

Heapsort and Priority Queues

CSE 680

Suggested reading: Chapter 6

1 Heaps

- Complete binary tree
- Nearly complete binary tree
- A heap is an array $A[1..n]$ or its subarray viewed as a (nearly) complete binary tree.
- $\text{heap-size}[A]$: number of elements in the heap.
- $\text{parent}(i) = \lfloor i/2 \rfloor$
- $\text{left}(i) = 2i$
- $\text{right}(i) = 2i + 1$
- Max-heap: $A[\text{parent}(i)] \geq A[i]$, for all $2 \leq i \leq \text{heap-size}[A]$
- Min-heap: $A[\text{parent}(i)] \leq A[i]$, for all $2 \leq i \leq \text{heap-size}[A]$

2 Max-Heapify

Problem: In a heap, suppose the subtrees rooted at $\text{left}(i)$ and $\text{right}(i)$ are max-heaps. We want to make the tree rooted at i a max-heap.

procedure MaxHeapify(A, i)

$l \leftarrow \text{left}(i)$

$r \leftarrow \text{right}(i)$

$size \leftarrow \text{heap-size}[A]$

$largest \leftarrow i$

if ($l \leq size$) **and** ($A[l] > A[largest]$) **then** $largest \leftarrow l$

if ($r \leq size$) **and** ($A[r] > A[largest]$) **then** $largest \leftarrow r$

if ($largest \neq i$) **then**

 exchange $A[i] \leftrightarrow A[largest]$

 MaxHeapify($A, largest$)

Running time: $O(\log n)$.

3 Build a heap

Problem: Convert an array $A[1..n]$ of integers into a max-heap.

```
procedure Build-Max-Heap( $A[1..n]$ )  
    heap-size[ $A$ ]  $\leftarrow n$   
    for  $i \leftarrow \text{parent}(n)$  downto 1 do  
        MaxHeapify( $A, i$ )
```

Running time: $O(n \log n)$.

4 Heapsort

Problem: Sort an array $A[1..n]$.

```
procedure Heapsort( $A[1..n]$ )  
  Build-Max-Heap( $A[1..n]$ )  
  for  $i \leftarrow n$  downto 2 do  
    exchange  $A[1] \leftrightarrow A[i]$   
    decrease heap-size[ $A$ ] by 1  
    MaxHeapify( $A, 1$ )
```

Running time: $O(n \log n)$.

5 Priority Queues

- A data structure for maintaining a set S of elements with the following operations:
 - $\text{Maximum}(S)$: returns the element in S with the largest key.
 - $\text{Insert}(S, x)$: inserts an element x into S .
 - $\text{Extract-Max}(S)$: removes and returns the element in S with the largest key.
- The following function is useful in many applications of priority queue.
 - $\text{Increase-Key}(S, i, k)$: increases i 's key to k .
- Basic idea: use a max-heap.
- All the above operations can be done in $O(\log n)$ time, where n is the size of S .

procedure $\text{Increase-Key}(A[1..n], i, k)$

/* increase $A[i]$ to k */

if $(k < A[i])$ **then error** “new key smaller than current key”

$A[i] \leftarrow k$

while $(i > 1)$ **and** $(A[\text{parent}(i)] < A[i])$ **do**

 exchange $A[i] \leftrightarrow A[\text{parent}(i)]$

$i \leftarrow \text{parent}(i)$