Introduction: Heaps

Recall that a heap is a tree-like data structure that satisfies the heap-order property.

Definition (Heap-Order Property). A tree has the heap-order property if for any parent node $P$ with a child $C$, the key of $P$ is ordered with respect to the child $C$.

Notice that this definition immediately implies that the root must contain either the “maximum” or the “minimum” of the ordering relationship that we define, since the root is the parent of every other node. Specializing this definition to keys that act like natural numbers, or keys that implement Comparable, we have our classic min-heap and max-heap. This basic idea is really powerful, as the heap data structure maintains the “maximum” or “minimum” element whenever we add to it or remove from it. If there are $n$ keys, removing and inserting elements takes time $O(\log n)$.

We say that the heap-order property induces a partial order over its elements. Intuitively, a partial order means that not every pair of elements are related. Even though we know that 17 is less than 23, when we insert these numbers into the heap, we cannot determine which number is “greater” solely by its position in the heap. Compare this to inserting both elements in a binary search tree, where we can determine the order by examining their relative positions. We say that the binary search tree establishes a total order.

For some problems, it is enough to have just a partial ordering. Later in the course when you learn more about sorting algorithms, you will find that you can get a stronger, total ordering at the cost of a larger runtime. In fact, building a heap has the nice property that it only takes time linear in the number of elements (can you figure out how?)! We get the maximum or minimum in linear time, and the partial ordering!

Binary heaps

A binary heap is a binary tree, but with the heap-order property. A binary heap is most commonly implemented by flattening a tree in level order into an array. It satisfies the following property:

Definition (Shape Property). A tree has the heap-shape property if the tree is a complete binary tree. That is, all levels of the tree are fully filled, except for possibly the last, where all nodes are as far left as possible.

With the shape property, we can easily index into a binary heap, since we will not have to worry about “gaps.”

Testing your understanding

Answer the following questions regarding implementations of binary heaps.

Problem 1. Consider the following array:

```
null  6  7  9  15  13  17  14  20  16  23  18  19  37  42  ···
```

Let this array be the underlying storage for a binary heap. Is this a max-heap or a min-heap? What is the parent of the key 17? What is the left child of 17? The right child?
Problem 2. Is it possible for the following array to be the underlying array for a heap?

```
null 64 42 37 19 21 38 43 23 17  · · ·
```

If it cannot be the underlying array for a binary heap, what key(s) would you have to change in order to make it a heap?

Problem 3. You are now in the shoes of the Java Virtual Machine, and you are tasked with maintaining the min-heap property for a binary heap that is represented in the following array:

```
null 1 2 3 4 5 6 7 x · · ·
```

A pesky CIS 121 student has called the `insert` method, which begins by placing the variable `x` into the underlying array at the location indicated above. If `x = 0`, what is the final state of this array after the `insert` method completes?

Problem 4. You are still in the shoes of the Java Virtual Machine, and you have to maintain the min-heap property for a binary heap that is represented in the following array:

```
null — 1 2 3 4 5 6 7 · · ·
```

The same pesky CIS 121 student has called the `removeMin` method, which already removed and returned the value at the location indicated above. What is the final state of this array after you, the JVM, fix the array again so that it has the min-heap property?

Problem 5. Imagine that you are parsing a text corpus, say, containing all the words in Wikipedia. Design an algorithm to find the first word that is the $k$-th longest. You should attain a runtime that is $O(n \log k)$, where $n$ is the number of words.

Problem 6. Consider an indefinitely long stream of unsorted integers. We are interested in knowing the median (in sorted order) at any given time. How would we do this in an efficient manner?