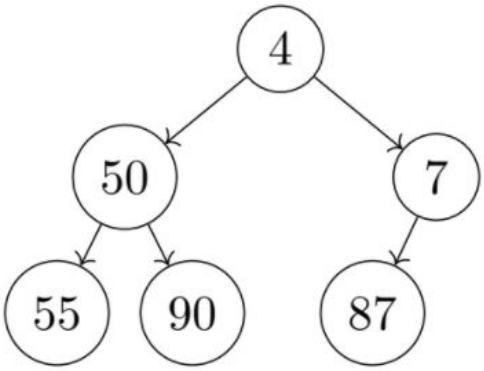
- This lab covers Heaps. Some of these questions are taken from Homework #10.
- You may want to refer to the text and your lecture notes during lab as you solve the problems.
- When approaching the problems, think before you code. It is good practice and generally helpful to lay out possible solutions for yourself.
- You should write test code to try out your solutions.
- You must stay for the duration of the lab. If you finish early, you can help other students to complete the lab. If you don't finish by the end of the lab, it is recommended that you complete the lab on your own time.
- Your TAs are available to answer your questions in lab, during office hours, and on Piazza.

Vitamins (maximum 30 minutes)

---

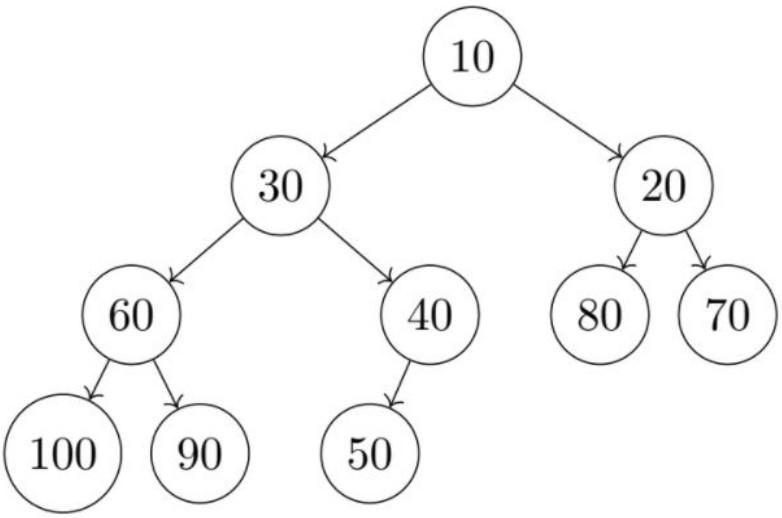
1. Given the following heap:



Complete the following operations, show how the heap looks (both in the array representation and the tree representation) after each operation:

1. Insert 2
2. Delete min
3. Insert 6
4. Insert 8
5. Delete min
6. Insert 52
7. Delete min

2. Given the following heap:



Complete the following operations, show how the heap looks (both in the array representation and the tree representation) after each operation:

1. Insert 25
2. Delete min
3. Delete min

Coding

In this section, it is strongly recommended that you solve the problem on paper before writing code.

1.
    a. Implement the ***FIFO queue ADT*** with the following methods using only a priority queue and one additional integer as data members.

        - `q.enqueue(elem)`
        - `q.dequeue()`
        - `q.first()`
        - `len(q)`
        - `q.is_empty())`

    b. Professor Idle suggests the following solution to part a: Whenever an element is inserted into the queue, it is assigned a priority that is equal to the current size of the queue.

        Does such a strategy result in *FIFO* semantics? Prove that it is so or provide a counterexample.

2. Add the following method to the `ArrayMinHeap` class:

    ```
    def find_less_than_or_equal_to(self, k)
    ```

    This method is given an integer `k`. When called, it creates and returns a list containing all the priorities in the heap that are less than or equal to `k`.

    For example, if `h` is the heap you started with in question 1, the call: `h.find_less_than_or_equal_to(11)`, could return: `[1, 7, 3, 11, 5, 9]`.

    Implementation requirements: Your method should run in time proportional to the size of the returned list, and it should **not** modify the heap.

Extra: In class you saw how heaps represented using an array can be useful when implementing a **Priority Queue**. Now, implement a heap that is represented using a `LinkedBinaryTree`.

> You should use our in-class implementation of the `LinkedBinaryTree`. You should **not** use any other data structure in your solution. You will need to implement all of the methods defined in the Priority Queue ADT. The runtimes should be identical to that of the implementation with the heap.
>
> Before implementing on your computer, conceptually think about how you would implement the priority queue. Be sure to also thoroughly test your code. Write several test cases of each function you implement.