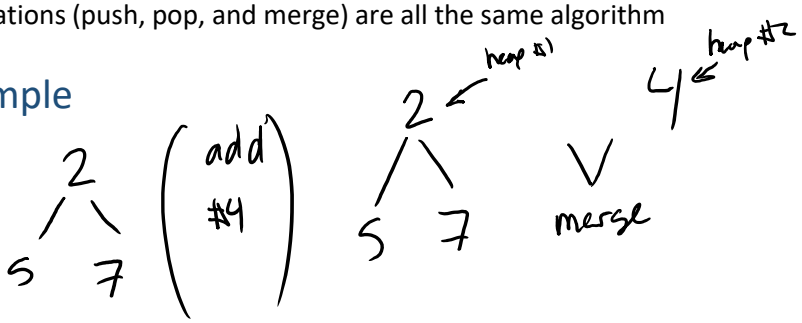
2018-03-20 Priority Queues - Skew Heap

Tuesday, March 20, 2018 8:58 AM

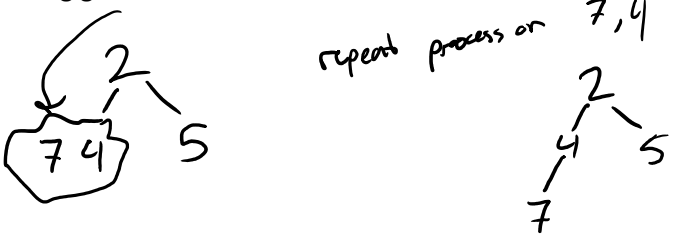
Skew Heap

- Remember that a binary heap uses a vector to represent the heap-tree. This required because:
  - We need to get parent information
  - We need to be able to quickly access the bottom-right most node in the tree
- Skew heaps are a linked-list based implementation of a priority queue
  - Binary heaps don't work well with LL, so we have to come up with a new implementation that satisfies priority queue rules
- Skew heaps are:
  - LL-based
  - Not guaranteed to be complete
  - Not guaranteed to be balanced
  - Have very good heap merging time
  - Have on average LogN runtime
- All operations (push, pop, and merge) are all the same algorithm

Basic Example



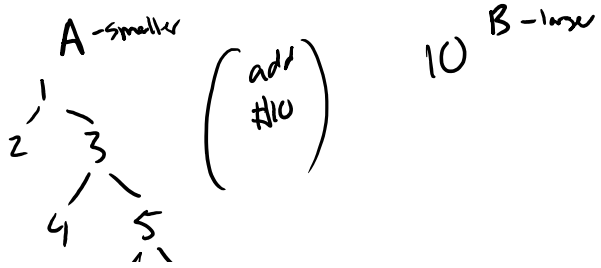
Examine both heaps, let the most important thing stay the top and make the less important thing go somewhere beneath.

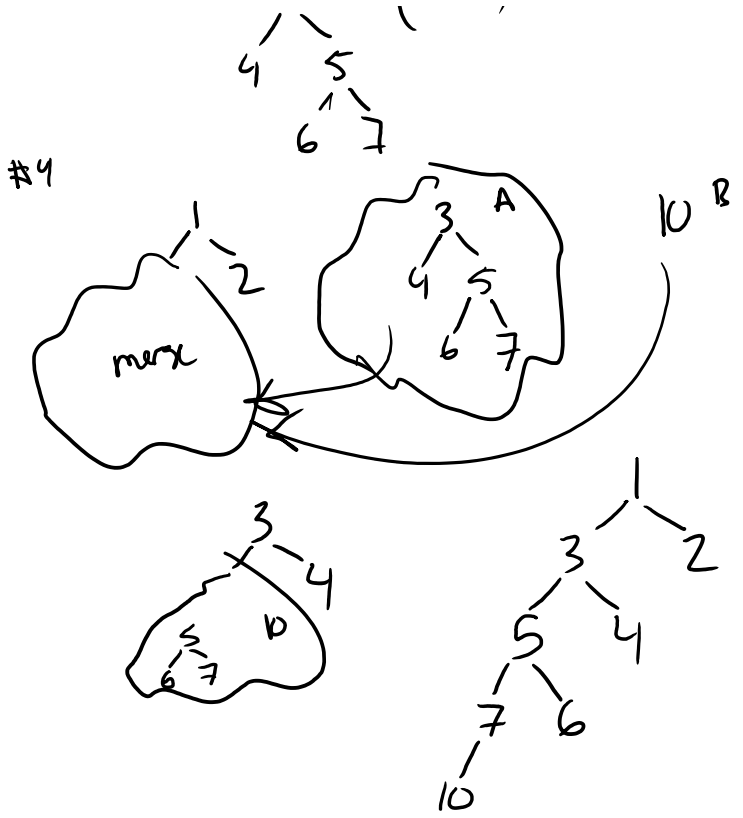


Merge Algorithm (min heap)

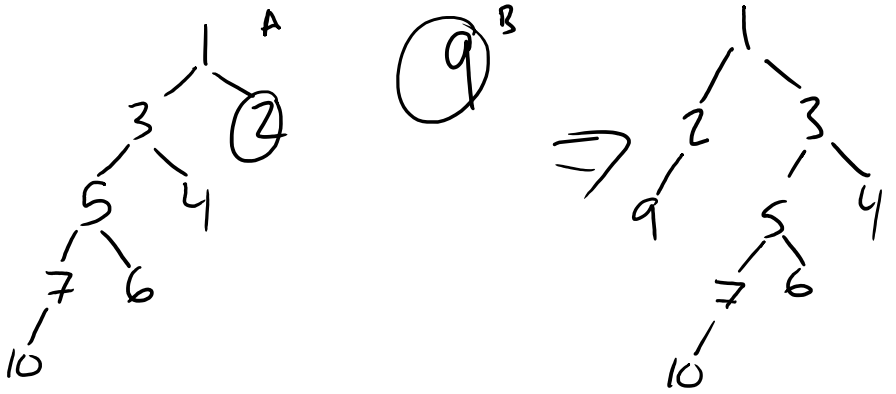
1. Given two heaps A, B
2. Let smaller = smaller value of A and B
3. Let larger = larger value of A and B
4. Set smaller->RightChild = smaller->LeftChild
5. Merge on smaller's old RightChild and larger, set smaller->LeftChild = result of this (recursive, go back to step #1)

Example #2

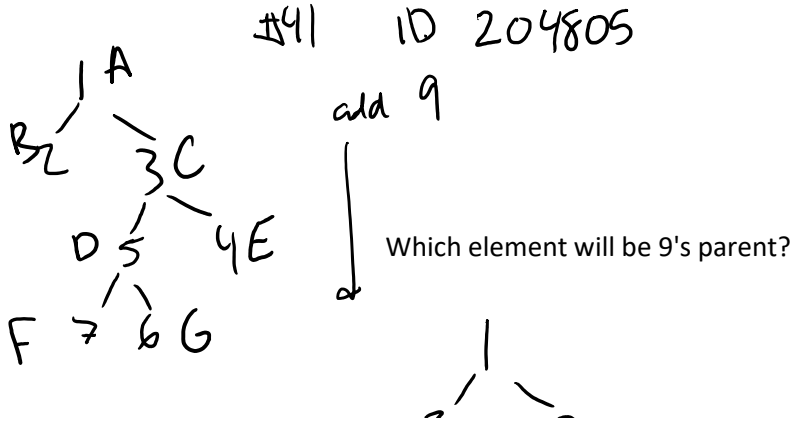




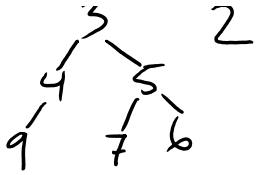
That add took a lot of operations (~5 operations for 8 elements) - NOT logarithmic
After every "slow" skew heap operation comes a really "fast" skew heap operation
The average of a "slow" and "fast" operation = LogN



This time, it took 3 operations for 9 elements. 3 + 5 = 8 for 17 elements (still not true Log, but a lot closer)

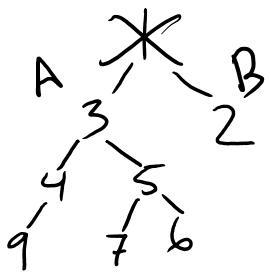


Answer: E

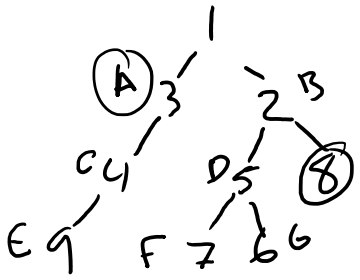


What does a dequeue look like on a skew heap?

- Remove top element, merge left and right subtrees using same algorithm

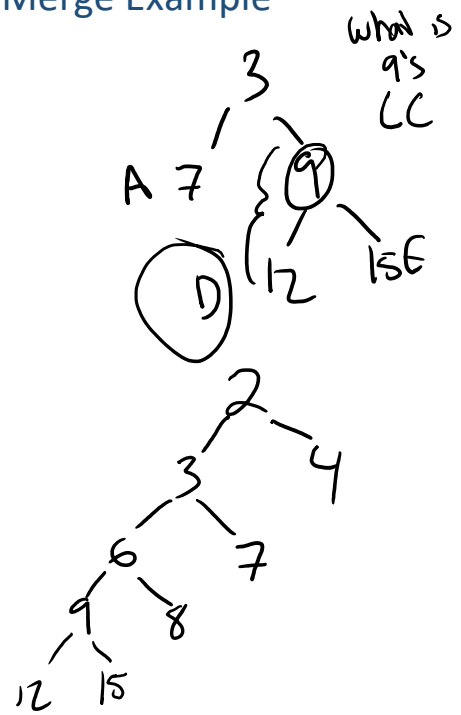


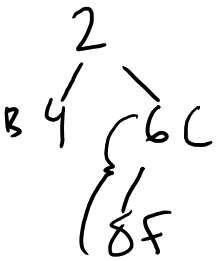
Dequeue Example

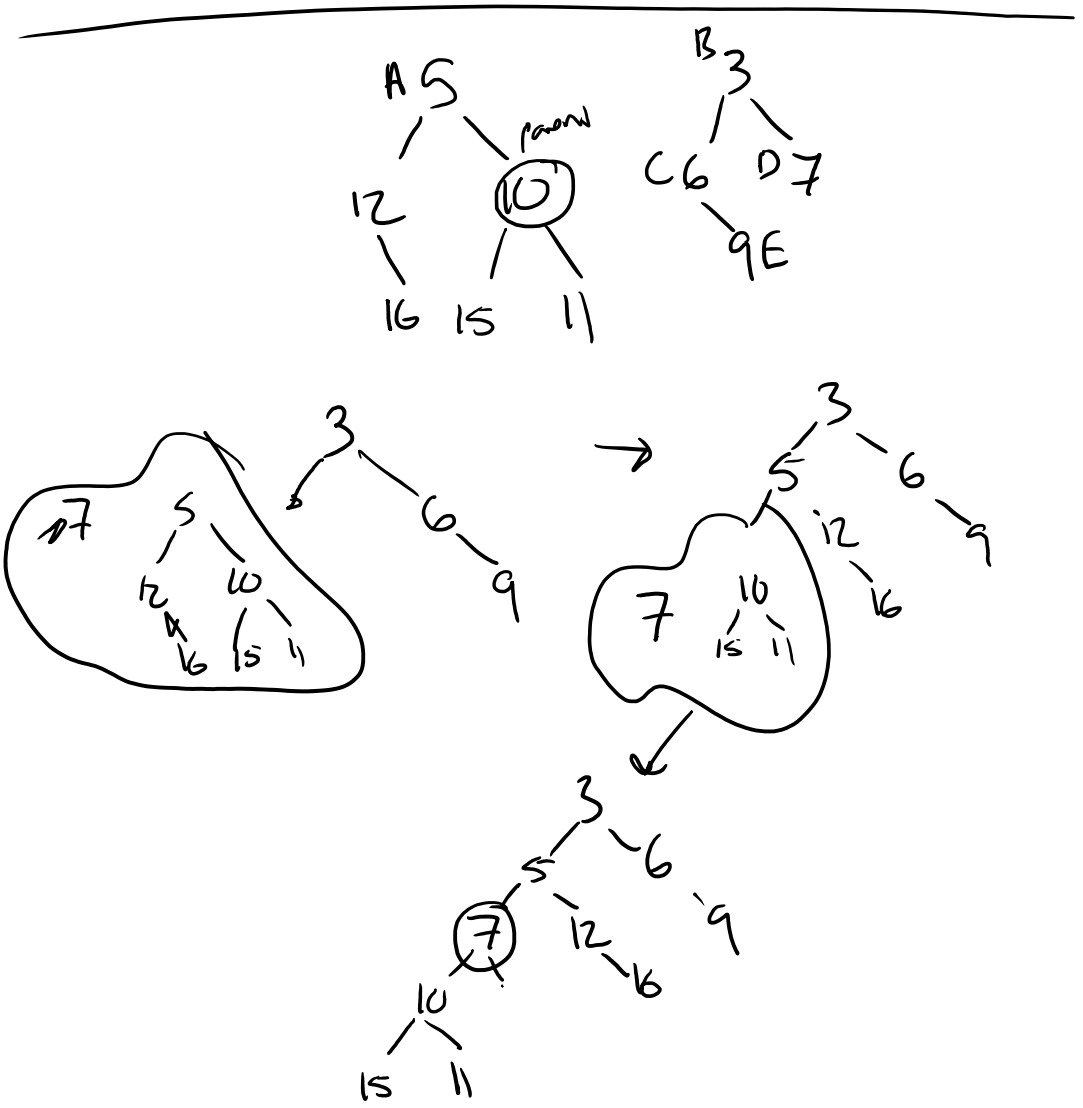


what is 8's parent after dequeue

Merge Example



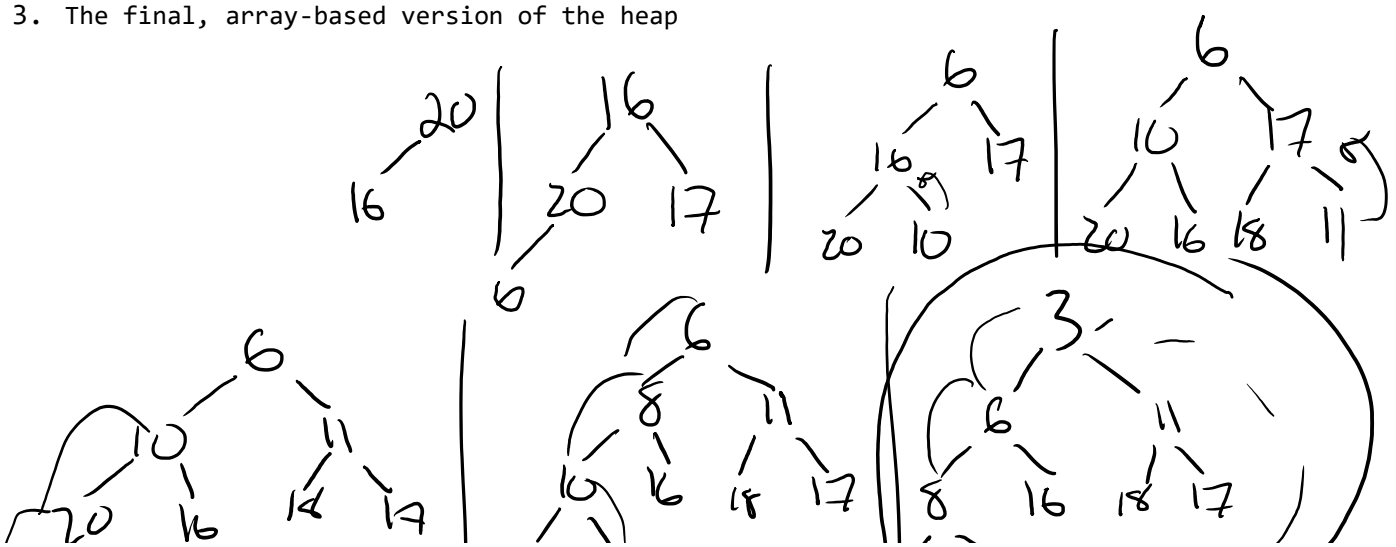


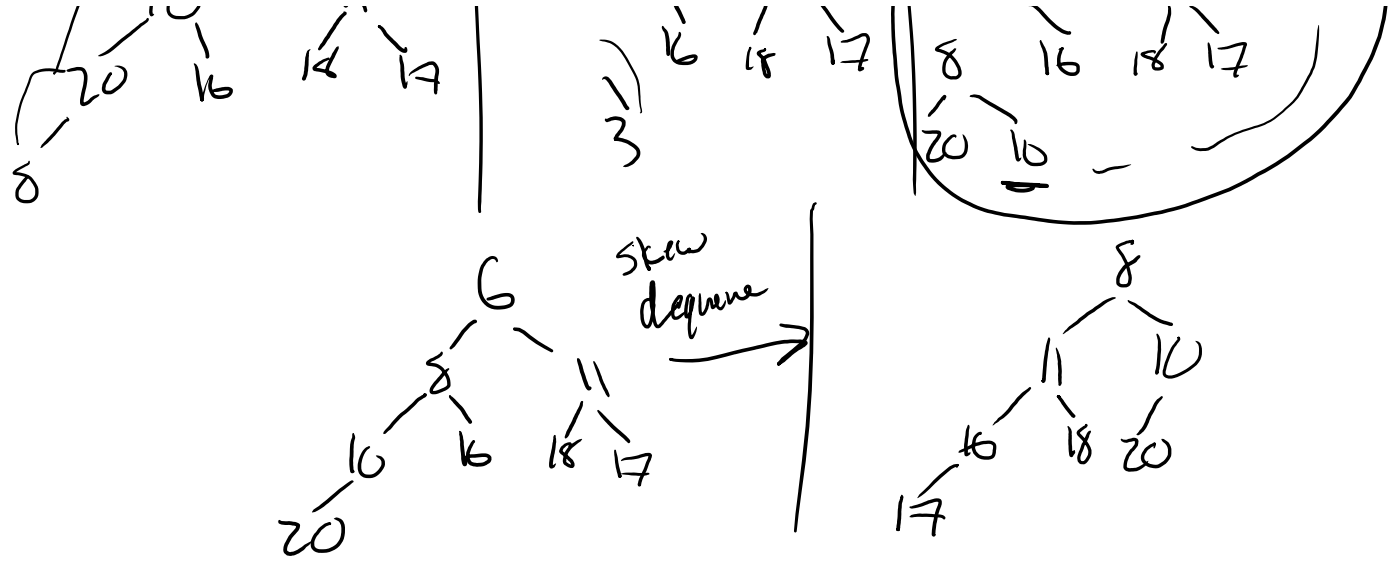


Heap Handout

1. Starting with an empty heap, show the following.

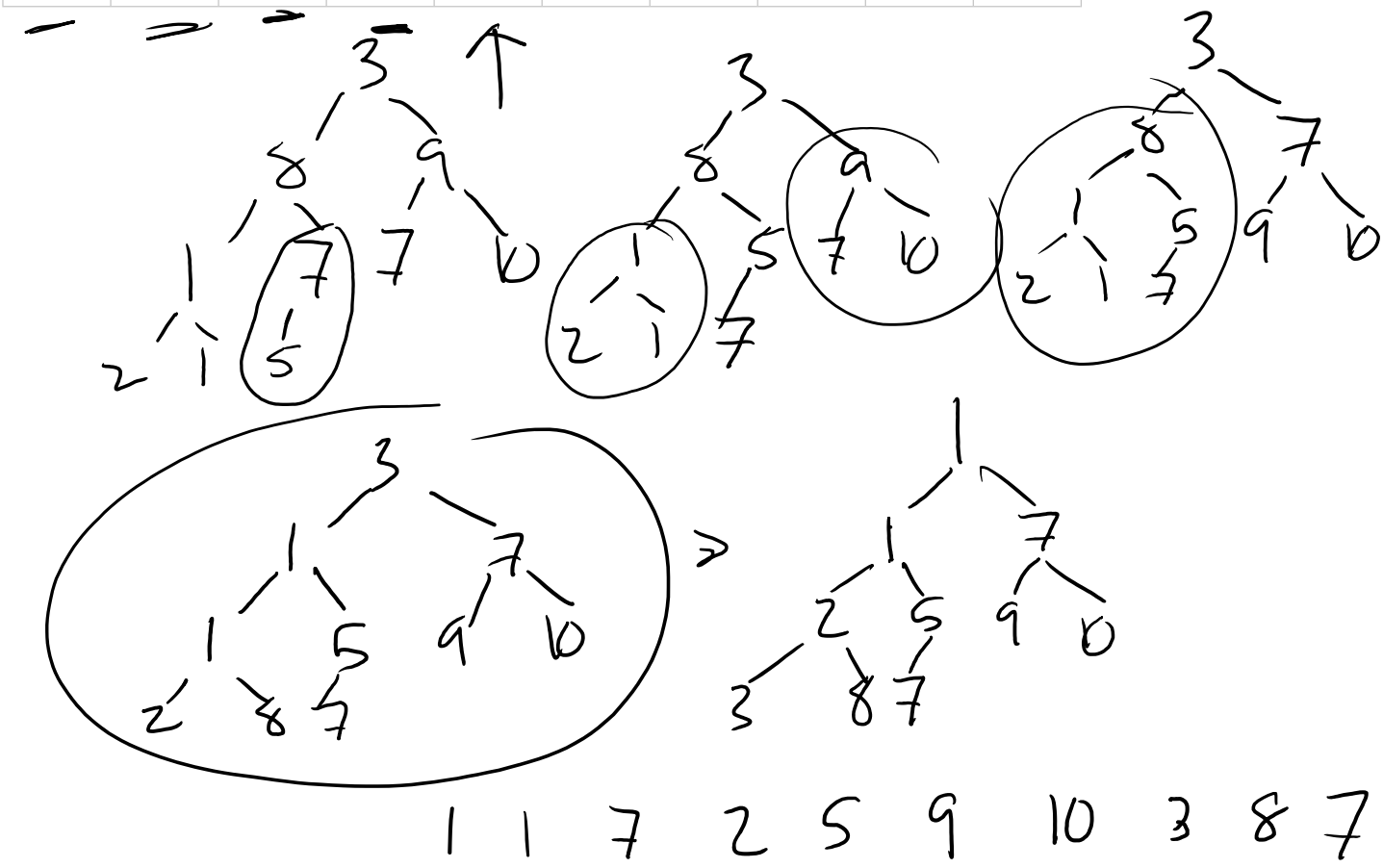
1. The state of the heap after adding in the values: 20, 16, 17, 6, 10, 18, 11, 8, 3
2. The state of the heap after two Dequeue() operations
3. The final, array-based version of the heap



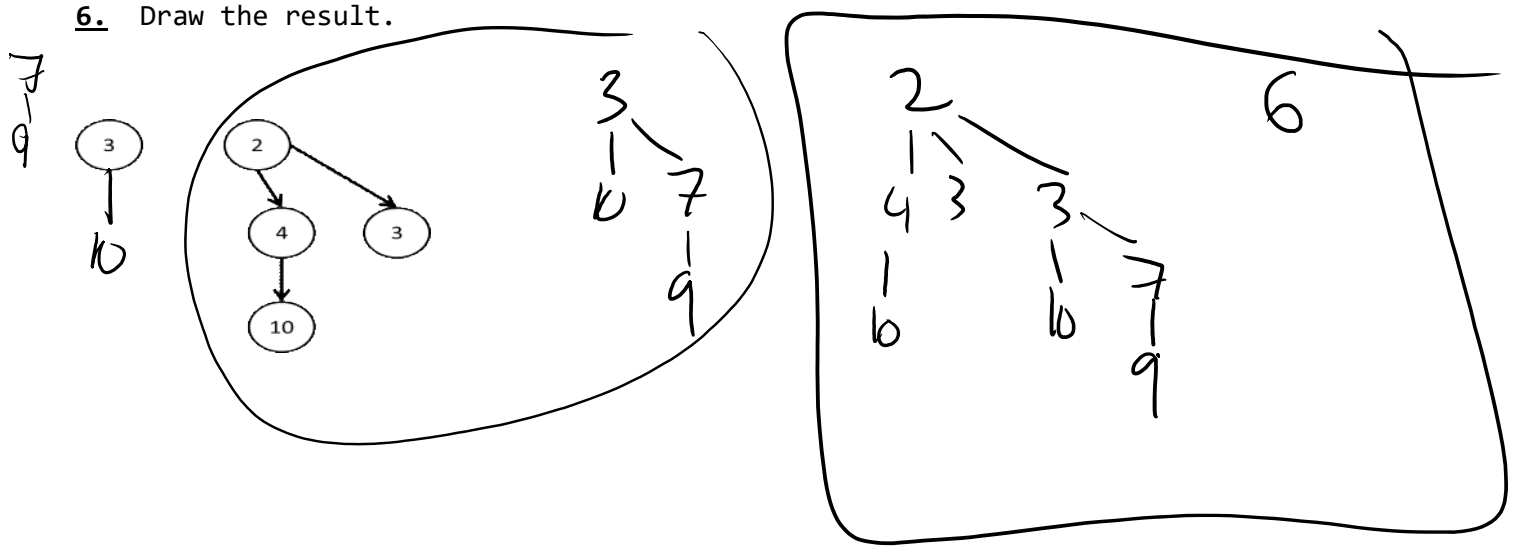


2. Perform a "build heap" operation on the following array:

3	8	9	1	7	7	10	2	1	5



3. Starting with the supplied binomial heap, add the values **10, 9, 7, 3, and 6.** Draw the result.



4. Perform 3 dequeue operations on the following binomial heap: