Data Structures and Algorithms

Lecture 16

Aniket Basu Roy

BITS Pilani Goa Campus

2026-02-25 Wed

Agenda

Data Structures

Heaps

- Array representation
- Max-heap property
- MAX-HEAPIFY
- BUILD-MAX-HEAP

Motivation: Priority Queue

We want a data structure that supports:

Motivation: Priority Queue

We want a data structure that supports:

- Insert a new element with a priority
- Extract the element with the highest priority

Motivation: Priority Queue

We want a data structure that supports:

- Insert a new element with a priority
- Extract the element with the highest priority

Examples

- Job scheduling
- Dijkstra's shortest path
- Event simulation

Motivation: Priority Queue

We want a data structure that supports:

- Insert a new element with a priority
- Extract the element with the highest priority

Examples

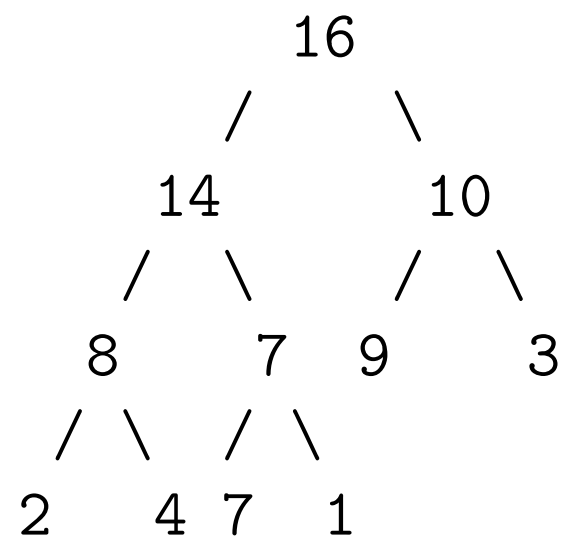
- Job scheduling
- Dijkstra's shortest path
- Event simulation

A heap is the right data structure for this.

What is a Heap?

- A heap is a nearly complete binary tree stored compactly as an array.
- Completely filled on all levels except possibly the lowest, filled from the left.

```
Array: [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]
Index:   1   2   3  4  5  6  7  8  9 10
```



Array Representation

For a node at index i:

Relation	Formula
Parent	$\lfloor i/2 \rfloor$
Left child	$2i$
Right child	$2i + 1$

```
PARENT(i)       return floor(i/2)
LEFT(i)         return 2i
RIGHT(i)        return 2i + 1
```

Max-Heap Property

Definition

For every node i other than the root:

$$A[\text{PARENT}(i)] \geq A[i]$$

- The largest element is at the root.
- Every subtree is itself a max-heap.

Min-Heap

Symmetric property: $A[\text{PARENT}(i)] \leq A[i]$.

- Smallest element at the root.
- Useful for a min-priority queue.

Height of a Heap

- A heap of n elements has height $h = \lfloor \lg n \rfloor$.

Height of a Heap

- A heap of n elements has height $h = \lfloor \lg n \rfloor$.
- At most $\lceil n/2^{h+1} \rceil$ nodes of height h.

Height of a Heap

- A heap of n elements has height $h = \lfloor \lg n \rfloor$.
- At most $\lceil n/2^{h+1} \rceil$ nodes of height h.
- In particular, at most $\lceil n/2 \rceil$ leaves (nodes of height 0).

MAX-HEAPIFY

The Problem

- $A[i]$ may be smaller than its children.
- But subtrees rooted at LEFT(i) and RIGHT(i) are valid max-heaps.

MAX-HEAPIFY "floats" $A[i]$ down to restore the max-heap property.

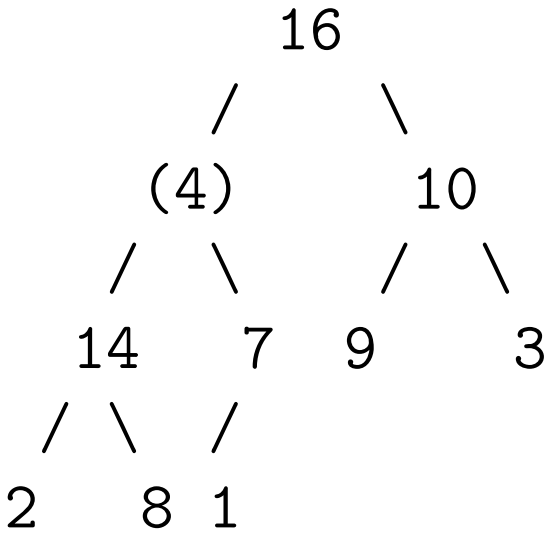
MAX-HEAPIFY

Pseudocode

```
MAX-HEAPIFY(A, i)
1.   l = LEFT(i)
2.   r = RIGHT(i)
3.   if l <= A.heap-size and A[l] > A[i]
4.        largest = l
5.   else largest = i
6.   if r <= A.heap-size and A[r] > A[largest]
7.        largest = r
8.   if largest != i
9.        exchange A[i] with A[largest]
10.       MAX-HEAPIFY(A, largest)
```

MAX-HEAPIFY: Example

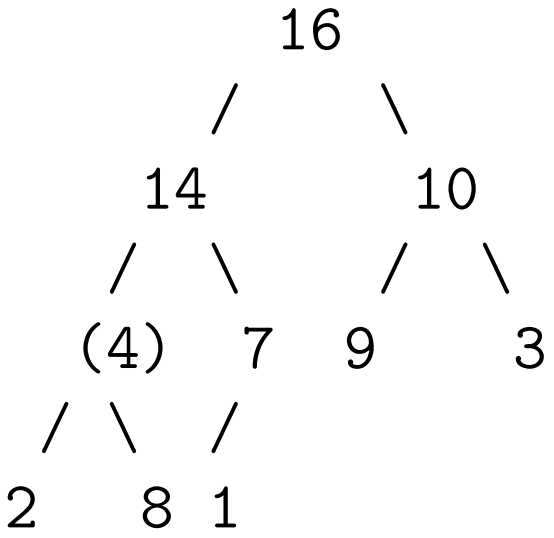
MAX-HEAPIFY$(A, 2)$:



largest $=$ LEFT$(2) = 14$, since $A[4] = 14 > A[2] = 4 \Rightarrow$ swap

MAX-HEAPIFY: Example (contd.)

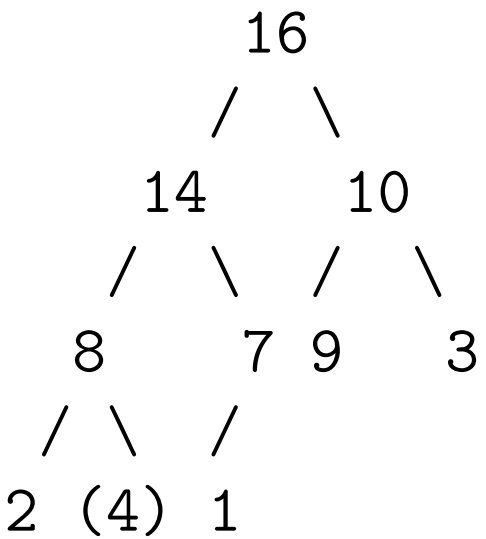
After swap, recurse MAX-HEAPIFY($A, 4$):



$A[9] = 8 > A[4] = 4 \Rightarrow$ swap $A[4]$ and $A[9]$

MAX-HEAPIFY: Example (contd.)

Final:



MAX-HEAPIFY: Time Complexity

- $O(1)$ work per level, recurse on a subtree.
- Subtree has at most $2n/3$ elements (worst case: bottom level half full).

Recurrence:

$$T(n) \leq T(2n/3) + \Theta(1)$$

By Master Theorem (Case 2):

$$T(n) = O(\lg n) = O(h)$$

BUILD-MAX-HEAP

Key Observation

Elements $A[\lfloor n/2 \rfloor + 1], \ldots, A[n]$ are leaves — trivially valid max-heaps.
Call MAX-HEAPIFY only on internal nodes, bottom to top.

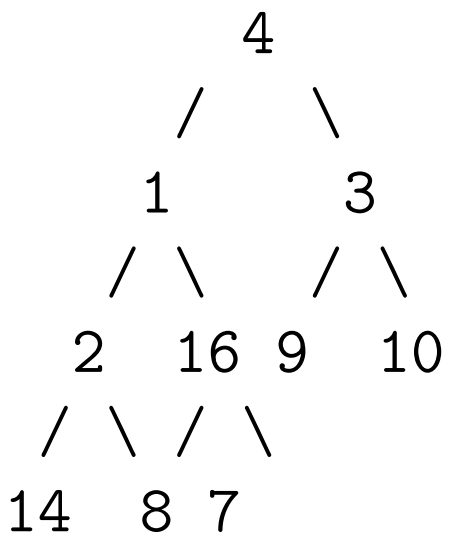
Pseudocode

```
BUILD-MAX-HEAP(A, n)
1.  A.heap-size = n
2.  for i = floor(n/2) downto 1
3.      MAX-HEAPIFY(A, i)
```

BUILD-MAX-HEAP: Example

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

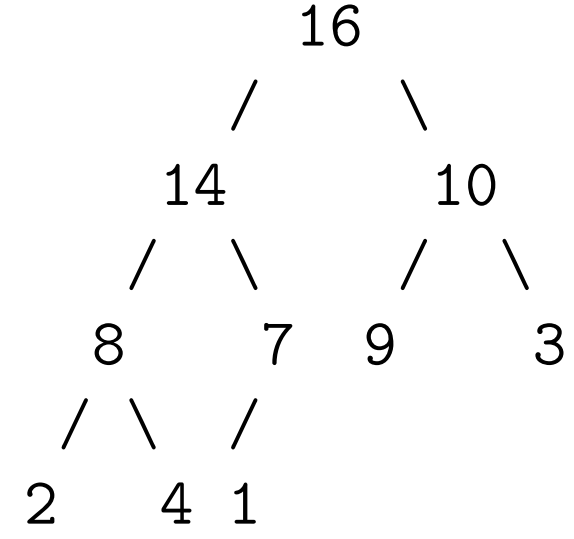
Initial tree:



floor(10/2) = 5, call MAX-HEAPIFY on i=5,4,3,2,1

BUILD-MAX-HEAP: Example

Result:



BUILD-MAX-HEAP: Time Complexity

- Naive: n calls $\times$ $O(\lg n)$ each $= O(n \lg n)$ — not tight.

Tighter: nodes at height h cost $O(h)$; at most $\lceil n/2^{h+1} \rceil$ such nodes.

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h} \right) = O(n \cdot 2) = O(n)$$

(Using $\sum_{h=0}^{\infty} h x^h = \frac{x}{(1-x)^2}$ with $x = 1/2$.)

BUILD-MAX-HEAP runs in $\Theta(n)$ time.

Summary

Operation	Time Complexity
MAX-HEAPIFY	$O(\lg n)$
BUILD-MAX-HEAP	$\Theta(n)$