Java 2 Binary Heaps

- This is not a search structure.
  - Designed for selection

- Conceptually similar to a stack or queue
- A **priority queue** chooses the next element to delete based on priority.
  - Priority is some value associated with the element that could represent importance, cost, or some other problem-specific concept

PQ Method

- add
  - unsorted list
    - O(1)
  - sorted list
    - O(N)
  - balanced but
    - O(logN)
  - Binary heap
    - O(logN)
- remove
  - unsorted
    - O(N)
  - sorted
    - O(1)
  - balanced but
    - O(logN)
  - binary heap
    - O(logN)
- peek
  - unsorted
    - O(N)
  - sorted
    - O(1)
  - balanced but
    - O(logN)
  - binary heap
    - O(1)

- Heap usually implies an array based implementation is being used.

Binary Heap

- A binary heap is a **complete binary tree**
  - Which lends itself to an array-based implementation
  - Height is O(LogN)
- Every node obeys a **partial order property**
  - All ordering in this tree is vertical

Array-Based Implementation

- Binary heaps are almost always implemented as an array because:
  - Acceptably space efficient
  - Easy traversal: parent to child via multiplication, child to parent via division.

Binary Heap insertion

1. Insert the new element in the one and only one location that will maintain the complete shape.
2. Swap values as necessary on a leaf-to-root path to maintain the partial order.

Binary Heap Deletion

1. Maintain the complete shape by replacing the root value with the value, in the lowest right-most leaf. Than delete that leaf.
2. Swap values as necessary on a root-to-leaf path to maintain the partial order.

Heapsort

- Optimal sort
- Heap sort work in two phases:
  - The initial arbitrary order of the array is transformed into a partial order
  - The partial order is transformed into a total order.
  - This allows you to constantly heapify the smallest element.
- Beginning with the lowest, right-most parent and continuing to the root, heapify each subtree.
- Heapsort is not stable and it typically has larger constant factors than quick sort.

Huffman's Algorithm

- Variable-length codes
  - Number of bits per character determined by the char's relative frequency of occurrence.
    - Most frequently occurring characters should use the fewest bits.

Variable-Length Codes

- Number of bits per character determined by the char's relative frequency of occurrence
- Huffman's algorithm builds a tree from the bottom up with all of the alphabet of the most frequent characters.
  - The code for one character can't be a prefix of another character's code.
- Huffman's algorithm wants a code tree with minimum expected code length.

– $L(C) = \sum w * length(c)$

- Which is just a weighted average of all possible character code lengths.

create a single node code tree for each character and insert each of these trees into a priority queue (min heap).

```
while (pq has more than one element) {
      c1 = pq.deletemin();
      c2 = pq.deletemin();
      c3 = new code tree(c1, c2);
      pq.add(c3);
}
```