Lecture 3 – Priority Queues
Reading: KT Section 2.5

A few reminders

• Disabilities and accommodations

• Assignments
  • Collaboration
  • Resources to use
  • Struggle time
  • Late passes

• Come talk to me!
  • About anything 😊
Priority queues, which data structure should we use?

- It’d be great if we can keep the elements sorted.

- Array?

- Linked list?

- Binary Tree? Binary Search Tree?
But then... what kind of tree is this?

- A **minheap** is a complete binary tree,
  - each element is less than or equal to both of its children
- A minheap keeps the smallest valued element readily available

---

Heaps, which data structure should we use?

---

1. 1
2. 2
3. 3
4. 7
5. 5

- 10
  - 15
  - 17
- 20
- 9
- 15
- 8
- 16
Finding child nodes

Children of H[3] are at H[?] and H[?]

Finding parent nodes

Parent of H[10] is at H[?]
Adding a new element

Heapify-up(H, 15)
Heapify-up(H, 7)

Heapify-up(H, 3)
Algorithm 2.8, page 61

Heapify-up(H,i):
  If i > 1 then
    let j = parent(i) = \lfloor i/2 \rfloor
    If key[H[i]] < key[H[j]] then
      swap the array entries H[i] and H[j]
      Heapify-up(H,j)
    Endif
  Endif

Why does this work?

Deleting an element

![Diagram showing the process of deleting an element from a heap]
Deleting an element

Heapify-down(H, 3)
Algorithm 2.9, page 63

Heapify-down(H, i):
Let $n = \text{length}(H)$
If $2i > n$ then
    Terminate with $H$ unchanged
Else if $2i < n$ then
    Let left = $2i$, and right = $2i + 1$
    Let $j$ be the index that minimizes key[$H$[left]] and key[$H$[right]]
Else if $2i = n$ then
    Let $j = 2i$
Endif
If key[$H$[j]] < key[$H$[i]] then
    swap the array entries $H[i]$ and $H[j]$
    Heapify-down($H$, j)
Endif

Why does this work?