Design and Analysis of Algorithms I

# Data Structures

## Heaps and Their Applications

# Heap: Supported Operations

- a container for objects that have keys
  - employer records, network edges, events, etc.

INSERT: add a new object to a heap.

Running time: $O(\log n)$

EXTRACT-MIN: remove an object in heap with a minimum key value. [ties broken arbitrarily]

(equally well, EXTRACT MAX)

Running time: $O(\log n)$ [$n$ = # of objects in heap]

Also: HEAPIFY ($n$ batched Inserts in $O(n)$ time), DELETE ($O(\log n)$ time)

# Application: Sorting

Canonical use of heap: fast way to do repeated minimum computations.

Example: SelectionSort $\approx \Theta(n)$ linear scans, $\Theta(n^2)$ runtime on array of length n.

HeapSort: (1) insert all n array elements into a heap
(2) Extract-Min to pluck out elements in sorted order

Running time = 2n heap operations = $O(n \log n)$ time.

$\Rightarrow$ optimal for a "comparison-based" sorting algorithm!

# Application: Event Manager

"Priority queue" – synonym for a heap.

Example: Simulation (e.g., for a video game)

- objects = event records [action/update to occur at given time in the future]
- key = time event scheduled to occur
- Extract-Min => yields the next scheduled event

# Application: Median Maintanence

**I give you:** a sequence $x_1, \ldots, x_n$ of numbers, one-by-one.

**You tell me:** at each time step $i$, the median of $\{x_1, \ldots, x_i\}$.

**Constraint:** use $O(\log i)$ time at each step $i$.

**Solution:** maintain heaps $H_{Low}$: supports EXTRACT MAX
$H_{High}$: supports EXTRACT MIN

**Key idea:** maintain invariant that $\approx i/2$ smallest (largest) elements in $H_{Low}$ ($H_{High}$)

YOU CHECK: 1) can maintain invariant with $O(\log i)$ work
2) given invariant, can compute median in $O(\log i)$ work

# Application: Speeding Up Dijkstra

Dijkstra's Shortest-Path Algorithm

\# vertices  \# edges

- naive implementation ⇒ run time = $\Theta(nm)$

\# loop iteration

work per iteration [linear scan through edges for minimum computation]

- with heaps ⇒ run time = $O(m \log n)$