Today’s Outline

- Finish Asymptotic Analysis
- Questions
- Trees Review
- Priority Queues
- Heaps
- d-Heaps

Simplifying Recurrences

1. Given some equation for the running time:
   e.g. \( T(n) = \log \lfloor n/2 \rfloor \)

2. Solve the recursive equation
   - For an upper-bound analysis, you can optionally simplify the equation to something larger
     e.g. \( T(n) = T(\lfloor n/2 \rfloor) + 1 \neq T(n) < T(n)/2 + 1 \)
   - For a lower-bound analysis, you can optionally simplify the equation to something smaller
     e.g. \( T(n) = 2T(n/2) + 1 \Rightarrow T(n) > 2T(n/2) + 1 \)

The One Page Cheat Sheet

- Calculating series:
  e.g. \( \sum_{i=1}^{n} i(n-1) \)

- Solving recurrences:
  e.g. \( T(n) = T(n/2) + 1 \)

  1. Brute force (Section 1.2.3)
  2. Induction (Section 1.2.5)
  3. Memorize simple ones!

- General proofs (Section 1.2.5)
  e.g. How many edges in a binary tree?
       1. Counterexample
       2. Induction
       3. Contradiction

(we’ll see more examples coming up)

Tree Review

root: A
leaf: B, C
child: D, E, F
parent: A, B, C, E
sibling: B, D, E
ancestor: A, B
descendent: H, I
subtree:

More Tree Terminology

depth:
depth of B:
height:
height of A:
degree:
degree of A:
branching factor:
One More Tree Terminology Slide

- Binary:
- N-ary:
- Complete:

Back to Queues

- Some applications
  - ordering CPU jobs
  - simulating events
  - picking the next search site
- Problems?
  - short jobs should go first
  - earliest (simulated time) events should go first
  - most promising sites should be searched first

Priority Queue ADT

- Priority Queue operations
  - create
  - destroy
  - insert
  - deleteMin
  - isEmpty
- Priority Queue property: for two elements in the queue, x and y, if x has a lower priority value than y, x will be deleted before y

Applications of the Priority Q

- Hold jobs for a printer in order of length
- Store packets on network routers in order of urgency
- Simulate events
- Select symbols for compression
- Sort numbers
- Anything greedy

Naïve Priority Q Data Structures

- Unsorted array:
  - insert:
  - deleteMin:
- Sorted array:
  - insert:
  - deleteMin:

Binary Search Tree Priority Q Data Structure (that’s a mouthful)

- Unsorted array:
  - insert:
  - deleteMin:
- Sorted array:
  - insert:
  - deleteMin:

Problems
1.
2.
Binary Heap Priority Q Data Structure

- Heap-order property
  - parent's key is less than children's keys
  - result: minimum is always at the top
- Structure property
  - complete tree with fringe nodes packed to the left
  - result: depth is always $O(\log n)$; next open location always known

How do we find the minimum?

Nifty Storage Trick

- Calculations:
  - child:
  - parent:
  - root:
  - next free:

DeleteMin

```java
Object deleteMin() {
    assert(!isEmpty());
    returnVal = Heap[1];
    size--;
    newPos = percolateDown(1, Heap[size + 1]);
    Heap[newPos] = Heap[size + 1];
    return returnVal;
}
```

Percolate Down – Basic

```
def percolateDown(int hole, Object val):
    while (2*hole <= size):
        left = 2*hole;
        right = left + 1;
        if (right <= size and Heap[right] < Heap[left]):
            target = right;
        else:
            target = left;
        if (Heap[target] < val):
            Heap[hole] = Heap[target];
            hole = target;
        else:
            break;
    return hole;
```

DeleteMin Code (Optimized)

```
int percolateDown (int hole, Object val) {
    while (2*hole <= size) {
        left = 2*hole;
        right = left + 1;
        if (right <= size && Heap[right] < Heap[left]) {
            target = right;
        } else {
            target = left;
        }
        if (Heap[target] < val) {
            Heap[hole] = Heap[target];
            hole = target;
        } else {
            break;
        }
    }
    return hole;
}
```
**Insert Code**

```java
void insert(Object o) {
    assert(!isFull());
    size++;
    newPos = percolateUp(size, o);
    Heap[newPos] = o;
}
```

**Other Priority Queue Operations**

- **decreaseKey**
  - given a pointer to an object in the queue, reduce its priority value

  Solution: change priority and ____________________________

- **increaseKey**
  - given a pointer to an object in the queue, increase its priority value

  Solution: change priority and ____________________________

**BuildHeap**

Floyd’s Method. Thank you, Floyd.

Pretend it's a heap and fix the heap-order property!

**Still More Priority Queue Operations**

- **remove**
  - given a pointer to an object in the queue, remove it

  Solution: set priority to negative infinity, percolate up to root and deleteMin

- **buildHeap**

  Naive solution:

  Running time:

  Can we do better?
Thinking about Heaps

- Observations
  - finding a child/parent index is a multiply/divide by two
  - operations jump widely through the heap
  - each operation looks at only two new nodes
  - inserts are at least as common as deleteMins
- Realities
  - division and multiplication by powers of two are fast
  - looking at one new piece of data sucks in a cache line
  - with huge data sets, disk accesses dominate

Solution: d-Heaps

- Each node has $d$ children
- Still representable by array
- Good choices for $d$:
  - optimize performance based on # of inserts/removes
  - choose a power of two for efficiency
  - fit one set of children in a cache line
  - fit one set of children on a memory page/disk block

Does this help insert or deleteMin more?

One More Operation

- Merge two heaps. Ideas?

To Do

- Finish Homework #1
  - Start Homework #2 if you’ve already finished
- Read chapter 6 in the book
Coming Up

• Mergable Priority Q’s
• Leftist heaps
• Skew heaps

• No class on July 4!!!