binary search bug

heaps

use as priority queues

recursion ← Monday

forprog 7 you will write tests


binary search bug

int middle = (low + high) / 2;

What happens on overflow?

In textbooks, etc., for ~30 years until

Java library impl. broke on a Google problem...
(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

physically, as an array

- Complete in all but last level
- Last level filled from left

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

\[
\begin{align*}
\text{left}(N) &= 2 \times N + 1 \\
\text{right}(N) &= 2 \times N + 2 \\
\text{parent}(N) &= \left\lfloor \frac{N-1}{2} \right\rfloor \quad \text{(floor is automatic with ints in C)}
\end{align*}
\]
Data in heap are organized according to the heap property:

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

\[ \text{heap}[\text{parent}(N)] \geq \text{heap}[N] \]

We'll start with an important operation: heapify.

Say that we have a node N for which the left and right sub-trees rooted at its left and right children obey the heap property, but the node itself may be smaller than one or 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 it in terms of len?

n/2 \to n/2 + 1 \to n/2 \to \ldots
How can we turn an array into a heap?

First, notice that all elements in the second half of array are already heaps!
(no children → satisfies heap property)

\[ 2 \times N + 1 \geq \text{len} \]

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

\[ \Rightarrow N \geq \frac{\text{len}}{2} \quad \text{in C} \]

```c
void build_heap(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 assumption satisfied for each call.
How long does building a heap take?

# iterations in heapify ≤ 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

\( \vdots \)

\[
\text{len} \left( \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots \right) \text{ iterations at height 1} \Rightarrow \text{len}
\]

+ \[
\text{len} \left( \frac{1}{4} + \frac{1}{8} + \ldots \right) \text{ iterations at height 2} \Rightarrow \frac{1}{2} \text{ len}
\]

+ \[
\text{len} \left( \frac{1}{8} + \ldots \right) \text{ iterations at height 3} \Rightarrow \frac{1}{4} \text{ len}
\]

+ \[\vdots\]

\( \overbrace{(1 + \frac{1}{2} + \frac{1}{4} + \ldots)}^{n} \text{ len} \)

Linear in length of array.

\( \Rightarrow 2 \text{ len} \)
Is a heap sorted? [Ask]

How

heap, but not sorted

[42, 10, 14]

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

One idea: biggest element is at end

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; i > 0; i--)
    {
        swap = heap [0];
        heap [0] = heap [i];
        heap [i] = swap;
        heapify (heap, i - 1, 0);
    }
}
```

How long does it take to sort?

len calls to heapify, each with ≤ lg(len) iterations
Heaps as Priority Queues

What is a priority queue?
Set of elements with ordered key

Supports
- insertion of new element
- extracting 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?

1. take it out
2. move the last one up to root
3. heapify!
How can we insert?

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

```c
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);
}
```