Heap Sort

Reminder on Selection Sort

```
for i := 1 to n - 1
    find mini := position of smallest element
                 in A[i..n]                    (1)
    swap A[i] and A[mini]                      (2)
```

Running time

(1) $\Theta(n)$
(2) $\Theta(1)$

} n times $\Longrightarrow$ $\Theta(n^2)$ in total

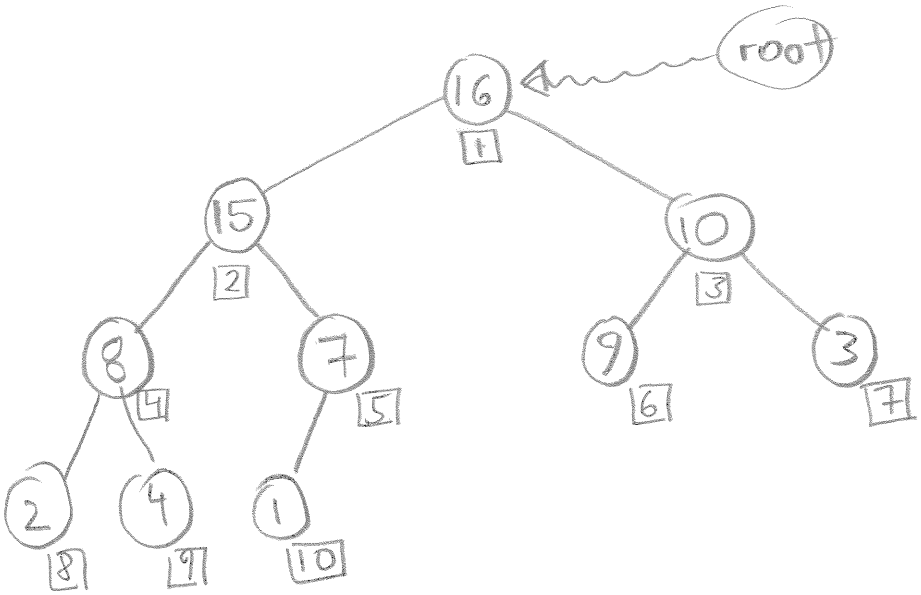
Can we improve this?

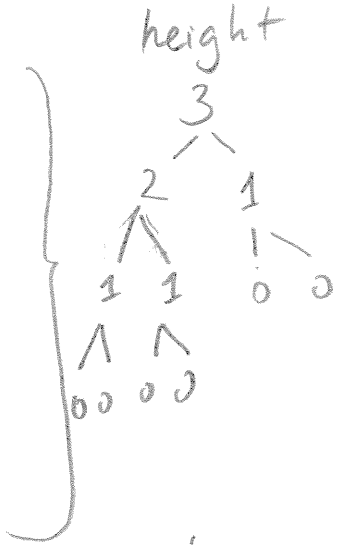
Idea: Find data structure s.th.

- (1), (2) take $\Theta(1)$
- maintain data structure in $O(\log n)$

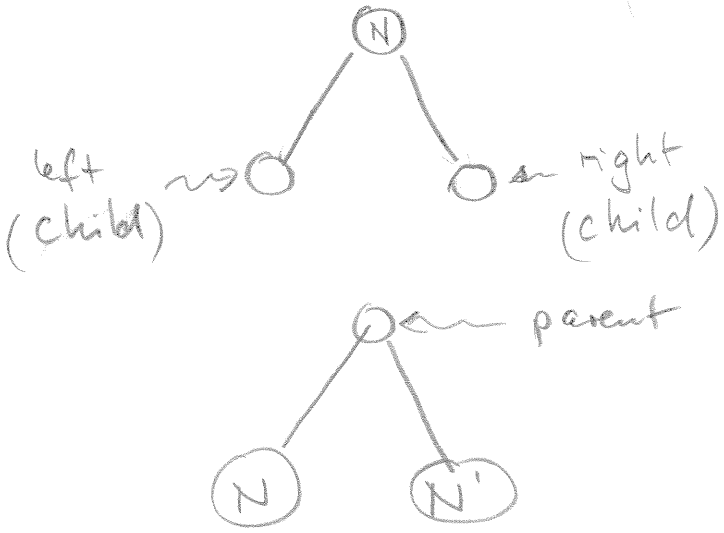
$\Rightarrow$ total time $O(n \cdot \log n)$

Heap Data Structure





Nearly complete binary tree s.th. value(N.parent) ≥ value(N)



Implement binary tree as array

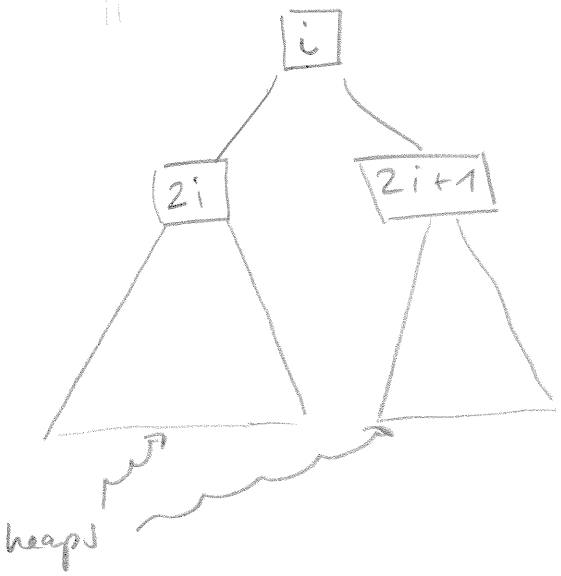
A	16	15	10	8	7	9	3	2	4	1
levels	0	1		2				3		
positions	1	2	3	4	...		7	8	...	

```
parent(i)
    return ⌊i/2⌋

left(i)
    return 2*i

right(i)
    return 2*i + 1
```

Maintain the heap property



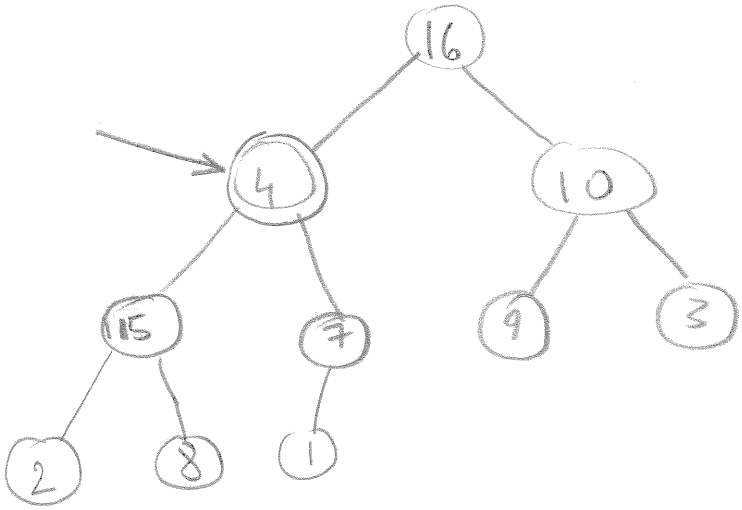
What if

$A[i] < A[left(i)]$

or $A[i] < A[right(i)]$?

Call Heapify (A, i)

Example



```
Heapify (A, i, n)   % n is heapsize
    l := left(i) ;   r := right(i)
    if l ≤ n  and  A[l] > A[i]
       then imax := l
       else imax := i ;
    if r ≤ n  and  A[r] > A[imax]
       then imax := r
    if imax ≠ i
       then swap A[i] ↔ A[imax]
            Heapify (A, imax, n)
```

Running Time

- height (i) = h ⇒ Running Time = $O(h)$ = $O(\log n)$

Correctness of Heapify

Induction on depth of tree rooted at i.

depth $= 0 \Rightarrow l > n$ (and $r > n$)

$\Rightarrow i_{max} = i$

$\Rightarrow$ Heapify does nothing

depth $= d + 1$

subtree(i) not heap

$\Rightarrow A[i] < A[l]$ or $A[i] < A[r]$

W.log $A[r] = \max\{A[i], A[l], A[r]\}$

and $A[r] > A[i]$, $A[r] > A[l]$

$\Rightarrow i_{max} = r$

After return of Heapify(A, max, n),

- subtree(r) is a heap (ind. hypothesis)
- subtree(l) is a heap (assumption)
- $A[i] \geq$ all elements in subtree(l), subtree(r)

Building a Heap

Convert $A[1..n]$ into a heap

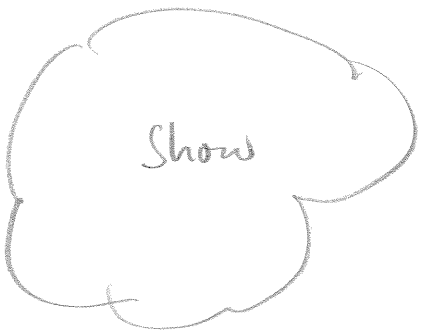
Note: subarray $A[\lfloor n/2 \rfloor + 1 .. n]$ is already heap

$\Rightarrow$ work downwards from $\lfloor n/2 \rfloor$

```
Build Heap (A)
    n := A.length
    for i := ⌊n/2⌋ downto 1
        do Heapify (A, i, n)
```

Example

$A =$ | 4 | 1 | 3 | 2 | 16 | 9 | 10 | 14 | 8 | 7 |



BuildHeap: Correctness

- Induction on i:

Loop invariant

when Heapify(A, i, n) is called

$\Rightarrow$ subtree(j) is a heap, f.a. $j > i$

Running Time

- Rough estimate

  $n/2$ calls to Heapify

  $O(\log n)$ r.t. per call

  $\Rightarrow$ $O(n \log n)$

- Better estimate

  - n element heap
    - maximal element height $= \lfloor \lg n \rfloor = h_{max}$
    - # nodes of height $h \leq \lceil \frac{n}{2^{h+1}} \rceil$

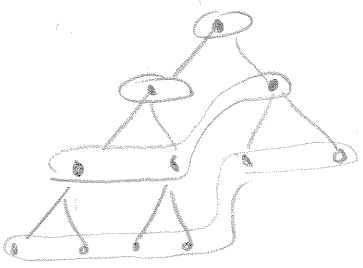
$n = 11$

$h = 3$

$h = 2$

$h = 1$

$h = 0$



$\leq$ Half the nodes are leaves

$\leq$ 1/4 the nodes are at level 1

...

- Cost per call of Heapify

  $O(h)$

$$T(n) = \sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h)$$

$$= O\left(\sum_{h=0}^{\lfloor \log n \rfloor} \frac{n}{2^{h+1}} h \right)$$

$$= O\left(n \sum_{h=0}^{\lfloor \log n \rfloor} h \left(\frac{1}{2}\right)^h \right)$$

We remember

- $\sum_{k=0}^{n} x^k = \frac{1 - x^{n+1}}{1 - x}$

$$\Rightarrow \sum_{k=0}^{\infty} x^k = \frac{1}{1-x}, \text{ if } |x| < 1$$

- $\sum_{k=0}^{\infty} k x^k = x \sum_{k=0}^{\infty} k x^{k-1} = x \sum_{k=0}^{\infty} \frac{d}{dx} x^k$

$$= x \frac{d}{dx} \sum_{k=0}^{\infty} x^k = x \frac{d}{dx} \frac{1}{1-x} = \frac{x}{(1-x)^2}, \text{ if } |x| < 1$$

Apply for $x = 1/2$

$$T(n) = O\left(n \frac{1/2}{(1-1/2)^2} \right) = O(n)$$

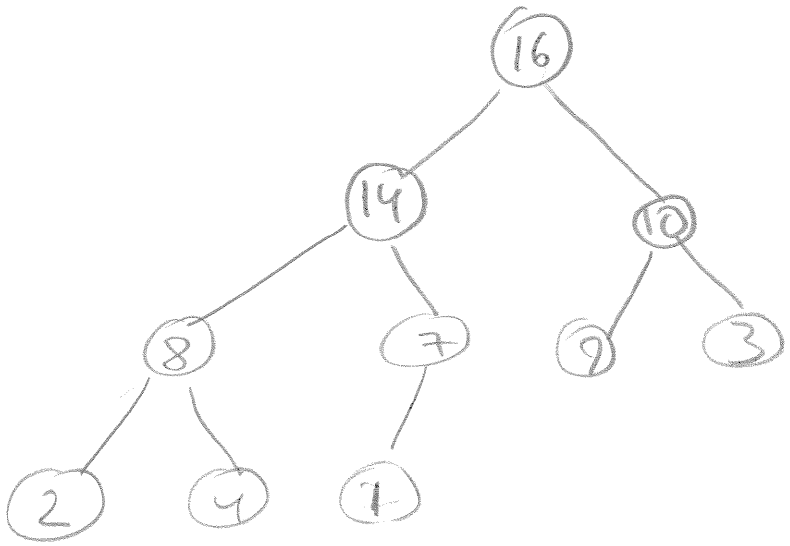
Intuition: In a 2ary tree, almost all nodes are at the bottom

Heap Sort

	Running Time
HeapSort (A)	
BuildHeap (A)	$O(n)$
heapsize := A.length	
for i := A.length downto 2	n times
do swap A[1] $\leftrightarrow$ A[i]	$O(1)$
heapsize := heapsize − 1	$O(1)$
Heapify (A, 1, heapsize)	$O(\log n)$

$\Rightarrow T(n) = O(n \log n)$

Show for heap



Correctness of Heapsort

Loop invariant

- $A[1..heapsize]$ is a heap containing the hs least elements of A
- $A[hs+1..A.length]$ is sorted containing the $A.length - hs$ greatest elements of A