Heapsort, Priority Queue

Outline
- Heap data structure
- Extract min
- Insert
- Priority queue
- Heapsort

Binary Min-heap
- Nearly complete binary tree that satisfies the heap property
  - tree completely filled on all levels except lowest level, which is filled from the left
- Heap property:
  - $A[parent(x)] \leq A[x]$

Array Representation

Extract Min

```
EXTRACT-MIN(A)
1    min ← A[1]
3    heap-size[A] ← heap-size[A] - 1
4    MIN-HEAPIFY(A, 1)  // maintain heap property
5    return min
```
- Extract the min element for priority queue
- Simple way to sort - repeatedly extract-min.
Re-establish heap property

Min-Heapify

```
MIN-HEAPIFY(A,i)
2        then smallest ← 2i
3        else smallest ← i
5        then smallest ← 2i + 1
6    if smallest ≠ i
7        then swap A[i] ↔ A[smallest]
8    MIN-HEAPIFY(A, smallest)
```

- What is the running time of MIN-HEAPIFY?

Insert

```
INSERT(key, A)
1    heap ← size(A) ← heap ← size(A) + 1
2        i ← heap − size(A)
3    while i > 1 and A[\lceil i/2 \rceil] > key
4        do A[i] ← A[\lceil i/2 \rceil]
5        i ← \lceil i/2 \rceil
6    A(i) ← key
```

- Start at new leaf, move parent down until we meet a parent smaller than key.
- Running time: \( O(\log n) \)
Recitation 3: Heapsort, Priority Queue

Re-establish heap property

Priority Queue

- Data structure for maintaining a set of elements, each with an associated value called a key.
- Example: scheduling jobs on a shared computer
  - When job finished, highest priority job is selected to be executed.
- One of the most popular application of heap
  - Insert new element in $O(\lg n)$
  - Extract element with smallest (largest) key in time $O(\lg n)$.

Heapsort

$\text{HEAPSORT}(A)$
1. $\text{BUILD-MAX-HEAP}(A)$
2. for $i \leftarrow \text{heap-size}(A)$ down to 2
3. do $\text{swap} A[i] \leftrightarrow A[i']$
4. heap $\leftarrow \text{heap-size}(A) - 1$
5. $\text{MAX-HEAPIFY}(A, i)$

- Use min-heap to get decreasing seq
- Sorts in place
- Running time = $O(n \log n) + \text{Build-max-heap time}$.

Build-Min-Heap

$\text{BUILD-MIN-HEAP}(A)$
1. for $i \leftarrow \lceil \text{heap-size}(A)/2 \rceil$ down to 1
2. do $\text{MIN-HEAPIFY}(A, i)$

- Correctness:
  - Invariant: All trees rooted at $m\geq i$ are heaps
- Running time:
  - Height of tree is $\lfloor \lg n \rfloor$
  - Number of nodes at height $h$ is at most $\lceil n/2^h \rceil$
  - $\text{MIN-HEAPIFY}$ runs in time $O(h)$ when the node is of height $h$.

Summary

- Heap data structure
  - nearly complete binary tree
  - $A[\text{parent}(x)] \leq A[x]$
- Extract min, insert
  - $O(\lg n)$
- Heapsort
  - $O(n \lg n)$
  - in place sort
- Priority queue
  - popular application of heap - $O(\lg n)$ insert and extract min