1 Introduction

- We have discussed several fundamental algorithms (e.g. sorting)
- We will now turn to data structures; Play an important role in algorithms design.
  - Today we will discuss priority queues and next time structures for maintaining ordered sets.

2 Priority Queue

- A priority queue supports the following operations on a set $S$ of $n$ elements:
  - INSERT: Insert a new element $e$ in $S$
  - FINDMIN: Return the minimal element in $S$
  - DELETEMIN: Delete the minimal element in $S$

- Sometimes we are also interested in supporting the following operations:
  - CHANGE: Change the key (priority) of an element in $S$
  - DELETE: Delete an element from $S$

- We can obviously sort using a priority queue:
  - Insert all elements using INSERT
  - Delete all elements in order using FINDMIN and DELETEMIN

- Priority queues have many other applications, e.g. in discrete event simulation, graph algorithms

2.1 Array or List implementations

- The first implementation that comes to mind is ordered array:

  1 3 5 6 7 8 9 15 12 15 17

  - FINDMIN can be performed in $O(1)$ time
  - DELETEMIN and INSERT takes $O(n)$ time since we need to expand/compress the array after inserting or deleting element.

- If the array is unordered all operations take $O(n)$ time.
• We could use double linked sorted list instead of array to avoid the $O(n)$ expansion/compression cost
  
  – but INSERT can still take $O(n)$ time.

2.2 Heap implementation

• One way of implementing a priority queue is using a heap

• Heap definition:
  
  – Perfectly balanced binary tree
    * lowest level can be incomplete (but filled from left-to-right)
  
  – For all nodes $v$ we have $\text{key}(v) \geq \text{key}(\text{parent}(v))$

• Example:

  ![Heap example diagram]

  Heap can be implemented (stored) in two ways (at least)
  
  – Using pointers
  – In an array level-by-level, left-to-right

  Example:

  ![Heap array example diagram]

  * Note the nice property that the left and right children of node stored in entry $i$ is in entry $2i$ and $2i + 1$, respectively

• Properties of heap:
  
  – Height $\Theta(\log n)$
  – Minimum of $S$ is stored in root

• Operations:
  
  – INSERT
    * Insert element in new leaf in leftmost possible position on lowest level
    * Repeatedly swap element with element in parent node until heap order is reestablished ($\text{up-heapify}$)
Example: Insertion of 4

- **FindMin**
  - Return root element
- **DeleteMin**
  - Delete element in root
  - Move element from rightmost leaf on lowest level to the root (and delete leaf)
  - Repeatedly swap element with element in child node with *minimal* element until heap order is reestablished (**down-heapify**)

Example:

- Running time: All operations traverse at most one root-leaf path ⇒ $O(\log n)$ time.
- **Change** and **Delete** can be handled similarly in $O(\log n)$ time
  - Assuming that we know the element to be changed/deleted.
- Sorting using heap (**Heap-Sort**) takes $\Theta(n \log n)$ time.
  - $n \cdot O(\log n)$ time to insert all elements (build the heap)
  - $n \cdot O(\log n)$ time to output sorted elements
- Sometimes we would like to build a heap faster than $O(n \log n)$
  - Insert elements in any order in perfectly balanced tree
  - **down-heapify** all nodes level-by-level, bottom-up

**Correctness:**
- Induction on height of tree: When doing level $i$, all trees rooted at level $i-1$ are heaps.

**Analysis:**
- Define leaves to be on level 1 (root on level $\log n$)
- $n$ elements ⇒ $\leq \left\lceil \frac{n}{2^\rho} \right\rceil$ leaves ⇒ $\left\lceil \frac{n}{2^\rho} \right\rceil$ elements on level $h$
- Cost of **down-heapify** on a node on level $h$ is $h$
- Total cost: $\sum_{i=1}^{\log n} h \cdot \left\lceil \frac{n}{2^\rho} \right\rceil = \Theta(n) \cdot \sum_{i=1}^{\log n} \frac{h}{2^\rho}$
- $\sum_{i=1}^{\log n} \frac{h}{2^\rho} = O(1)$ so cost is $\Theta(n)$
  - Assume $|x| < 1$ and differentiate $\sum_{h=0}^{\infty} x^h = \frac{1}{x-1}$
  - $\sum_{h=0}^{\infty} h x^{h-1} = \frac{1}{(x-1)^2}$ ⇒ $\sum_{h=0}^{\infty} h x^h = \frac{x}{(x-1)^2}$ ⇒ $\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1/2-1)^2} = O(1)$