Lecture 14: Complexity & Heaps

- Review/catch-up for prof. C.
- Heap data structure

Announcements:
- P3 extension to tomorrow @ 5PM
- Poll/survey coming soon.
$f$ is $O(g)$ means:
- want to ignore constant speedups
  (e.g. run alg. on comp. that's $2x$ faster)
- ignore what happens for small inputs.
  (no's trivial optimization)

Consider just upper bounds
(won't take more time than $n$ steps)

$f$ is $O(g)$ means
- there exists constants $n_0, c$
  such that
- for any $n > n_0$
  $f(n) \leq cg(n)$

\[ n \]
Divide & Conquer

for each input:
do something
in constant time.

for each input:
for each input:
do something
(prohibitively expensive)

Search or Optimization

\[ n \log n \rightarrow \text{sorting} \]

\[ \log_2 n \rightarrow \text{log2 of digits} \]

\[ n \rightarrow \text{# of digits} \]

\[ n \rightarrow \text{size of digits} \]

Input: size 10,000

- Split into 10,
  recursively handle one
  (size 1,000)

- Split again
  (size 100)

- Split again
  (size 10)

- Base case
  (size 1)
Priority Queue:
- collection
  - add an item with a priority
  - remove and return the item
    with maximum priority.

Implementation:
- linked list (store (priority, item) pairs)
  - Item 1, priority 7
  - Item 2, priority 5
  - add an item: put at beginning
    (constant time)
  - find/remove max priority:
    search through list,
    $O(n)$ time.

- ordered linked list
  - maximum priority element is always
    at head; find/remove max is
    constant time.

- binary search tree

- sorted array
  - same issues as above
  - need to move elements over
    to make space for new elt

  - linear time in worst case
    (unbalanced)
  - hard (expensive) to maintain
    balance.

Linear time in worst case

Linear time insert, but can use
binary search.
A heap is a binary search tree with the following invariants:

1. The value of every node is greater than (or equal to) the priority of both of its children.
   (Note: no requirement between siblings.)

2. The tree is full.
   - Perfect: all paths have the same length
   - Full: all levels are full, except possibly the last, which is filled left-to-right.
How to

- find largest element in a heap?
  - it's at the top!

- remove largest element in a heap?

- insert a new element?

1. node larger than its children
2. heap is full.

Insert value 12
(ignoring 1st invariant)

Swap parent of new node & its two children
so biggest is at top.

Step to maintain invariant

Since 12 < 17,
both invariants satisfied.