

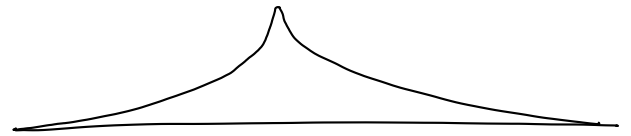
lecture 25

- $O(n)$ algorithm for building a heap
- A3 notes

Binary trees should not be drawn like this:

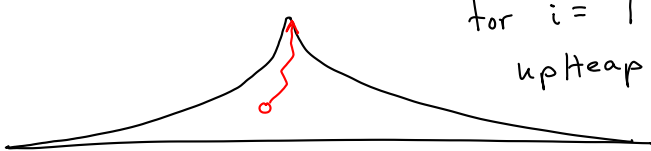


Rather they should be drawn like this:



because level l has 2^l nodes.

Last lecture: build heap

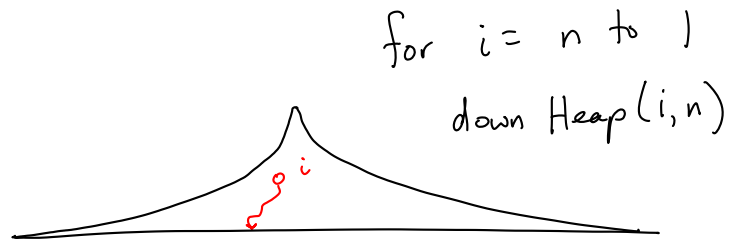


for $i = 1$ to n
upHeap(i)

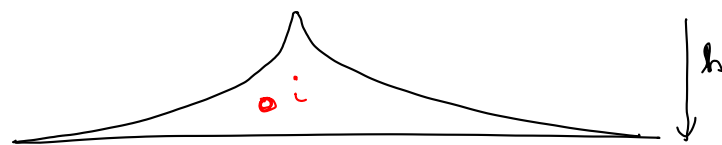
worst case:

total number of swaps $\approx n \log n$

A second way to build a heap



for $i = n$ to 1
downHeap(i, n)



Most nodes are near level h !

for $i = 1$ to n
upHeap(i)

slow

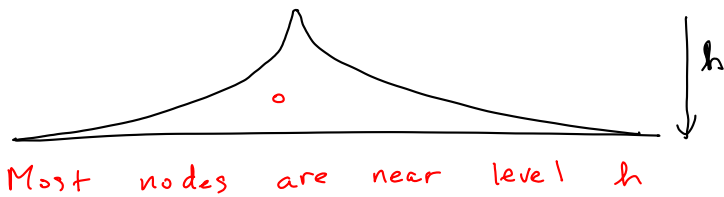
for $i = n$ to 1
downHeap(i, n)

fast

How many swaps (worst case)?



- there are 2^l nodes at level l
- downHeap a node at level l requires at most $h-l$ swaps.

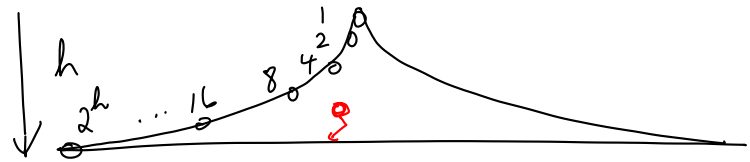


1	node	at level 0	get swapped $\leq h$ times
2	nodes	" 1	" $\leq h-1$
4	"	" 2	" $\leq h-2$
			⋮
2^{h-1}	"	" $h-1$	" ≤ 1
2^h	"	" h	" ≤ 0

2^l nodes at level l get swapped $\leq h-l$ times

\therefore total # Swaps $\leq \sum_{l=0}^h (h-l) 2^l$

Claim: total # Swaps $< 2^{h+1} < 2n$



Proof (see lecture notes)

Uses the following trick

$$\sum_{l=0}^h l 2^{-(l-1)} = \sum_{l=0}^h l x^{l-1}, \quad x = \frac{1}{2}$$

$$= \frac{d}{dx} \sum_{l=0}^h x^l$$

Summary: Heaps

- add(), remove Min()
- upHeap(), downHeap()
- building a heap
 - using upHeap $\sim n \log n$
 - using downHeap $\sim n$
- Also, heap Sort (last class)

Assignment 3

- Casting primitive types (char and int)
- prefix codes

Recall from lecture 3

char c;

// In Java, a char represented using 2 bytes (16 bits)

int i;

// An int is represented using 4 bytes (32 bits)

