Overview

- Heaps
  - Definitions
  - Properties
  - Insertion / Deletion
  - Implementation
    - Vector
  - Heap Sort

- Dictionaries
Heaps

- A heap is a binary tree $T$ that stores a collection of keys (or key-element pairs) at its internal nodes and that satisfies two additional properties:
  - **Order Property**: $\text{key(parent)} \leq \text{key(child)}$
  - **Structural Property**: all levels are full, except the last one, which is left-filled (*complete binary tree*)
Not Heaps

- bottom level is not left-filled

- key(parent) > key(child)
Height of a Heap

A heap $T$ storing $n$ keys has height $h = \lfloor \log(n + 1) \rfloor$, which is $O(\log n)$.

- $n \geq 1 + 2 + 4 + \ldots + 2^{h-2} + 1 = 2^{h-1} - 1 + 1 = 2^{h-1}$
  - For $n = 1$, $h = 1$,
  - For $n = 2$, $h = 2$,
  - For $n = 4$, $h = 3$,
  - For $n = 8$, $h = 3$,
  - For $n = 16$, $h = 4$,

- $n \leq 1 + 2 + 4 + \ldots + 2^{h-1} = 2^h - 1$
  - For $n = 16$, $h = 4$,
  - For $n = 25$, $h = 5$,
  - For $n = 32$, $h = 5$,

- Therefore $2^{h-1} \leq n \leq 2^h - 1$

- Taking logs, we get $\log (n + 1) \leq h \leq \log n + 1$

- Which implies $h = \lfloor \log(n+1) \rfloor$
Heap Insertion

So here we go ...

The key to insert is 6
Heap Insertion

Add the key in the *next available position* in the heap.

Now begin *Upheap*.
Upheap

• Swap parent-child keys out of order

Heaps I

6.7
Upheap Continues

Heaps I
• **Upheap** terminates when new key is greater than the key of its parent or the top of the heap is reached

• \((\text{total swaps}) \leq (h - 1)\), which is \(O(\log n)\)
Removal From a Heap

**RemoveMin()**

- The removal of the top key leaves a hole
- We need to fix the heap
- First, replace the hole with the last key in the heap
- Then, begin *Downheap*
Downheap compares the parent with the smallest child. If the child is smaller, it switches the two.
Downheap Continues

Heaps I
Downheap Continues
• **Downheap** terminates when the key is greater than the keys of both its children or the bottom of the heap is reached.

• (total #swaps) \( \leq (h - 1) \), which is \( \mathcal{O}(\log n) \)
Implementation of a Heap

```java
public class HeapPriorityQueue implements PriorityQueue {
    BinaryTree T;
    Position last;
    Comparator comparator;
    ...
}
```
Implementation of a Heap (cont.)

- Two ways to find the insertion position $z$ in a heap:

```
(2,C)
/    \
(5,A)  (4,C)
|     |
(15,K) (9,F) (7,Q) (6,Z)
|     |
(16,X) (25,J) (14,E) (12,H) (11,S) (8,W) (20,B) (10,L)
```

```
(4,C)
/    \
(5,A)  (6,Z)
|     |
(15,K) (9,F) (7,Q) (20,B)
|     |
(16,X) (25,J) (14,E) (12,H) (11,S) (8,W)
```

Heaps II 6.3
Vector Based Implementation

- Updates in the underlying tree occur only at the “last element”

- A heap can be represented by a vector, where the node at rank $i$ has
  - left child at rank $2i$ and
  - right child at rank $2i + 1$

- The leaves do no need to be explicitly stored

- Insertion and removals into/from the heap correspond to `insertLast` and `removeLast` on the vector, respectively
Heap Sort

• All heap methods run in logarithmic time or better

• If we implement PriorityQueueSort using a heap for our priority queue, insertItem and removeMin each take $O(\log k)$, $k$ being the number of elements in the heap at a given time.

• We always have at most $n$ elements in the heap, so the worst case time complexity of these methods is $O(\log n)$.

• Thus each phase takes $O(n \log n)$ time, so the algorithm runs in $O(n \log n)$ time also.

• This sort is known as heap-sort.

• The $O(n \log n)$ run time of heap-sort is much better than the $O(n^2)$ run time of selection and insertion sort.

In-Place Heap-Sort

• Do not use an external heap

• Embed the heap into the sequence, using the vector representation
The Dictionary ADT

• a dictionary is an abstract model of a database

• like a priority queue, a dictionary stores key-element pairs

• the main operation supported by a dictionary is searching by key

• simple container methods:
  - size()
  - isEmpty()
  - elements()

• query methods:
  - findElement(k)
  - findAllElements(k)

• update methods:
  - insertItem(k, e)
  - removeElement(k)
  - removeAllElements(k)

• special element
  - NO_SUCH_KEY, returned by an unsuccessful search
Implementing a Dictionary with a Sequence

• unordered sequence

- searching and removing takes $O(n)$ time
- inserting takes $O(1)$ time
- applications to log files (frequent insertions, rare searches and removals)

• array-based ordered sequence (assumes keys can be ordered)

- searching takes $O(\log n)$ time (binary search)
- inserting and removing takes $O(n)$ time
- application to look-up tables (frequent searches, rare insertions and removals)