Priority Queues
Heaps and Heapsort

Reading Assignment
Chapter 7

Priority Queue

- A priority queue stores a collection of prioritized elements
- Applications
  - 911 event queues
  - Airport landing patterns
  - Priority check in at the airport
  - Triage in a hospital
  - Plane sweep algorithms
- Operations
  - **`insert(), deleteMin()`**
  - **`deleteMin()`** or **`deleteMax()`** but not both
  - Note that **`member(), search()`** or **`find()`** are not supported
- Implementation strategies
  - Linear lists or sequences
  - Heaps

Priority Queue Interface

```
public interface PriorityQueue {
   void insert(Object x);
   Object deleteMin(); // or deleteMax() instead
   Object getMin(); // gets min but does not delete it
   int size();
   boolean isEmpty();
}
```

- Instead of Object, the priority queue interface might also store elements or associations
- To compare elements a Comparator class can be used

Integer Priority Queue Interface

- Assume an integer Priority Queue interface **`IntPQ`** to simplify the discussion and presentation

```
public interface IntPQ {
   void insert(int x);
   int deleteMin(); // or deleteMax() instead
   int getMin(); // gets min but does not delete it
   int size();
   boolean isEmpty();
}
```

Priority Queue Sort

- The priority queue operations allow for a simple sorting algorithm

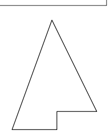
```
void pqSort(int a[]) {
   IntPQ pq = new IntPQ();
   for (int k=0; k<a.length; k++) { // first loop
      pq.insert(a[k]);
   }

   k = 0;
   while (!pq.empty() { // second loop
      a[k] = pq.deleteMin();
      k++;
   }
}
```


Time Complexity of PQ Operations

- How can we implement the Priority Queue operations efficiently?
- Running time analysis of `pqSort()` assuming n input values
- **First loop**
  - $T_{fl}(n) = n * T(insert)$
- **Second loop**
  - $T_{sl}(n) = n * T(deleteMin)$
- **Total**
  - $T_{pq}(n) = T_{fl}(n) + T_{sl}(n) = n * T(insert) + n * T(deleteMin) =$
  - $T_{pq}(n) = n * \{T(insert) + T(deleteMin)\}$

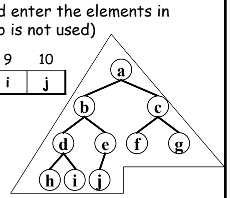
- **Linked list implementation**
  - Linked list is sorted at insert time
  - T(insert) = ϵ O(n)
  - T(delete) = ϵ O(1)
  - $T_{pq}(n) \in O(n^2) + O(n) \in O(n^2)$ ☹ ☹

Heap Encoding



- Array representation
- Assume complete binary tree
  - All levels are full except possibly the last level
  - No holes
  - *Heap shape property*
- Heap encoding
  - Process the binary tree in level order and enter the elements in an array starting with array index 1 (zero is not used)

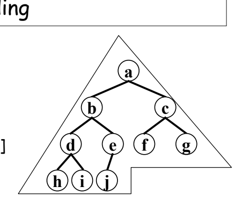
0	1	2	3	4	5	6	7	8	9	10
	a	b	c	d	e	f	g	h	i	j



- Parent of a[k] is at a[k/2]
- Left child of a[k] is at a[2k]
- Right child of a[k] is at a[2k+1]

Heap Encoding

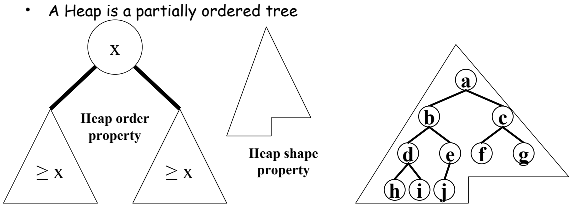
- Parent of a[5] is at a[5/2] = a[2]
  - Parent of "e" is "b"
- Left child of a[3] is at a[2*3] = a[6]
  - Left child of "c" is "f"
- Right child of a[3] is at a[2*3+1] = a[7]
  - Right child of "c" is "g"



0	1	2	3	4	5	6	7	8	9	10
	a	b	c	d	e	f	g	h	i	j

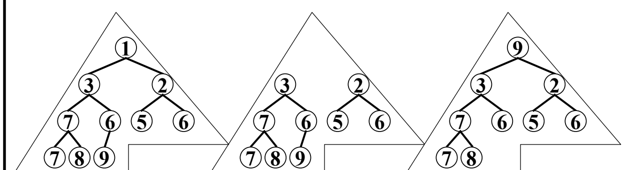
Heap Properties

- Shape property
  - All levels in a heap are complete except possibly the last level.
- Order property
  - A heap is a binary tree in which the nodes are labelled with elements of a set such that all elements in the left and right subtrees of a node labelled x are greater than or equal to x.
- A Heap is a partially ordered tree



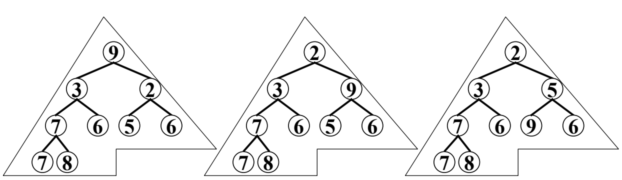
DeleteMin

- The smallest element is the root node
- Remove and return root node which is constant time O(1)
- Re- establish shape property
  - Move last element in the tree to the root
  - Except for the root node, order is fine too
- Re-establish order property
  - Push the root element, which is out of order, down by swapping elements until order property is established



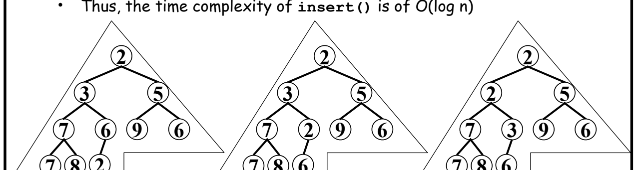
Pushdown — Bubbling down

- Push element 9 down until Heap order property is re-established
- Keep on swapping with the smallest child
- At most log n swap operations (i.e., # of levels)
- Thus, the time complexity of `pushdown()` is of O(log n)
- The time complexity of `deleteMin()` is also of O(log n)



Insert — Bubbling up

- Insert the element at the first open array position
- Shape property is trivially established
- Push the element up the tree until the order property is re-established by swapping with the parent
- At most log n swap operations (i.e., # of levels)
- Thus, the time complexity of `insert()` is of O(log n)



Interface Priority Queue

```
public interface PriorityQueue {
  Object deleteMin();
  Object getMin();
  void insert(int key, String data);
  boolean isEmpty();
  int size();
}
```


Class Priority Queue

```
public class PQ implements PriorityQueue {
   private int size;
   private Node[] heap;
   private final static int defaultPQSize = 30;
   private final static int rootIndex = 1;

  public PQ() {
    this(defaultPQSize);
  }
  public PQ(int pqSize) {
    size = 0;
    heap = new Node[pqSize];
  }
}
```


Insert Implementation

```
public void insert(int key, String data) {
   size++;
   Node p = new Node(key, data);
   heap[size] = p;
   if (size > 1) pushup();
}
```


DeleteMin Implementation

```
public Object deleteMin() {
   if (size == 0) {
       return null;
   } else {
       Node p = heap[rootIndex];
       if (size == 1) {
              heap[rootIndex] = null;
              size--;
       } else { // size > 1
              heap[rootIndex] = heap[size];
              heap[size] = null;
              size--;
              pushdown();
       }
       return p;
   }
}
```

Integer Heap Interface

```
public interface IntHeap {
   void insert(int x);
   int deleteMin(); // or deleteMax() instead
   int getMin(); // gets min but does not delete it
   int size();
   boolean isEmpty();
}
```


Heapsort

```
void heapSort(int a[]) {
   IntHeap heap = new IntHeap();
   for (int k=0; k<a.length; k++) {
      heap.insert(a[k]);
   }
   k = 0;
   while (!heap.empty() {
      a[k] = heap.deleteMin();
      k++;
   }
}
```

- **insert()** and deleteMin() each take O(log n) time
- The running time of Heapsort is $T_{hs}(n) = n \log n + n \log n = 2 n \log n$
- Hence the time complexity of Heapsort is of O(n log n)
- Fundamental result of Computer Science
  - Sorting takes O(n log n) time

Summary

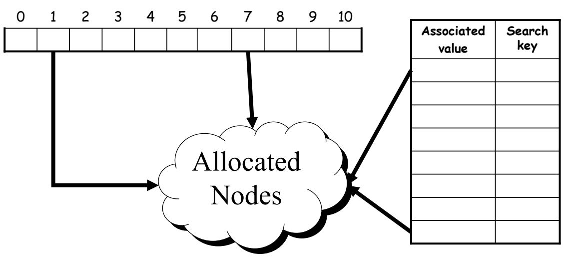
- PriorityQueue
  - **insert(), deleteMin() (or deleteMax())**
  - Applications
  - Implementation strategies: list or heap
- PriorityQueue Sort
  - Using linear list data structure $O(n^2)$
- Heap
  - Encoding of a binary tree in an array
  - Shape and order property
- **deleteMin()**
  - Remove min (root); bubble down by swapping
- **insert()**
  - Insert at the end of array; bubble up by swapping
- Heapsort
  - Using heap data structure O(n log n)

Assignment 5

- Priority Queue using heap
- Hashtable

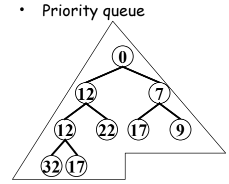
Searchable Priority Queue

- Heap User access O(log n)
  - insert()
  - deleteMin()
- Hashtable user access O(1)
  - insert()
  - search()



Searchable Priority Queue Example

- Priority queue



- Heap

0	1	2	3	4	5	6	7	8	9	10
	0	**12**	**7**	**12**	**22**	**17**	**9**	**32**	**17**	

12	Hausi
32	Bette
17	Peggy
22	Carmen
7	Daniel
0	Jens
9	Ulrike
17	Dale
12	Frank