Binary Heaps

Instructors: Subrahmanyam Kalyanasundaram
Karteek Sreenivasaiah

Week 6

Heaps

Heap

A max-heap supports the following functions:

- Insert(*val*) – Inserts *val* into the heap.
- ExtractMax() – Returns and removes the maximum element from the heap.

Binary max heap

A binary max-heap satisfies the following properties:

1. Structural Property: Is a complete binary tree except possibly for the lowest level, which is "left-filled".
2. Heap Property: The value of a node is greater than that of both its children.

Binary max heap

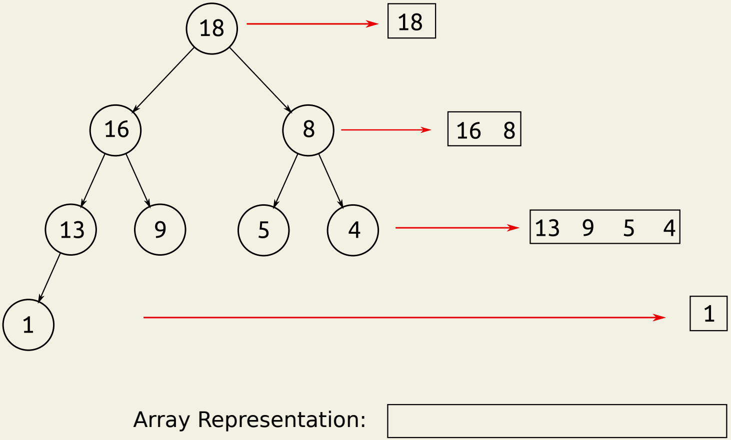
A binary max-heap satisfies the following properties:

1. Structural Property: Is a complete binary tree except possibly for the lowest level, which is "left-filled".
2. Heap Property: The value of a node is greater than that of both its children.

Note: It is not a search tree.

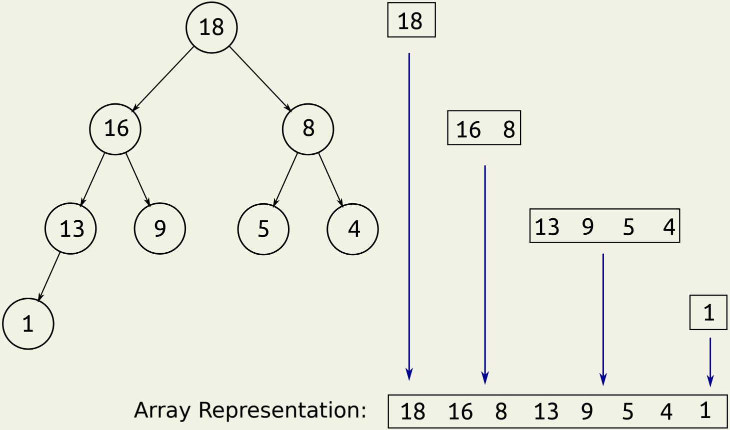
Data Structure

Read off from top to bottom, left to right.



Data Structure

Read off from top to bottom, left to right.



Questions

About Heaps:

1. How many nodes does a height h heap have? (both bounds)
2. What is the maximum height of a heap with n nodes?

About the array implementation:

1. What is the array index of the children of the node at $A[i]$?
2. What is the array index of the right sibling of the node at $A[i]$?

Heaps using arrays

Typically, a heap is built starting with an arbitrary array:

- Procedure BuildHeap(Array A) – Takes an array and rearranges the elements to form a heap.

In Object Oriented languages, BuildHeap is essentially the *Constructor* of class Heap.
The procedure BuildHeap works by using a method called Heapify(*node*).

Heapify

The Heapify(*node*) procedure:

- If *node* violates the heap property:
  1. Swap value of *node* with the largest of its two children.
  2. Call Heapify on the child replaced.
- Else, do nothing and return.

Heapify

The HEAPIFY(*node*) procedure:

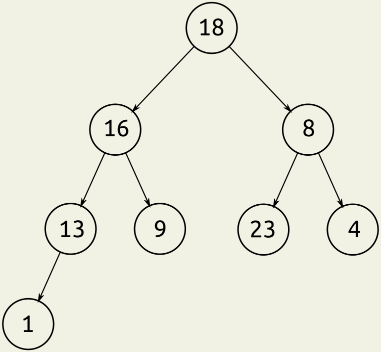
- If *node* violates the heap property:
  1. Swap value of *node* with the largest of its two children.
  2. Call HEAPIFY on the child replaced.
- Else, do nothing and return.

Note:

- The Heapify procedure assumes that both the subtrees under *node* are already heaps.
- It merely resolves the possible conflict between the value at *node* and its children and recurses.
- Can take $O(\log n)$ time.

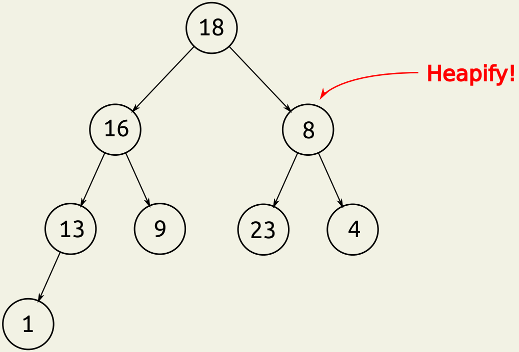
Heapify

Example:



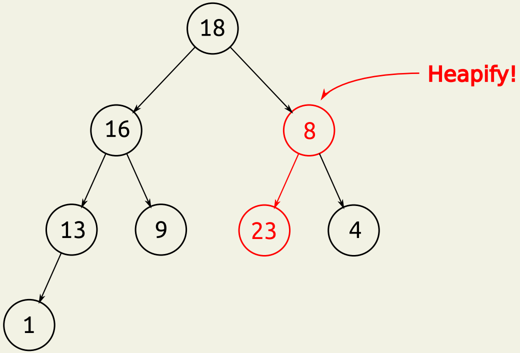
Heapify

Example:



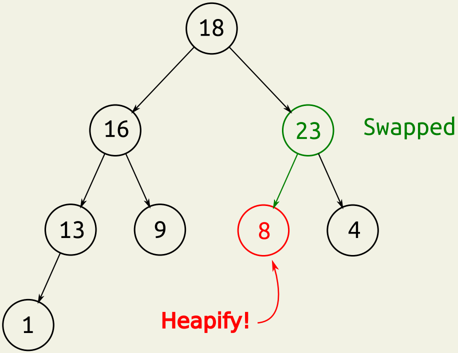
Heapify

Example:



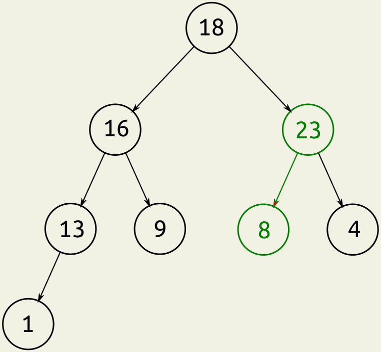
Heapify

Example:



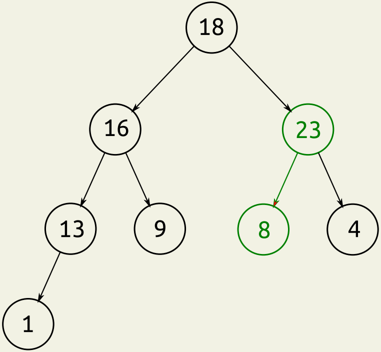
Heapify

Example:



Heapify

Note that Heapify only resolves conflicts downwards.



Exercises

Write the following procedures:

- INSERT(*val*):
  - Insert new value as the last element in the array.
  - Repeatedly Heapify *upwards* from the new element.
  - This can also be viewed as "sifting".
- EXTRACTMAX(): Swap positions of root with last leaf. Heapfiy at new root.

Exercises

Write the following procedures:

- INSERT(val):
  - Insert new value as the last element in the array.
  - Repeatedly Heapify *upwards* from the new element.
  - This can also be viewed as "sifting".
- EXTRACTMAX(): Swap positions of root with last leaf. Heapfiy at new root.
- Running time?

Building a Heap

Two ways:

- William's method: Take each element and use INSERT procedure.
- Floyd's method: Take all elements in an arbitrary array. Heapify repeatedly.

Building a Heap

Two ways:

- William's method: Take each element and use INSERT procedure. Takes $O(n \log n)$ time.
- Floyd's method: Take all elements in an arbitrary array. Heapify repeatedly.

Building a Heap

The procedure BuildHeap(A) by Floyd is the following:

- For i from n to 1:
  - Heapify(i)

Building a Heap

The procedure $\textsc{BuildHeap}(A)$ by Floyd is the following:

- For i from n to 1:
  - $\textsc{Heapify}(i)$

Note: Indices $n/2$ to n form leaves of the heap.
The leaves are already heaps (trivially).
Hence it suffices to run the above loop from $n/2$ to 1.

Visualization

On the board

Analysis of Floyd's Method

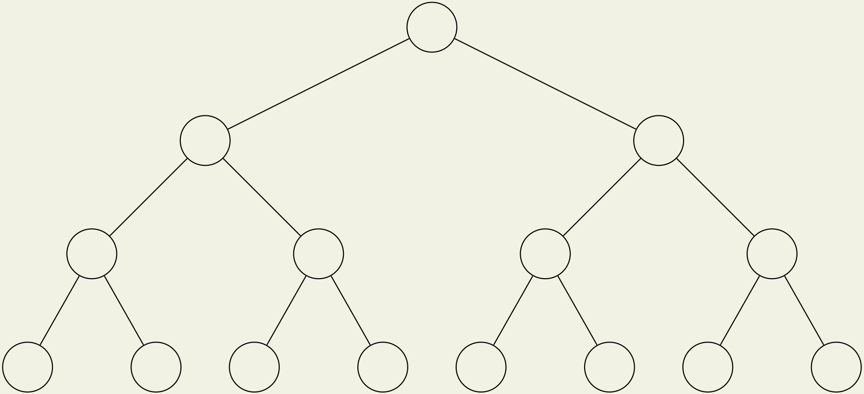
- We need to do $n/2$ Heapify operations
- Each Heapify can take $O(\log n)$ time
- So total time is $O(n \log n)$

Analysis of Floyd's Method

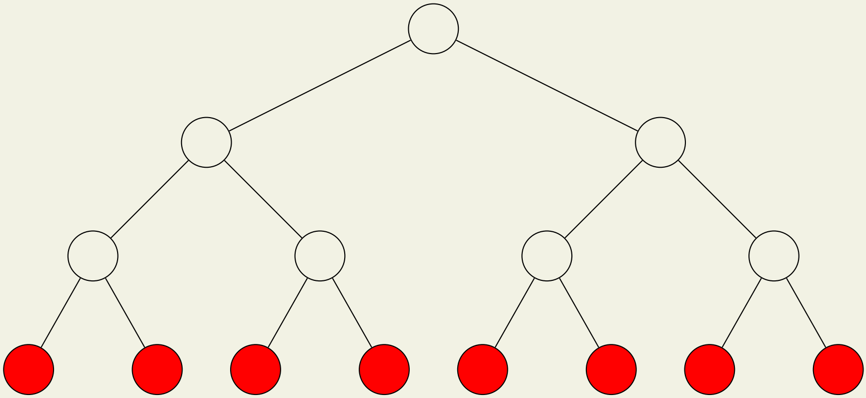
- We need to do $n/2$ Heapify operations
- Each Heapify can take $O(\log n)$ time
- So total time is $O(n \log n)$

- But most of the Heapify operations are small
- We have $n/2$ nodes at height 1, $n/4$ nodes at height 2 and so on
- It can be shown that BuildHeap(A) takes only $O(n)$ time

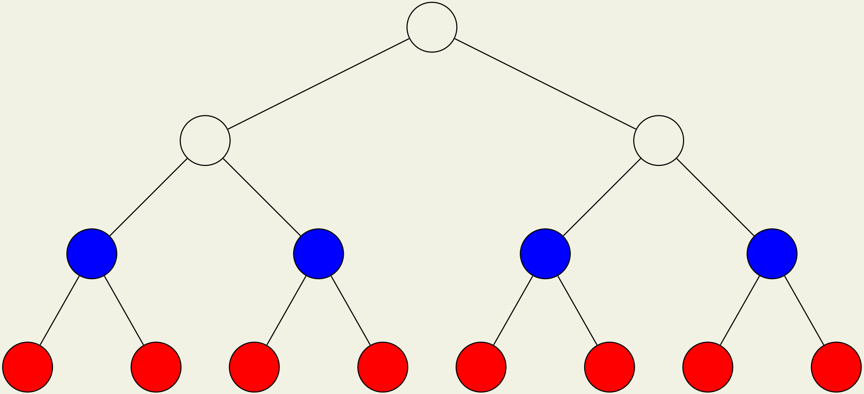
Analysis of Floyd's Method



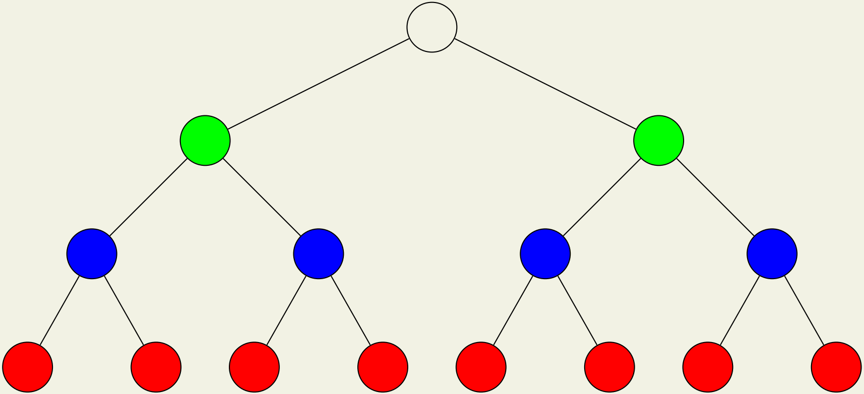
Analysis of Floyd's Method



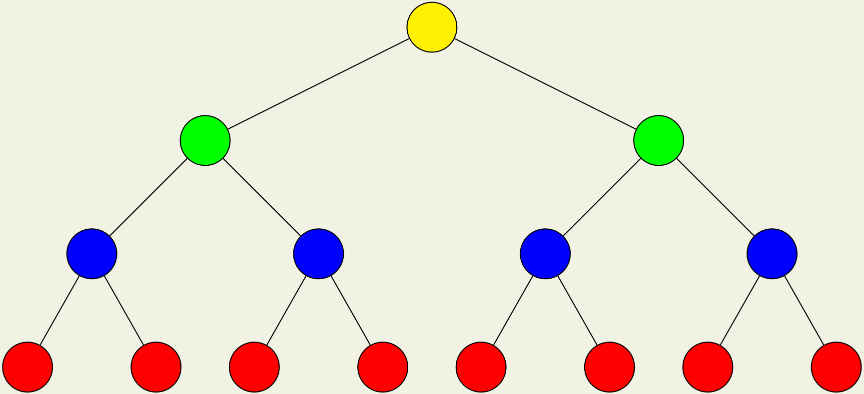
Analysis of Floyd's Method



Analysis of Floyd's Method



Analysis of Floyd's Method



Analysis of Floyd's Method

- We have $n/2$ values needing at most 1 swap
- We have $n/4$ values needing at most 2 swaps
- We have $n/8$ values needing at most 3 swaps, and so on.

Analysis of Floyd's Method

- We have $n/2$ values needing at most 1 swap
- We have $n/4$ values needing at most 2 swaps
- We have $n/8$ values needing at most 3 swaps, and so on.

$$\begin{aligned}
\text{Total no. of swaps} &= \frac{n}{2} \cdot 1 + \frac{n}{4} \cdot 2 + \frac{n}{8} \cdot 3 + \ldots \\
&= n \left(\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \ldots \right) \\
&= n \sum_{i=1}^{\log n} \frac{i}{2^i} \leq n \sum_{i=1}^{\infty} \frac{i}{2^i}
\end{aligned}$$

Analysis of Floyd's Method

- The number of swaps is at most $n \sum_{i=1}^{\infty} \frac{i}{2^i}$
- This can be shown to be at most $2n$
- Thus BuildHeap is performed in $O(n)$ time

Heap Sort

- Given an array A:
- Run $\textsc{BuildHeap}(A)$
- Repeatedly do $\textsc{ExtractMax}()$
- What is the total time?