Introduction to Heaps

Sanjit Dandapanthula

January 2, 2024

Warm-up: binary search.

In this packet, we're going to learn about a neat "data structure" called a heap. Data structures are very important because they make certain computations more efficient; they are used ubiquitously in computers, phones, and other devices. Let's start with a warm-up.

Exercise 1. *Suppose we play the following children's game, which many of you have likely played before. I pick a number between 1 and 32 (including the end-points) and you need to guess the number that I chose. For each guess you make, I will tell you whether my number is larger than, smaller than, or equal to your guess. What is the best strategy for you to guess my number quickly? What is the maximum number of guesses that you would need? Hint: if you are confused, try playing this game with your neighbor a few times.*

Exercise 2. *Suppose I choose a number between 1 and 2^n, inclusive of the endpoints. How many guesses will it take before you find my number? Hint: each guess can be chosen to cut the search space in half.*

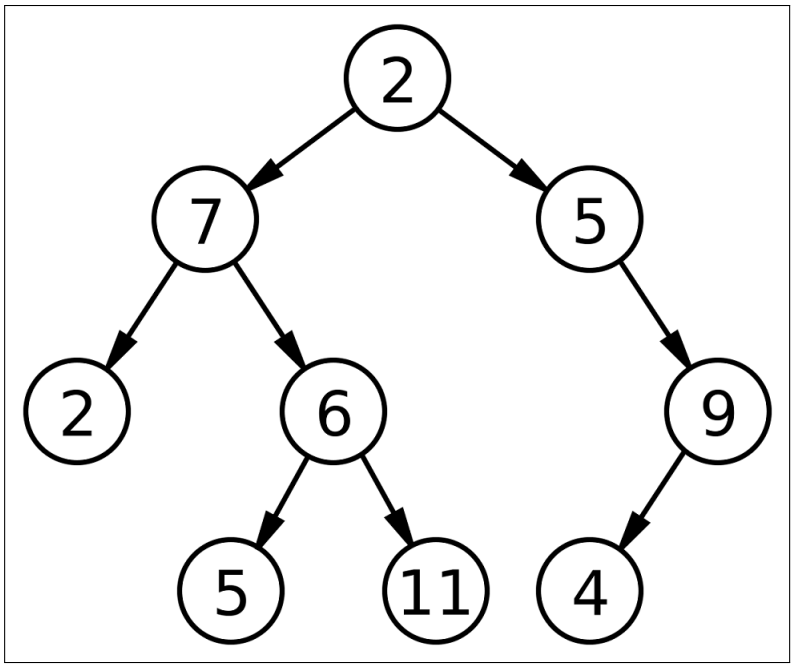
How many times can we cut a number in half before we get to 1? Remember that $b^a = c$ if and only if $a = \log_b(c)$; logarithms are the inverse operation of exponentiation.

Binary trees.

A binary tree is a graph (a set of nodes and edges between them) with the following properties:

- There are no cycles.
- There is a special node, called the root node.
- Each node has at most two children and one parent, except for the root node, which has no parent.
- Each node has a number associated with it (this is the data contained in the node).

The graph is constructed by drawing edges between any two nodes with a parent-child relationship. Here is an example of a binary tree:



A binary tree containing 9 elements.

Exercise 3. *Draw two different binary trees with 7 elements, numbered from 1-7.*

A **complete** binary tree has its elements filled in from left to right, where each level must be filled in before one can move to the next level.

Exercise 4. *Draw a complete binary tree having 20 elements.*

Exercise 5. *How many nodes are in a complete binary tree of depth d (where the depth of the tree is the number of levels)?*

Exercise 6. *Find the depth of a complete binary tree with n nodes.*

Every complete binary tree can be associated uniquely with a list of numbers as follows. Take the list of numbers, and fill in the vertices of the complete binary tree in order (from top to bottom, then from left to right) using the elements of the list.

Exercise 7. *Draw the complete binary tree associated to the list* $[15, 4, 3, 6, 2, 9, 10]$.

Exercise 8. *Suppose we have a zero-indexed list of numbers (the first element is at index 0, the second element is at index 1, etc.), and we associate with it a complete binary tree in the usual way. Consider*

the element at index i in the list. At what index can we find the child of this element in the binary tree? Similarly, at what index can we find the parent of this element? We would like to find a closed-form formula. Hint: try a few examples and look for a pattern.

Heaps.

A **heap** (more specifically, a max-heap) is a complete binary tree where every element is larger than both of its children.

Exercise 9. *Put the following elements into a heap:* $\{15, 4, 3, 6, 2, 9, 10\}$. *Then write the associated list of numbers for this heap (since the heap is, of course, a complete binary tree).*

Exercise 10. *Put the following elements into a heap:* $\{234, 634, 456, 64, 33, 2345, 3546, 76\}$. *Then write the associated list of numbers for this heap (since the heap is, of course, a complete binary tree).*

Exercise 11. *Please convince yourself that the top element of a heap is always the largest one. In addition, the left and right subtrees in a heap are also themselves heaps.*

It should be clear to you at this point that each heap can be represented as a list of numbers and vice versa. This is a simple example of an **isomorphism**, or a perfect identification between two sets of things. From now on, we will treat these representations interchangeably. Consider the following algorithm for inserting an element into a heap (this is called a "heap push"):

1. Put our new element at the end of the heap (this corresponds to putting our new number at the end of the list).

2. As long as the current element is larger than its parent, swap it with its parent (this is called "sifting up").

We expect this algorithm to be fast with the list representation because a) inserting into a list is fast and b) thanks to Exercise 8, it is fast to locate the parent or child of a node given its index.

Exercise 12. *Put the following elements into a heap as before:* $\{234, 634, 456, 64, 33, 2345, 3546, 76\}$. *Then write the sequence of lists (a.k.a. heaps) obtained from applying the above algorithm to inserting the element 60, through the sifting process.*

Exercise 13. *How many operations does it take, in the worst case, to insert an element into a heap containing n elements? An operation consists of either inserting an element at the end of the heap, or swapping two elements of known positions in the heap. Hint: recall that it is fast to find the parent or child of a node given its index, as a result of Exercise 8.*

Exercise 14. *Please come up with an algorithm to delete the top element from a heap (this is called a "heap pop"). Hint: the first step of a heap pop is to swap the top element (at the zeroeth index) with the*

last element of the heap. Hint: the next step will involve sifting the first element down in the tree.

Exercise 15. *Use the algorithm you derived in the previous exercise to delete the root element of your heap created from the elements* $\{234, 634, 456, 64, 33, 2345, 3546, 76\}$. *Show all the steps.*

A **perfect** binary tree is a complete binary tree with its lowest level completely filled in.

Exercise 16. *How many operations does it take, in the worst case, to delete an element from a heap containing n elements? An operation consists of either inserting an element at the end of the heap, or swapping two elements of known positions in the heap. Hint: recall that it is fast to find the parent or child of a node given its index, as a result of Exercise 8.*

Applications.

Exercise 17. *Find an algorithm (using heaps along with a sequence of heap pushes and heap pops) to sort a list of numbers of length n using $C \cdot n \log(n)$ operations, where C is a constant. In fact, it can be shown that this is the best we can do.*

Exercise 18. *Heaps make it easy to find the maximum of a list, so we want to make it easy to find the median. This problem is purposefully vague, but can you use heaps (you will need a max-heap and "min-heap") to implement a data structure so that it is relatively fast to insert elements, relatively fast to delete the median, and extremely fast to find the median?*