Binary heap data structure

- A binary heap is a special kind of binary tree
  - has a restricted structure (must be complete)
  - has an ordering property (parent value is smaller than child values)
  - NOT a Binary Search Tree!
- Used in the following applications
  - Priority queue implementation: supports enqueue and deleteMin operations in $O(\log N)$
  - Heap sort: another $O(N \log N)$ sorting algorithm (see last slide).

Complete Binary Trees

- A complete binary tree can be easily stored in an array
  - place the root in position 1 (for convenience)
**Complete Binary Trees**

**Properties**

- The height of a complete binary tree is \(\text{floor}(\log_2 N)\) (floor = biggest int less than)
- In the array representation:
  - put root at location 1
  - use an int variable (size) to store number of nodes
  - for a node at position \(i\):
    - left child at position \(2i\) (if \(2i \leq \text{size}\), else \(i\) is leaf)
    - right child at position \(2i+1\) (if \(2i+1 \leq \text{size}\), else \(i\) is leaf)
    - parent is in position \(\text{floor}(i/2)\) (or use integer division)

**Binary Heap:**

**ordering property**

- In a heap, if \(X\) is a parent of \(Y\), value(X) is less than or equal to value(Y).
- the minimum value of the heap is always at the root.
- findMin() is \(O(1)\)

**Heap: insert(x)**

- First: add a node to tree.
  - Maintain a complete tree: place item at next available location, size+1
- Next: maintain the ordering property:
  - if \(x\) is greater than its parent: done
  - else swap with parent, repeat
- Called “percolate up” or “reheap up”
- preserves ordering property
- \(O(\log N)\) worst case
Heap: deleteMin()

- Minimum is at the root, removing it leaves a hole.
- First: maintain complete tree: move last element up to the root.
- Next: maintain the ordering property, start with root:
  - if both children are greater than the parent: done
  - otherwise, swap the smaller of the two children with the parent, repeat (why not the larger one?)
- Called “percolate down” or “reheap down”
- preserves ordering property
- \(O(\log N)\) worst case

Heap: buildHeap()

- buildHeap takes a tree that does not have heap order and establishes it.
- The algorithm works bottom-up:
  - when processing a given node, its two children will already be in heap order.
  - then we can use percolate down to put the current node in the right place, and preserve the heap order property.
- No need to apply to leaves.
- Turns out this algorithm is \(O(N)\) (see book for proof)
- \(N\) inserts using insert\((x)\) would be \(O(N \log N)\) worst case.
Heap: buildHeap()

Using a heap to sort a list:
1. insert every item into a binary heap
2. extract every item by calling deleteMin N times.

Can make it slightly more efficient by using buildHeap on the unsorted vector instead of using insert N times.

Runtime Analysis:  O(N log N)
- step 1 is O(N) if you use buildHeap
- step 2: deleteMin is O(log N), and it’s done N times, so it’s O(N log N).

Heapsort

- Using a heap to sort a list:
  1. insert every item into a binary heap
  2. extract every item by calling deleteMin N times.
- Can make it slightly more efficient by using buildHeap on the unsorted vector instead of using insert N times.
- Runtime Analysis:  O(N log N)
  - step 1 is O(N) if you use buildHeap
  - step 2: deleteMin is O(log N), and it’s done N times, so it’s O(N log N).