Heaps – Binary

- tree-based
- types: min/max
- partially sorted
- priority queue implementation
- insertion/removal
  @ breadth-first position

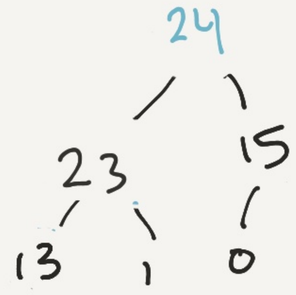
0	1	2	3	4	5	6	7
24	23	15	13	1	0	12	4

p	c
0	1, 2
1	3, 4
2	5, 6
3	7, 8

p

c1 c2

24



TreeNode Disads
– more traversal time
– more space

$c1 = 2p + 1$

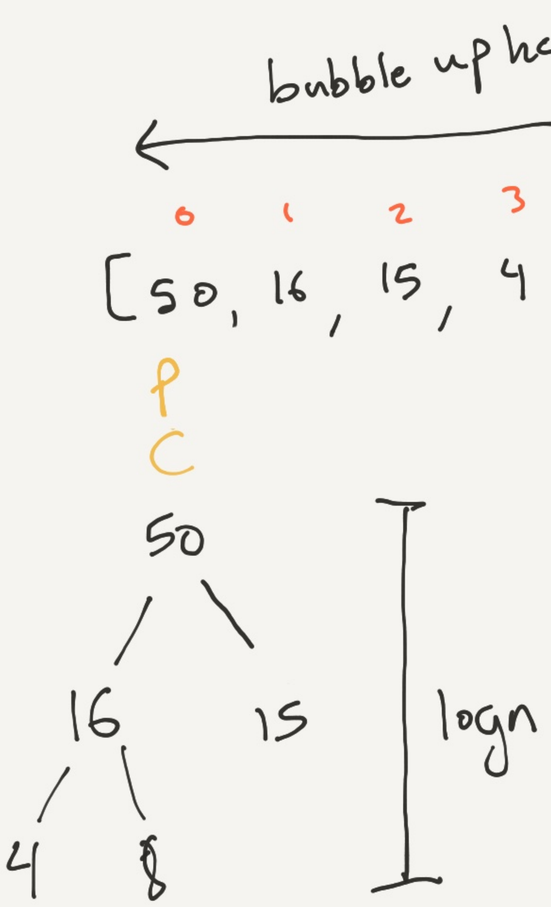
$c2 = 2p + 2$

$p = \text{floor}\left(\frac{c-1}{2}\right)$

```
TreeNode {
    int value
    TreeNode left, right, parent
```

Heapsort

1. Heapify to max heap
2. Turn max heap into sorted array



0 1 2 3 4 5 6 7

[4, 15, 16, 50, 8, 23, 42, 108]

bubbleup → check if parent < child ↗ larger elements before swap

p	c
0	1, 2
1	3, 4
2	5, 6
3	7, 8

$c_1 = 2p+1$

$c_2 = 2p+2$

$p = \text{floor}\left(\frac{c-1}{2}\right)$

Single Insertion: $O(\log n)$

Overall: $O(n \log n)$

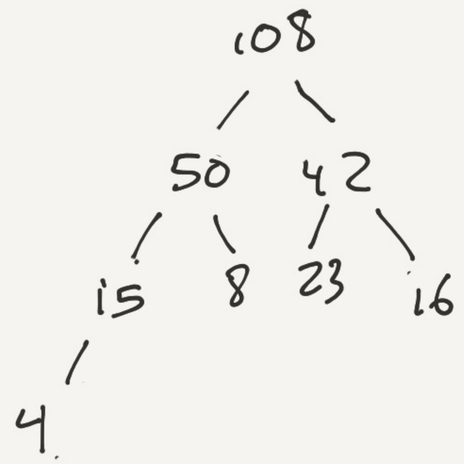
1. Bubble Down Approach Heapify

bubble down smaller elements

	0	1	2	3	4	5	6	7
	108	50	42	15	8	23	16	4

p

p c1 c2



P	C
0	1, 2
1	3, 4
2	5, 6
3	7, 8

$c_1 = 2p+1$

$c_2 = 2p+2$

$p = \text{floor}\left(\frac{c-1}{2}\right)$

Time: $O(n)$

```
for p = arr.length - 1 → 0
    bubbleDown(p)
```

2. Convert to Sorted Array

0	1	2	3	4	5	6	7
4	8	15	16	23	42	50	108

4

P	C
0	1, 2
1	3, 4
2	5, 6
3	7, 8

$c_1 = 2p+1$

$c_2 = 2p+2$

$p = \text{floor}\left(\frac{c-1}{2}\right)$

Time: $O(n \log n)$

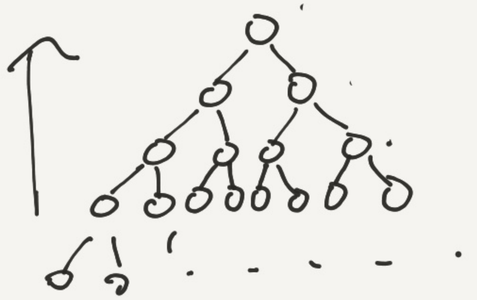
b/c

1. Swap peak with end
2. Shrink heap by 1
3. Bubble down new peak

Heapsort Overall : $O(n \log n)$

Heapify Approach Comparisons

Bubble-up



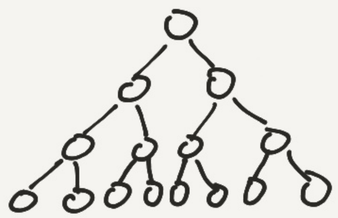
# nodes		# ops
1	×	0
2	×	1
4	×	2
8	×	3

34 operations

15 nodes

31 nodes → Bubble up 96 ops

Bubble-down



# nodes		# ops
1	×	3
2	×	2
4	×	1
8	×	0

11 operations

Bubble-down 26 ops

$$O\left(\frac{n}{2} - \log_2(n+1)\right)$$

$$\Downarrow$$

$$O(n)$$

Comparing Quasilinear Sorting Algorithms

Algorithm	Time Complexity	Space Complexity	Stability	When to Use
Quicksort	worst: $O(n^2)$ avg: $O(n \log n)$	$O(n)$ / $O(\log n)$ (depending on implementation)	Unstable	Optimizing for best time on average
Heapsort	$O(n \log n)$	$O(1)$	Unstable	Optimizing for Space
Mergesort	$O(n \log n)$	$O(n)$	Stable	Optimizing for Stability (slightly faster than heapsort)