# CS 361, Lecture 13

Jared Saia
University of New Mexico

## Administrative

- Midterm will be Thursday, March 13th at regular class time and place
- You can bring 2 pages of "cheat sheets" to use during the exam. Otherwise the exam is closed book and closed note
- There will be a hw due March 13th which I'll put up on the web page today. It'll be somewhat easier than usual and is intended to help you study for the exam.
- Note that the web page contains links to prior classes and their midterms. *Many of the questions on my midterm will be similar in flavor to these past midterms!*



## Outline

*"Partly because of his computational skills, Gerbert, in his later years, was made Pope by Otto the Great, Holy Roman Emperor, and took the name Sylvester II. By this time, his gift in the art of calculating contributed to the belief, commonly held throughout Europe, that he had sold his soul to the devil."*
*- Dominic Olivastro in the book* Ancient Puzzles, *1993*

- Intro to (Binary) Heaps (Chapter 6)
- Maintaining the Heap Property
- Building a Heap



## What is a Heap

- "A heap data structure is an array that can be viewed as a nearly complete binary tree"
- Each element of the array corresponds to a value stored at some node of the tree
- The tree is completely filled at all levels except for possibly the last which is filled from left to right



## heap-size (A)

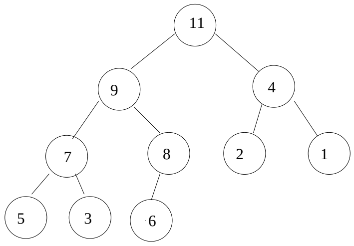
- An array $A$ that represents a heap has two attributes
  - length (A) which is the number of elements in the array
  - heap-size (A) which is the number of elems in the heap stored within the array
- I.e. only the elements in A[1..heap-size (A)] are elements of the heap



## Tree Structure

- A[1] is the root of the tree
- For all $i$, $1 < i <$ heap-size (A)
  - Parent (i) = $\lfloor i/2 \rfloor$
  - Left (i) = $2i$
  - Right (i) = $2i + 1$
- If Left (i) $>$ heap-size (A), there is no left child of $i$
- If Right (i) $>$ heap-size (A), there is no right child of $i$
- If Parent (i) $< 0$, there is no parent of $i$

## Example



A:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 11 | 9 | 4 | 7 | 8 | 2 | 1 | 5 | 3 | 6 |



## Max-Heap Property

- For every node $i$ other than the root, A[Parent (i)] $\geq$ A[i]



## Max-Heap Property

- For every node $i$ other than the root, A[Parent (i)] $\geq$ A[i]
- Parent is always at least as large as its children
- Largest element is at the root

(A Min-heap is organized the opposite way)



## Height of Heap

- Height of a node in a heap is the number of edges in the longest simple downward path from the node to a leaf
- Height of a heap of $n$ elements is $\Theta(n)$. Why?



## Maintaining Heaps

- Q: How to maintain the heap property?
- A: *Max-Heapify* is given an array and an index $i$
- Assumes that the binary trees rooted at $Left(i)$ and $Right(i)$ are max-heaps
- But $A[i]$ may be smaller than its children
- *Max-Heapify* ensures that after its call, the subtree rooted at $A[i]$ is a Max-Heap

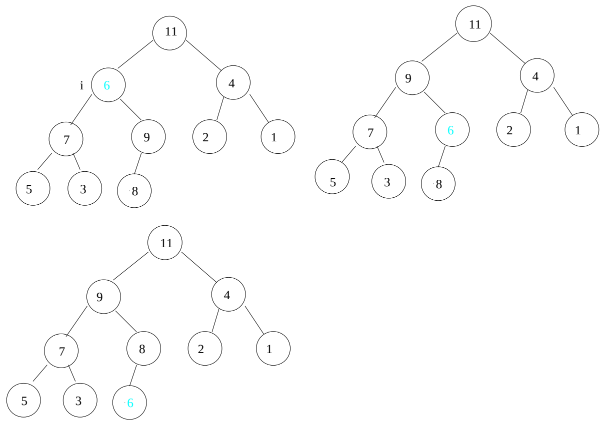


## Max-Heapify

Max-Heapify (A,i)

1. $l = Left(i)$
2. $r = Right(i)$
3. $largest = i$
4. if ($l \leq$ heap-size($A$) and $A[l] > A[i]$) then $largest = l$
5. if ($r \leq$ heap-size($A$) and $A[r] > A[largest]$) then $largest = r$
6. if $largest \neq i$ then
   (a) exchange A[i] and A[largest]
   (b) Max-Heapify (A,largest)

## Example






## Analysis

- Let $T(h)$ be the runtime of max-heapify on a subtree of height $h$
- Then $T(1) = \Theta(1)$, $T(h) = T(h-1) + 1$
- Solution to this recurrence is $T(h) = \Theta(h)$
- Thus if we let $T(n)$ be the runtime of max-heapify on a subtree of *size* $n$, $T(n) = O(\log n)$, since $\log n$ is the maximum height of heap of size $n$



## Build-Max-Heap

- Q: How can we convert an arbitrary array into a max-heap?
- A: Use Max-Heapify in a bottom-up manner
- Note: The elements A[$\lfloor n/2 \rfloor + 1$],..,A[n] are all leaf nodes of the tree, so each is a 1 element heap to begin with



## Build-Max-Heap

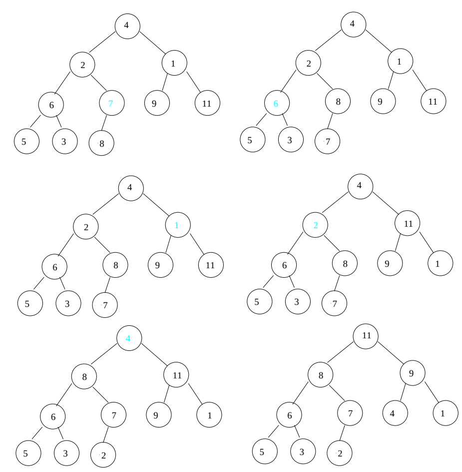
Build-Max-Heap (A)

1. heap-size (A) = length (A)
2. for ($i = \lfloor length(A)/2 \rfloor$;$i > 0$;$i--$)
   (a) do Max-Heapify (A,i)



## Example

A = 4 2 1 6 7 9 11 5 3 8






## Loop Invariant

- Loop Invariant: "At the start of the $i$-th iteration of the for loop, each node $i+1, i+2, \ldots n$ is the root of a max-heap

## Correctness

- **Initialization:** $i = \lfloor n/2 \rfloor$ prior to first iteration. But each node $\lfloor n/2 \rfloor + 1$, $\lfloor n/2 \rfloor + 2, \ldots, n$ is a leaf so is the root of a trivial max-heap
- **Termination:** At termination, $i = 0$, so each node $1, \ldots, n$ is the root of a max-heap. In particular, node 1 is the root of a max heap.



## Maintenance

- **Maintenance:** First note that if the nodes $i+1, \ldots n$ are the roots of max-heaps before the call to Max-Heapify (A,i), then they will be the roots of max-heaps after the call. Further note that the children of node $i$ are numbered higher than $i$ and thus by the loop invariant are both roots of max heaps. Thus after the call to Max-Heapify (A,i), the node $i$ is the root of a max-heap. Hence, when we decrement $i$ in the for loop, the loop invariant is established.



## Time Analysis

(Naive) Analysis:

- Max-Heapify takes $O(\log n)$ time per call
- There are $O(n)$ calls to Max-Heapify
- Thus, the running time is $O(n \log n)$



## Time Analysis

Better Analysis. Note that:

- An $n$ element heap has height $\lfloor \log n \rfloor$
- There are at most $\lceil n/2^{h+1} \rceil$ nodes of any height $h$ (to see this, consider the min number of nodes in a heap of height $h$)
- Time required by Max-Heapify when called on a node of height $h$ is $O(h)$
- Thus total time is: $\sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h)$



## Analysis

$$\begin{aligned}
\sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) &= O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} \frac{h}{2^h}\right) && (1) \\
&= O\left(n \sum_{h=0}^{\text{infinity}} \frac{h}{2^h}\right) && (2) \\
&= O(n) && (3)
\end{aligned}$$



## Analysis

The last step follows since for all $|x| < 1$,

$$\sum_{i=0}^{\text{infinity}} i x^i = \frac{x}{(1-x)^2} \qquad (4)$$

Can get this equality by recalling that for all $|x| < 1$,

$$\sum_{i=0}^{\text{infinity}} x^i = \frac{1}{1-x},$$

and taking the derivative of both sides!

## Heap-Sort

Heap-Sort (A)

1. Build-Max-Heap (A)
2. for (i=length (A);$i > 1$;$i --$)
   (a) do exchange A[1] and A[i]
   (b) heap-size (A) = heap-size (A) - 1
   (c) Max-Heapify (A,1)



## Analysis

- Build-Max-Heap takes $O(n)$, and each of the $O(n)$ calls to Max-Heapify take $O(\log n)$, so Heap-Sort takes $O(n \log n)$
- Correctness???



## Todo

- Read Chapter 6