ECE 190KL

L20pA
1 Mar '13

binary search bug

heaps
use as priority queues
recursion ← Monday

for Prog 7 you will write tests

---

binary search bug

int middle = ~~low~~ (low + high) / 2;

What happens on overflow?

In textbooks, etc, for ~30 years until
Java library impl broke on a Google problem...

L20 P1

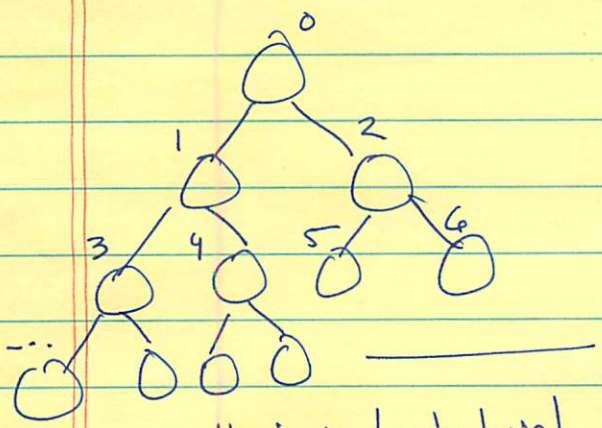
(based on CLR Ch. 7)

note: heap data structure ≠ heap of memory for dynamic allocation

binary heap data structure
- reasonably fast sorting
- good priority queue
- no dynamic memory (just an array)

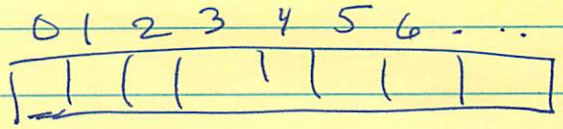
two ways to view a heap

logically, as an almost-complete binary tree




- complete in all but last level
- last level filled from left

physically, as an array




given index N, how do we get parent, left, right children[ask]?

left(N) = 2*N+1

right(N) = 2*N+2

parent(N) = $\left\lfloor \frac{N-1}{2} \right\rfloor$ (floor is automatic with ints in C)

Data in heap are organized according to the heap property:

For every node N other than 0 (the root),

$$heap[parent(N)] \geq heap[N]$$

We'll start with an important operation: heapify

Say that we have a node N for which ~~the left and right~~ the sub-trees rooted at its left and right children obey the heap property, but the node itself may be smaller than one/both children.

Heapify checks for and fixes this problem so that the subtree rooted at the node obeys the heap property.

```
void heapify (int heap[], int len, int N)
{
    int left, right, largest, swap;

    while (len > N) {
        left = 2*N + 1;
        right = left + 1;
        if (len > left && heap[left] > heap[N]){
            largest = left;
        } else {
            largest = N;
        }
        if (len > right && heap[right] > heap[largest]){
            largest = right;
        }
        if (largest == N) {
            return;
        }
        swap = heap[N];
        heap[N] = heap[largest];
        heap[largest] = swap;
        N = largest;
    }
}
```

How many iterations of this loop can execute?
Can you bound in terms of len?
N ≥ 0 → N ≥ 1 → N ≥ 2 → ...

How can we turn an array into a heap?

First, notice that all elements in second half of array are already heaps!

(no children → satisfies heap property)

$2 * N + 1 \geq len$

$N \geq \frac{len - 1}{2}$ (but we don't want to round down!)

$\Rightarrow N \geq \frac{len}{2}$ in C

example: len = 10 → node 4 has node 9 as left child

```
void buildheap(int heap[], int len)
{
    int i;

    for (i = (len/2)-1; 0 <= i; i--){
        heapify (heap, len, i);
    }
}
```

Note that order moves up tree, so heapify assumptions satisfied for each call.

How long does building a heap take?

# iterations in heapify $\leq$ height of tree (from starting point of call)

Roughly ...

$\frac{1}{2}$ nodes at height 1 (no call actually made)

$\frac{1}{4}$ nodes at height 2

⋮

len $(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots)$ iterations at height 1 $\Rightarrow$ len

+ len $(\frac{1}{4} + \frac{1}{8} + \dots)$ iterations at height 2 $\Rightarrow$ $\frac{1}{2}$ len

+ len $(\frac{1}{8} + \dots)$ iterations at height 3 $\Rightarrow$ $\frac{1}{4}$ len

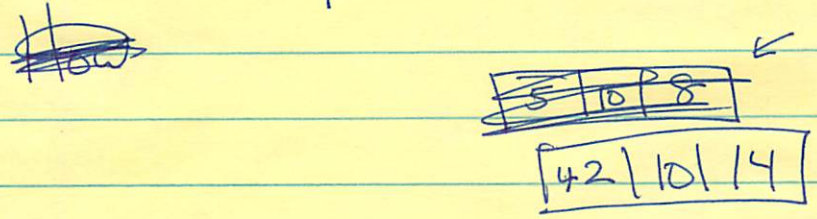
+ ⋯ + ⋯

$(1 + \frac{1}{2} + \frac{1}{4} + \dots)$ len

Linear in length of array.

$\Rightarrow$ 2 len

Is a heap sorted? [ask]




How can we sort a heap (say low to high?)

One idea: biggest element is at index 0

pull it out, put at end, heapify, repeat!

```
void heap-sort (int heap[], int len)
{   int i, swap;
    build-heap (heap, len);
    for (i = len-1; 0 < i; i--) {
        swap = heap[0];
        heap[0] = heap[i];
        heap[i] = swap;
        heapify (heap, i-1, 0);
    }
}
```

length parameter leaves sorted portion of array untouched

How long does it take to sort?

len calls to heapify, each with $\leq \lg(\text{len})$ iterations

# Heaps as Priority Queues

What is a priority queue?

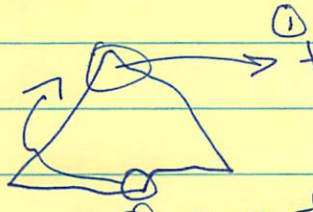
Set of elements with ordered keys

Supports
- insertion of new element
- ~~examining maximum~~
- identifying element with maximal key
- extracting element with maximal key

Useful in (for example) discrete event simulation, where key could be time of event (events create future events)

finding max is easy — heap element 0 is largest

What about removing the max?



How can we insert?

opposite direction of heapify: add to the end and percolate upwards

```
int  <- return new length
int heap_insert(int heap[], int len, int value)
{
    int pos, parent;

    pos = len;
    While (0 < pos){
        parent = (pos -1) /2;
        if (heap[parent] >=value) {
            break;
        {
        heap[pos] = heap[parent];
        pos = parent;
    }
    heap[pos]=value;
    return (len+1);
}
```