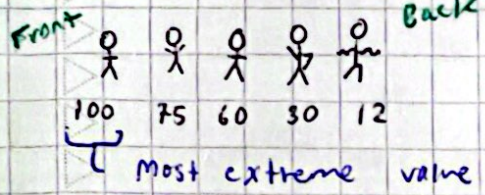
CS-21 LECTURE #3

Heap: Nearly complete binary tree that respects the heap property

heap ---- array → can be used to build structures

Priority Queue / Heapsort → heap ---- array = efficient implementation for heap...

Priority queue:



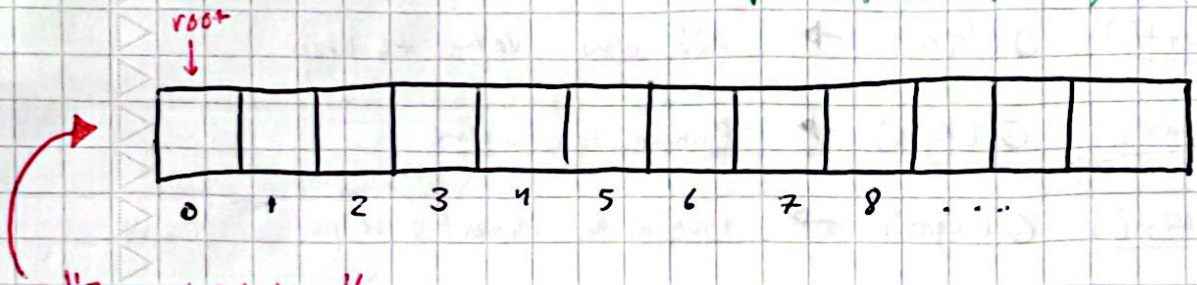
Most extreme value

Min heap: Smallest value at front

max heap: Biggest value at front

a heap ≠ the heap

$A[Parent(i)] \geq A(i)$ Imagine a heap of integers (or objects w/ key value)



"Zero based heap"

root is A[0]

Parent(i) = $(i-1)/2$

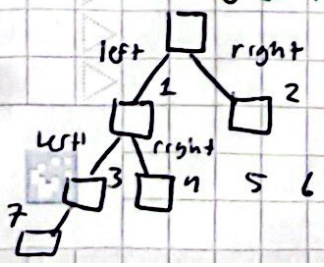
Left(i) = $i*2+1$

Right(i) = $i*2+2$

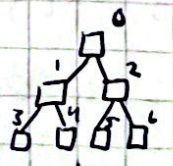
Binary Tree:

Node can have 0, 1, or 2 children

0 ← Array index to store this node...

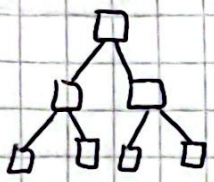


Complete tree = A tree where every level is filled...



↓:

"Height of a node" - Defined as # of edges on longest simple downward path from node to a leaf



Height = 0

HEIGHT = O(lg n)

Basic Heap Operations

Heapify(i) O(lgn) → To maintain heap

Build Heap() O(n) → array → heap

Heap sort() O(nlgn) → heap → sorted array

Priority Queue

- Heap insert() O(lgn) → add new value to heap
- Extract max() O(lgn) → Remove top value

Increase key() O(lgn) → Promote an element; helper to insert

Support Heap Operations

Swap(i,j) → swaps A[i] & A[j]

parent(i)
left(i)
right(i)
} Inline

Index of max(i,j,k)

return index of max value

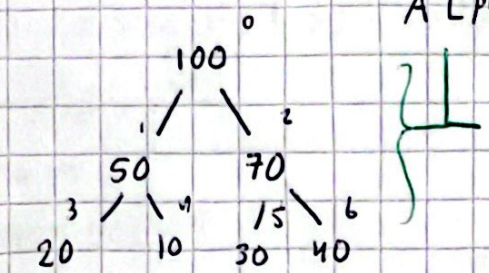
A[i], A[j], A[k]

WITH BOUNDS CHECK

Heapify (i)

assuming that binary heap is rooted at left(i) & right(i) are valid heaps but that node at i might violate the heap property

$A[parent(i)] \geq A(i)$

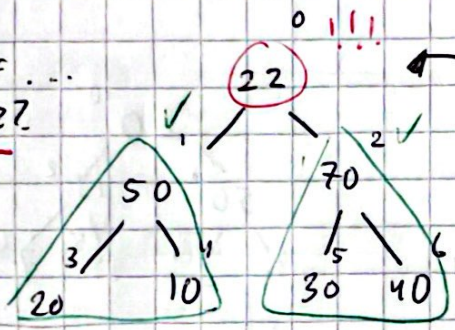


A valid heap!

=

100	50	70	20	10	30	40
0	1	2	3	4	5	6

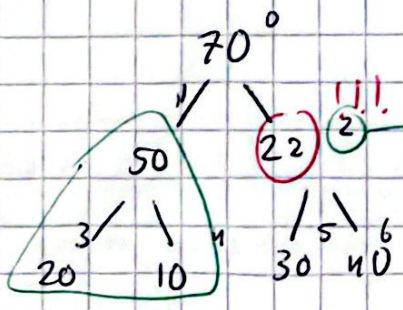
what if ... added 22?



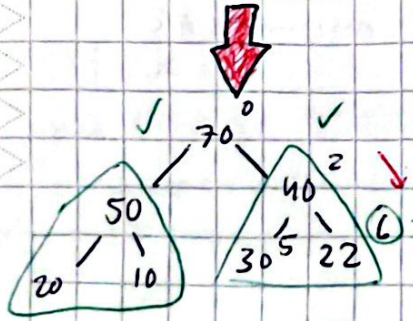
Heapify called to fix this!

HOW?

Value at i "floats downward" so that the heap rooted at i becomes a heap. Check for new violation at swapped elm.



Heapify (2)



Heapify (6)

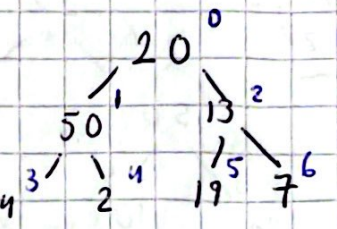
```
Heapify (i) {
    int n = indexOfMax (i, left(i), right(i));
    if (n != i) {
        swap(i, n)
        heapify(n)
    }
}
```

Private

Build heap

Convert an array A[0 .... n-1] (unordered) into a heap

Elements $A\left[\lfloor n/2 \rfloor \ldots n-1\right]$



~~Entries~~ ~~it calls heapify~~ calls heapify a shitload of times!

```
Build heap () {
    for (int i = size/2 - 1; i >= 0; i--) {
        heapify (i)
    }
}
```

↑ O (lg n)

↳ Have to code this but won't be needed for assignment #2

Heapsort {

→ assumes Heap

→ Build Heap $O(n)$

```
for( int i = size-1 ; i>0 ; i--){
    swap(0, i);
    size--;
    heapify(0);
}
```

}

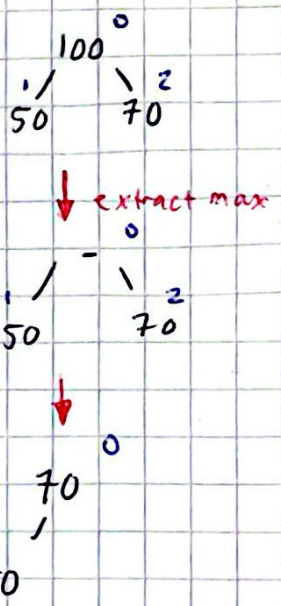
Implementation Requirements:

- Returns sorted array
- Heap is preserved
- DO minimum work
- Remember size
- Restore size
- Another array (copy values into this)
- Return sorted array must be exactly size of its elements.

ExtractMax {

```
// Check size...
int max = A[0];
A[0] = A[size-1];
size--;
heapify(0);
return max;
```

}



Size: 3

100	50	70	
0	1	2	3

public **increaseKey(i, k)** i = index { k = key value

```
// Bounds check, don't forget...
if (A[i] >= k) return;
A[i] = k;
while (i>0 && A[parent(i)] < A[i]) {
    swap(i, parent(i));
    i = parent;
}
```

}

```
public
heap Insert (int k) {
// bounds? Check size
size ++;
A[size-1] = k-1;
increaseKey (size-1, k);
}
```