Tutorial 11: Heaps

April 15th, 2025

Introduction to Heaps

A **heap** is a specialized tree-based data structure that satisfies the heap property:

- **Max-Heap:** For every node i, the key is greater than or equal to the keys of its children.
- **Min-Heap:** For every node i, the key is less than or equal to the keys of its children.

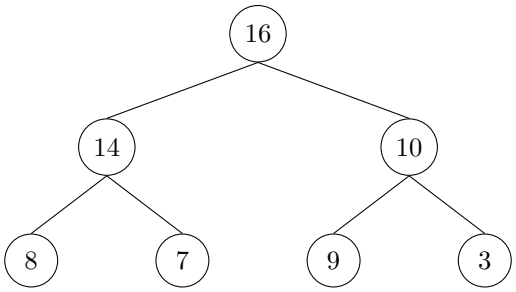
Heaps are typically implemented using arrays for efficiency. The relationships in an array-based heap are:

$$\text{Parent}(i) = \lfloor i/2 \rfloor, \quad \text{Left}(i) = 2i, \quad \text{Right}(i) = 2i + 1$$

Example of a Max-Heap

Consider the max-heap below represented as both a tree and an array:

Tree representation:



Array representation:

[16, 14, 10, 8, 7, 9, 3]

Max-Heap Operations

MAX-HEAPIFY

The MAX-HEAPIFY operation ensures that the subtree rooted at a given index i satisfies the max-heap property. It assumes subtrees rooted at left and right children are already max-heaps.

Algorithm 1 MAX-HEAPIFY($A, i, heap\_size$)

1: $l \leftarrow 2i, \quad r \leftarrow 2i + 1, \quad largest \leftarrow i$
2: **if** $l \leq heap\_size$ and $A[l] > A[largest]$ **then**
3: $\quad largest \leftarrow l$
4: **end if**
5: **if** $r \leq heap\_size$ and $A[r] > A[largest]$ **then**
6: $\quad largest \leftarrow r$
7: **end if**
8: **if** $largest \neq i$ **then**
9: $\quad$ swap $A[i]$ and $A[largest]$
10: $\quad$ MAX-HEAPIFY($A, largest, heap\_size$)
11: **end if**

BUILD-MAX-HEAP

Builds a max-heap from an unordered array by repeatedly calling MAX-HEAPIFY.

Algorithm 2 BUILD-MAX-HEAP(A)

1: $heap\_size \leftarrow length(A)$
2: **for** $i \leftarrow \lfloor heap\_size/2 \rfloor$ down to 1 **do**
3: $\quad$ MAX-HEAPIFY($A, i, heap\_size$)
4: **end for**

HEAP-SORT

Utilizes BUILD-MAX-HEAP to sort an array in ascending order.

Algorithm 3 HEAP-SORT(A)

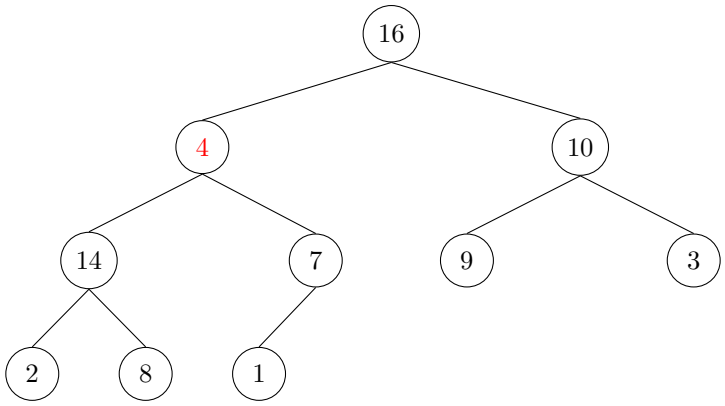
1: BUILD-MAX-HEAP(A)
2: $heap\_size \leftarrow length(A)$
3: **for** $i \leftarrow length(A)$ down to 2 **do**
4: $\quad$ swap $A[1]$ and $A[i]$
5: $\quad heap\_size \leftarrow heap\_size - 1$
6: $\quad$ MAX-HEAPIFY($A, 1, heap\_size$)
7: **end for**

Example: MAX-HEAPIFY

Consider the array (violates heap property at index 2):

$$A = [16,\ 4,\ 10,\ 14,\ 7,\ 9,\ 3,\ 2,\ 8,\ 1]$$

Corresponding tree:



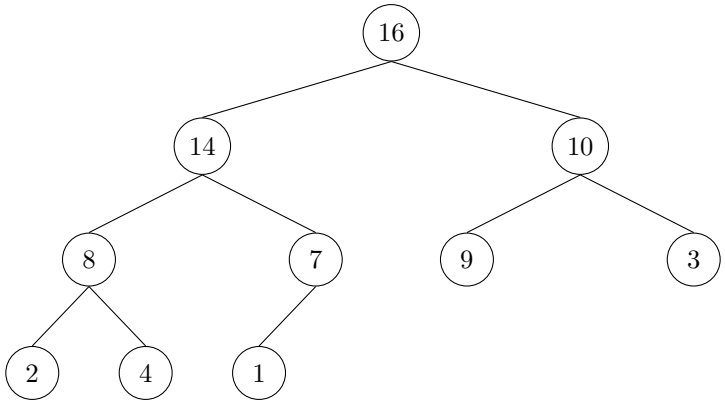
We call MAX-HEAPIFY on node 2 (key=4):
- Node 4 is smaller than its left child (14). Swap them:

$$[16, 14, 10, 4, 7, 9, 3, 2, 8, 1]$$

- Node 4 again violates the heap with its new left child (8). Swap again:

$$[16, 14, 10, 8, 7, 9, 3, 2, 4, 1]$$

Final heap after MAX-HEAPIFY:



Example: HEAP-SORT

Consider the following unsorted array:

$$A = [4, 10, 3, 5, 1]$$

Step 1: Build a Max-Heap
Convert the array into a max-heap by applying `BUILD-MAX-HEAP`:
Initial array/tree:

$$[4, 10, 3, 5, 1]$$

Applying `MAX-HEAPIFY` from bottom-up:
- Start from node $\lfloor n/2 \rfloor = 2$, key=10. Already satisfies max-heap. - Move to node 1, key=4. Swap with larger child (10):

$$[10, 4, 3, 5, 1]$$

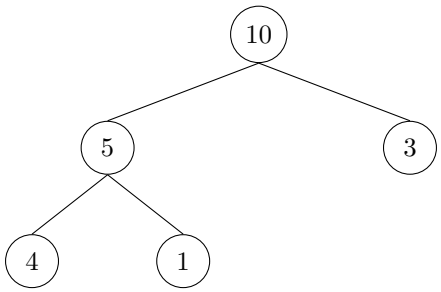
- Node 4 violates heap property again, swap with child 5:

$$[10, 5, 3, 4, 1]$$

Now we have a valid max-heap:

$$[10, 5, 3, 4, 1]$$

Visual representation:



Step 2: Sorting Process (Repeated extraction of max)

Now repeatedly swap the root (maximum) with the last element and reduce heap size:

- **Extract 10**: Swap with last element (1), heap-size=4:

$$[1, 5, 3, 4 \,|\, 10]$$

Apply MAX-HEAPIFY at root (1), swap with 5, then swap with 4:

$$[5, 4, 3, 1 \,|\, 10]$$

- **Extract 5**: Swap root with last element (1), heap-size=3:

$$[1, 4, 3 \,|\, 5, 10]$$

Apply MAX-HEAPIFY at root (1), swap with larger child 4:

$$[4, 1, 3 \,|\, 5, 10]$$

- **Extract 4**: Swap root with last element (3), heap-size=2:

$$[3, 1 \,|\, 4, 5, 10]$$

Apply MAX-HEAPIFY (already heap):

$$[3, 1 \,|\, 4, 5, 10]$$

- **Extract 3**: Swap root with last element (1), heap-size=1:

$$[1 \,|\, 3, 4, 5, 10]$$

Only one element left, heap sort is finished.

Final sorted array:

$$[1, 3, 4, 5, 10]$$