**COMP251: Heaps & Heapsort**

Jérôme Waldispühl  
School of Computer Science  
McGill University  
From (Cormen et al., 2002)  
Based on slides from D. Plaisted (UNC)

---

**Heap data structure**

- Tree-based data structure (here, binary tree, but we can also use k-ary trees)
- Max-Heap  
  - Largest element is stored at the root.  
  - For all nodes $j$, excluding the root, $A[\text{PARENT}(j)] \geq A[j]$.
- Min-Heap  
  - Smallest element is stored at the root.  
  - For all nodes $j$, excluding the root, $A[\text{PARENT}(j)] \leq A[j]$.

---

**Heaps as arrays**

- Max-heap as a binary tree.  
- Last row filled from left to right.

---

**Heaps – Example**

Max-heap as an array.

- Map from array elements to tree nodes and vice versa  
  - Root = $A[1]$  
  - Left($i$) = $A[2i]$  
  - Right($i$) = $A[2i+1]$  
  - Parent($i$) = $A[\lfloor i/2 \rfloor]$

---

**Height**

- Height of a node in a tree: the number of edges on the longest simple path down from the node to a leaf.  
- Height of a heap = height of the root = $\Theta(\log n)$.
- Most Basic operations on a heap run in $O(\log n)$ time  
- Shape of a heap

---

**Heaps in Sorting**

- Use max-heaps for sorting.  
- The array representation of max-heap is not sorted.  
- Steps in sorting  
  - Convert the given array of size $n$ to a max-heap ($\text{BuildMaxHeap}$)  
  - Swap the first and last elements of the array.  
    - Now, the largest element is in the last position – where it belongs.  
    - That leaves $n-1$ elements to be placed in their appropriate locations.  
    - However, the array of first $n-1$ elements is no longer a max-heap.  
    - Fix the element at the root down one of its subtrees so that the array remains a max-heap ($\text{MaxHeapify}$)  
    - Repeat step 2 until the array is sorted.
Heapsort

- Combines the better attributes of merge sort and insertion sort.
  - Like merge sort, but unlike insertion sort, running time is $O(n \log n)$.
  - Like insertion sort, but unlike merge sort, sorts in place.
- Introduces an algorithm design technique
  - Create data structure (heap) to manage information during the execution of an algorithm.
- The heap has other applications beside sorting.
  - Priority Queues (See COMP250)

Maintaining the heap property

- Suppose two subtrees are max-heaps, but the root violates the max-heap property.
- Fix the offending node by exchanging the value at the node with the larger of the values at its children.
  - May lead to the subtree at the child not being a heap.
- Recursively fix the children until all of them satisfy the max-heap property.

MaxHeapify – Example

- Node n=2

MaxHeapify – Example

Running Time for MaxHeapify($A, n$)

- $T(n) = T(largest) + \Theta(1)$
- $largest \leq 2n/3$ (worst case occurs when the last row of tree is exactly half full)
- $T(n) \leq T(2n/3) + \Theta(1) \Rightarrow T(n) = O(\log n)$
- Alternately, MaxHeapify takes $O(h)$ where $h$ is the height of the node where MaxHeapify is applied

Procedure MaxHeapify

Assumption: Left(i) and Right(i) are max-heaps.

Building a heap

- Use MaxHeapify to convert an array $A$ into a max-heap.
- Call MaxHeapify on each element in a bottom-up manner.

```
BuildMaxHeap(A)
1. \( n \leftarrow \text{length}(A) \)
2. \text{for } i \leftarrow \lfloor \text{length}(A)/2 \rfloor \text{ downto } 1
3. \text{ do MaxHeapify}(A, i, n)
```
BuildMaxHeap – Example

Input Array:

24 21 23 22 36 29 30 34 28 27

Starting tree (not max-heap)

BuildMaxHeap – Example

MaxHeapify(10/2) = 5
MaxHeapify(4)
MaxHeapify(3)
MaxHeapify(2)
MaxHeapify(1)

Correctness of BuildMaxHeap

- Loop Invariant: At the start of each iteration of the for loop, each node $i, i+2, \ldots, n$ is the root of a max-heap.
- Initialization:
  - Before first iteration $i = \lceil n/2 \rceil$
  - Nodes $\lceil n/2 \rceil + 1, \lceil n/2 \rceil + 2, \ldots, n$ are leaves, hence roots of trivial max-heaps.
- Maintenance:
  - By LI, subtrees at children of node $i$ are max heaps.
  - Hence, MaxHeapify($i$) renders node $i$ a max heap root (while preserving the max heap root property of higher-numbered nodes).
  - Decrementing $i$ reestablishes the loop invariant for the next iteration.

Running Time of BuildMaxHeap

- Loose upper bound:
  - Cost of a MaxHeapify call $\times$ No. of calls to MaxHeapify
  - $O(\log n) \times O(n) = O(n \log n)$
- Tighter bound:
  - Cost of MaxHeapify is $O(h)$.
  - $\sum_{h=\log n}^{\log n(n/2)} O(h) = O(n \log n(n/2)) = O(n)$

Heapsort

1. Builds a max-heap from the array.
2. Put the maximum element (i.e. the root) at the correct place in the array by swapping it with the element in the last position in the array.
3. "Discard" this last node (knowing that it is in its correct place) by decreasing the heap size, and call MAX-HEAPIFY on the new root.
4. Repeat this process (goto 2) until only one node remains.

Heapsort($A$)

1. Build-Max-Heap($A$)
2. for $i \leftarrow \text{length}[A]$ downto 2
4. MaxHeapify($A$, 1, i-1)
Heapsort – Example

Heapsort – Example

Heapsort – Example

Heapsort – Example

Heap Procedures for Sorting

- BuildMaxHeap \( O(n) \)
- for loop \( n-1 \) times (i.e. \( O(n) \))
  - exchange elements \( O(1) \)
  - MaxHeapify \( O(\lg n) \)

\( \Rightarrow \) HeapSort \( O(n \lg n) \)