

## Readings

- [Lecture Notes Chapter 14: Binary Heaps and Heapsort](#)

## Review: Heaps

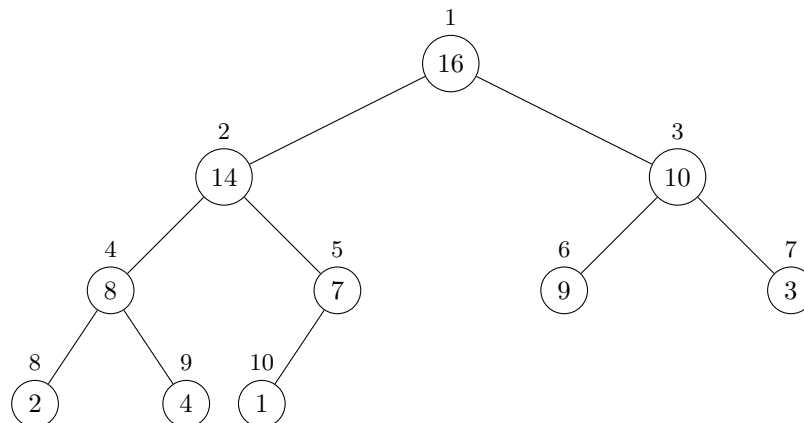
A heap is a tree-like data structure that implements the priority queue abstract data structure (ADT), which allows us to maintain a set of elements, each with an associated key, and select the element with the highest/lowest priority. Heaps satisfy the two following properties:

**Heap Property:** In a max-heap, for each node  $i$ , we have  $A[\text{PARENT}(i)] \geq A[i]$ , so the maximum value is stored at the root. In a min-heap, for each node  $i$ , we have  $A[\text{PARENT}(i)] \leq A[i]$ , so the minimum value is stored at the root.

**Shape Property:** A heap is an almost complete binary tree, meaning that every level of the tree is completely filled except for the last, which must be filled from left to right.

### Implementation Details

Because a heap is an almost complete binary tree, we are able implement it using an array with 1-indexing as shown below:



Index	0	1	2	3	4	5	6	7	8	9	10
Value	null	16	14	10	8	7	9	3	2	4	1

Observe that we can populate the array from left to right by doing a level-order traversal of the tree, where we start from the root and go through each level of the tree from left to right. Additionally, because of the shape property, if the root is stored at index 1 of the array, given a node at index  $i$ , its left child can be found at index  $2i$ , its right child can be found at index  $2i + 1$ , and its parent can be found at index  $\lfloor i/2 \rfloor$ .

## Operations (Max-Heaps)

MAX-HEAPIFY maintains the max-heap property at the node called on, so the entire subtree rooted at the node will now be a max-heap. It assumes the node's left and right subtrees are both valid max-heaps, and then allows the node to "float-down," swapping it with its larger child or terminating if the max-heap property holds. It runs in  $O(h)$  time, where  $h$  is the height of the node, since in the worst case, the node must "float down" to the bottom of the tree. Since the height of any node is upper bounded by  $\log n$ , MAX-HEAPIFY runs in  $O(\log n)$  for any node (though this bound may not be tight for some nodes, which is a property we leverage when analyzing the runtime of BUILD-MAX-HEAP).

BUILD-MAX-HEAP constructs a max-heap from an unsorted array by repeatedly calling MAX-HEAPIFY on nodes from the "bottom-up", starting at the nodes right above the leaves (which by definition are max-heaps!). It runs in  $O(n)$  time; the mathematical proof of this upper-bound can be found [here](#).

EXTRACT-MAX removes and returns the element with the maximum key. We remove the root, replace it with the right-most element in the bottom level/last element in the array, and then call MAX-HEAPIFY on the "new" temporary root to maintain the max-heap property. We perform constant work besides calling MAX-HEAPIFY, so it runs in  $O(\log n)$  time.

INSERT adds an element by first adding it to the end of the array/max-heap, and then allowing it to "float-up" to its correct position by repeatedly swapping it with its parent as necessary to maintain the max-heap property. It runs in  $O(\log n)$  time, since the path it takes while it "floats-up" has length  $O(\log n)$ .

PEEK returns the maximum element in the heap stored at the root. Since we implement a heap with an array, this runs in  $O(1)$  time because we just index into the array.

## Problems

---

### Problem 1

Given a data stream of  $n$  test scores, design an  $O(n \log k)$  time algorithm to find the  $k$ -th highest test score. Since PEFS provides minimal monetary resources, CIS 1210 Staff has limited access to storage space and can only afford you  $O(k)$  space, where  $k \ll n$ .