HEAP EXERCISES

In this mini-workbook, we'll get you up and running with heaps. Heaps (or similarly priority queues) are a great utility data structure for certain types of problems, so knowing how they work and how to implement them is key.

In the guide, we will quickly highlight the key concepts that you need to know for your interview. If you're unfamiliar with any of these, I highly recommend taking some time to review them.

Then, I've provided you with a series of exercises to help you get up and running quickly. I recommend setting aside 2-3 days to work through these exercises. Simply start from the top and work through them as much as you have time.

Make sure to track the time you spend here in your Interview Prep Tracker. You can categorize this as "Study" time.

For these exercises, we have complete code samples and templates available here.

TABLE OF CONTENTS

KEY HEAP TERMINOLOGY

- **Min Heap**
  A heap that maintains sorted order from smallest to largest. This heap allows access of the minimum value in the heap in constant time.

- **Max Heap**
  A heap that maintains sorted order from largest to smallest. This heap allows access of the maximum value in the heap in constant time.

- **Priority Queue**
  A priority queue is an abstract data type that functions as a sorted queue. It allows you to push items and pop the "highest priority" item (eg. the largest item). Often priority queues are implemented using a heap, but as an abstract data type, there are many other ways to implement them.

KEY HEAP PATTERNS

- **Accessing and Modifying Values**
  At their core, we can do 3 things with the data in a heap: `insert`, `peek`, and `pop`:
  - `insert` - Add a value to the heap while maintaining the order of the data
  - `peek` - Get the largest (or smallest) value
  - `pop` - Remove the largest (or smallest) value

- **Maintaining data in sorted order**
  Generally heaps are useful whenever we want to maintain the order of our data as we add or remove values since it can do this very efficiently. We can also access the largest (or smallest) value whenever we want in constant time.

  However, the one big caveat is that heaps generally only let us remove the largest (or smallest) value rather than removing arbitrary values. If we need the ability to remove arbitrary values, we may want to maintain a sorted list instead.

HEAP EXERCISES

Exercise Set #1 - Accessing and Modifying Values

1. Implement a min heap with the following methods:
    a. Constructor
    b. `insert(int n)` - Insert a value into the heap
    c. `peek()` - Return the largest value in the heap
    d. `pop()` - Remove the largest value from the heap and return it
    e. `size()` - Return the number of items in the heap
    f. `toString()` - Convert the heap data to a string

2. Given an array of integers, determine whether the array represents a valid heap

    eg.
    ```
    arr = [1,3,2,4,6,5,0]
    isValid(arr) = false

    arr = [1,3,2,6,5]
    isValid(arr) = true
    ```

    What is the time and space complexity of your solution?

Exercise Set #2 - Maintaining Data in Sorted Order

1. Given a list of integers, use a heap to find the largest value in the list

   eg.
   `findMax([1,2,3,4,5]) = 5`

   What is the time and space complexity of your solution?

2. Given a list of integers, use a heap to sort the list

   eg.
   `heapSort([5,4,3,2,1]) = [1,2,3,4,5]`

   What is the time and space complexity of your solution?

3. Find the k-th largest element in a stream of integers (Full Problem Definition)

   What is the time and space complexity of your solution?

4. Find the k closest points to the origin (Full Problem Definition)

   What is the time and space complexity of your solution?

Exercise Set #3 - Bonus: Popular Tricky Heap Problems

1. Find the median of a data stream (Full Problem Definition)

   What is the time and space complexity of your solution?

2. Merge k sorted lists (Full Problem Definition)

   What is the time and space complexity of your solution?