ECS 122A
Algorithm Design and Analysis

Instructor: Qirun Zhang
Agenda

- Heapsort
- Priority queue
Course updates

• About homework
  – Will be posted tomorrow
The Master Theorem Revisited

- if \( T(n) = aT(n/b) + f(n) \) then

\[
T(n) = \begin{cases} 
\Theta(n^{\log_b a}) & f(n) = O(n^{\log_b a - \varepsilon}) \\
\Theta(n^{\log_b a} \log n) & f(n) = \Theta(n^{\log_b a}) \\
\Theta(f(n)) & f(n) = \Omega(n^{\log_b a + \varepsilon}) \text{ AND } \quad a f(n/b) < c f(n) \text{ for large } n \\
\end{cases}
\]
Heap Operations: Heapify()

- **Heapify():** maintain the heap property
  - Given: a node $i$ in the heap with children $l$ and $r$
  - Given: two subtrees rooted at $l$ and $r$, assumed to be heaps
  - Problem: The subtree rooted at $i$ may violate the heap property
  - Action: let the value of the parent node “float down” so subtree at $i$ satisfies the heap property
Heap Operations: Heapify()
Analyzing Heapify()
Heap Operations: BuildHeap()
Analyzing BuildHeap()
Analyzing BuildHeap(): Tight
Heapsort

• **Given** `BuildHeap()`, an in-place sorting algorithm is easily constructed:
  
  - Maximum element is at `A[1]`
  - Discard by swapping with element at `A[n]`
    * Decrement `heap_size[A]`
    * `A[n]` now contains correct value
  - Restore heap property at `A[1]` by calling `Heapify()`
Analyzing Heapsort

- The call to `BuildHeap()` takes $O(n)$ time
- Each of the $n - 1$ calls to `Heapify()` takes $O(\lg n)$ time
- Thus the total time taken by `HeapSort()`
  
  \[
  = O(n) + (n - 1) O(\lg n) \\
  = O(n) + O(n \lg n) \\
  = O(n \lg n)
  \]
Priority Queues

- Heapsort is a nice algorithm, but in practice Quicksort (coming up) usually wins
- But the heap data structure is incredibly useful for implementing *priority queues*
  - A data structure for maintaining a set $S$ of elements, each with an associated value or *key*
  - Supports the operations $\text{Insert}()$, $\text{Maximum}()$, and $\text{ExtractMax}()$
Priority Queue Operations

- **Insert**(S, x) inserts the element x into set S
- **Maximum**(S) returns the element of S with the maximum key
- **ExtractMax**(S) removes and returns the element of S with the maximum key
The End