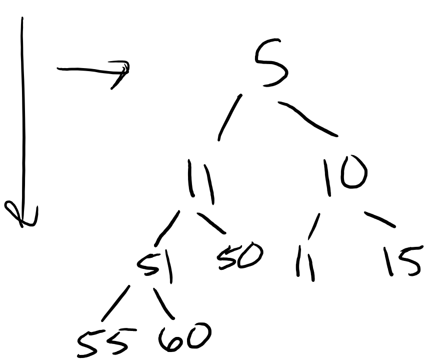
Alg for priority queue:
For each item in list: add to PQ
Total = 0
While pq.size() > 1
    V1 = pq.pop()
    V2 = pq.pop()
    Pq.push(v1 + v2)
    Total += v1+v2
Cout << total;

Algorithmic Efficiency



- PQs allows us to efficiently find the most important element
- Using a vector:
  - Enqueue: O(N)
  - Dequeue: O(N)
  - FindTop: O(1)
- Use AVL Tree
  - Enqueue: Log(N)
  - Dequeue: Log(N)
  - FindTop: Log(N)
- Binary Heap
  - Enqueue: Log(N)
  - Dequeue: Log(N)
  - FindTop: O(1)

Representing a Binary Heap using a vector



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

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

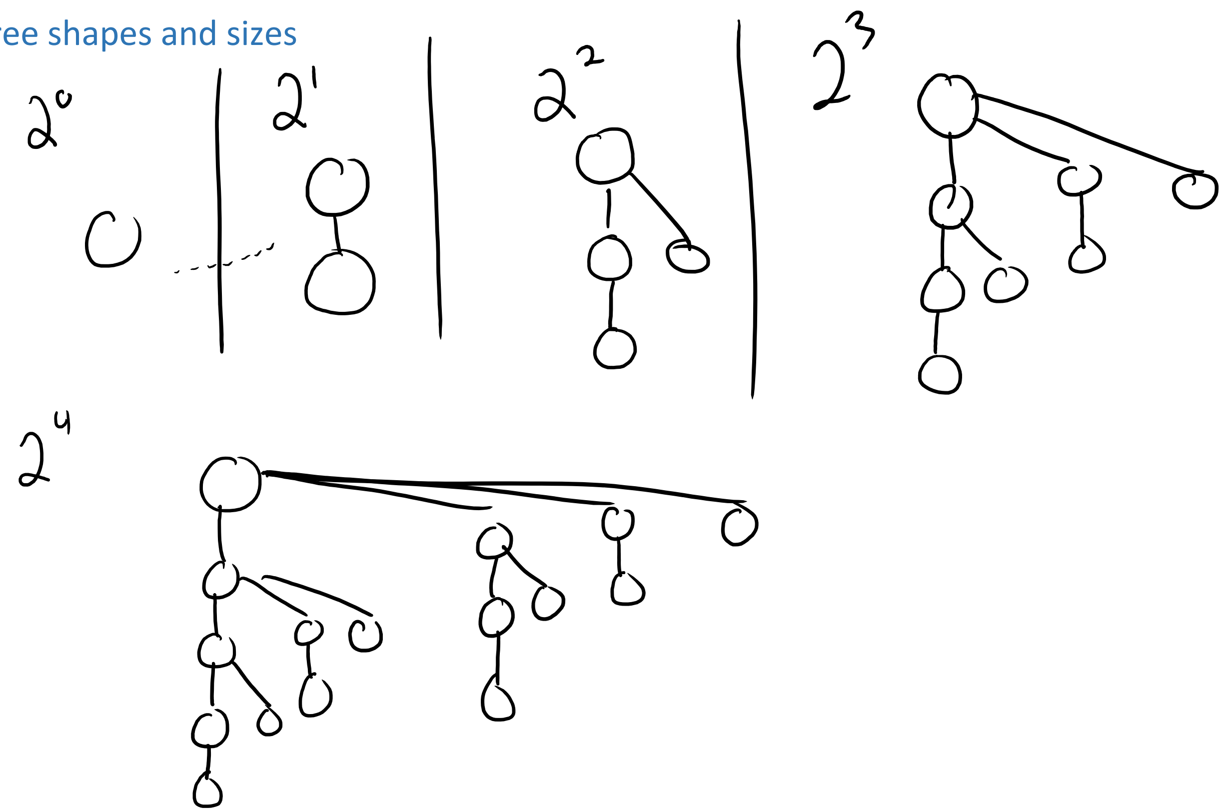
Recap: Why use a vector instead of LL for binary heap

- Vector-based implementations allow for easy lookup of parent
- Vector-based implementations allow us to quickly find bottom-right most element for enqueue / dequeue
- Complete trees can be efficiently stored inside a vector
    - 3 Units of memory per node in LL
    - 1 unit of memory for vector

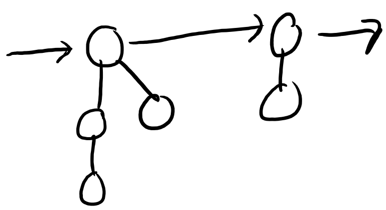
Binomial Heap

- Binomial heaps are comprised of a forest of trees
- Each tree in the forest has a unique size and shape
- Each size must be a power of two. And the shape is defined recursively.

Tree shapes and sizes



Representing a heap of size 6



Adding an element to a heap

- Add a new single node into the tree
- If this violates the 1 tree for each size rule, merge the conflicting sizes
    - Repeat until all unique.