Lecture: Analysis of Algorithms (CS483 - 001)

Amarda Shehu

Spring 2017
Outline of Today’s Class

- Sorting in $O(n \lg n)$ Time: Heapsort
Desired property of Mergesort: Time complexity of $O(n \log n)$
Desired property of Insertion sort: Sorting in place
Heapsort combines these two properties
Heapsort illustrates a powerful algorithm design technique: a data structure organizes information during execution
Heap Data Structure

- Array object regarded as nearly complete binary tree
- Attributes:
  - length(A): # elements in A
  - heap_size(A): # elements in heap
- Heap property: value of parent \( \geq \) values of children
- Filled from root to leaves, left to right
  - Root of tree is stored in \( A[1] \)
  - Given index \( i \) of a node:
    - parent: \( \text{PARENT}(i) \leftarrow \lfloor i/2 \rfloor \)
    - left child: \( \text{LEFT}(i) \leftarrow 2i \)
    - right child: \( \text{RIGHT}(i) \leftarrow 2i + 1 \)
Heaps as a Balanced Binary Tree

- In a tree:
  - Depth of a node is distance of node from root
  - Depth of a tree is depth of deepest node
  - Height is the opposite of depth
  - Height and depth are often confused

- A binary tree of depth $d$ is balanced if all nodes at depths 0 through $d - 2$ have two children

Illustration of balanced and unbalanced binary trees:
A balanced binary tree of depth $d$ is left-justified if:

1. It has $2^k$ nodes at depth $k \forall k < d$
2. All leaves at depth $d$ are as far left as possible
Back to the Heap Property

- In a max-heap: $A[\text{PARENT}(i)] \geq A[i]$, $\forall i \geq 1$
- In a min-heap: $A[\text{PARENT}(i)] \leq A[i]$, $\forall i \geq 1$
- We will focus on the heap property in max-heaps for sorting:

![Diagram of heap properties]

- Leaf nodes automatically have the heap property. Why?
- A binary tree is a heap if all nodes in it have the heap property
- What can you say about the root in a max/min-heap?
Maintaining the Heap Property in a Max-heap

- Given a node that does not have the heap property, one can give it the heap property by exchanging its value with that of the larger child:

  ![Diagram of heap property maintenance](image)

  - Blue node does not have heap property
  - Blue node has heap property

- Note: Upon the exchange, the heap property may be violated in the subtree rooted at the child

- The MAX – HEAPIFY subroutine restores the heap property on the subtree rooted at index \( i \)

- How? The value at \( A[i] \) is floated down in the max-heap
MAX-HEAPIFY(array A, index i)

1: if \( i \) is not a leaf and \( A[\text{LEFT}(i)] \)
   or \( A[\text{RIGHT}(i)] > A[i] \) then
2:   let \( k \) denote larger child
3:   swap\((A[i], A[k])\)
4:   MAX − HEAPIFY\((A, k)\)

Time Complexity: MAX-HEAPIFY

- Let \( H(i) \) denote running time
- Show \( H(i) \in O(lgn) \)
- Hint: Down the tree we go
BUILD-MAX-HEAP(A, size n)

1: for $i \leftarrow \lfloor \frac{n}{2} \rfloor$ to 1 do
2: \hspace{1em} MAX-HEAPIFY(A, i)

**Time: BUILD-MAX-HEAP**
- Why is $\lfloor \frac{n}{2} \rfloor$ bound sufficient?
- Hint: ≠ internal nodes in heap?
- Let $B(n)$ be running time
- Show $B(n) \in O(n \log n)$

**Figure: Trace on above array.**
Tighter Asymptotic Bound on BUILD-MAX-HEAP

1. Show that an \( n \)-element heap has depth (and height) \( \lfloor \lg n \rfloor \).
2. Show that there are at most \( \lceil n/2^{h+1} \rceil \) nodes of height \( h \).

\[
B(n) = \sum_{i=1}^{n/2} H(i) = \sum_{h=0}^{\lfloor \lg n \rfloor} \{ \lceil \frac{n}{2^{h+1}} \rceil O(h) \} \\
\in O(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h})
\]

Note: \( \sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} \) \( \text{(A.8)} \)

Hence:
\[
O(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}) = O(n \sum_{h=0}^{\infty} \frac{h}{2^h}) = O(n)
\]

Conclusion: A heap can be built in \( O(n) \) time.
HEAPSORT: Pseudocode and Time Complexity

**HEAPSORT(A)**

1: BUILD-MAX-HEAP(A)
2: for \( i \leftarrow A.length \) to 2 do
3: swap([A[1], A[i]])
4: A.heap-size \leftarrow A.heap-size - 1
5: MAX-HEAPIFY(A, 1)

**Time: HEAPSORT**

- HEAPSORT takes \( O(n \lg n) \).
- BUILD-MAX-HEAP runs in linear time - \( O(n) \).
- There are \( n - 1 \) calls to MAX-HEAPIFY.
- Each call takes \( O(\lg n) \) time.
- So: \( T(\text{HEAPSORT}(A, n)) \in O(n + (n - 1)\lg n) \in O(n\lg n) \)

**Basic Idea Behind HEAPSORT**

- What property holds for root after BUILD-MAX-HEAP?
- Why can we put it at index \( i \)?
- Why do we need to run MAX-HEAPIFY after swap?
A priority queue maintains a set $S$ of elements, each one associated with a key.

Max- or min-priority queues help to rank jobs for scheduling.

Operations for a max-priority queue:

1. $\text{INSERT}(S, x)$ inserts element $x$ in the set $S$
2. $\text{MAXIMUM}(S)$ returns the element of $S$ with the largest key
3. $\text{EXTRACT-MAX}(S)$ removes from $S$ and returns the element with the largest key
4. $\text{INSERT-KEY}(S, x, k)$ increases the value of the key of $x$ to the new value $k$, which is assumed to be at least as large as the current value of the key of $x$