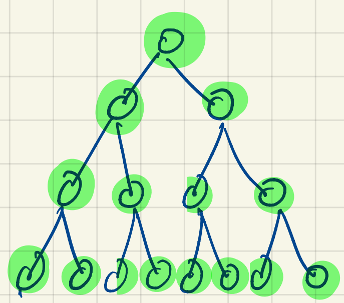
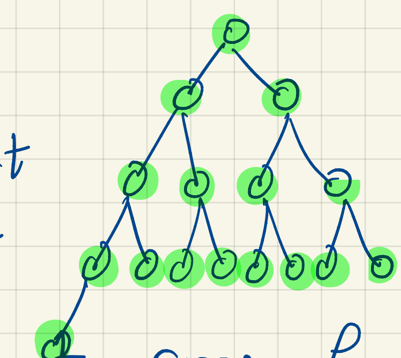
Heap sort

- Heapsort is an $O(n \log n)$ sorting algorithm
- It uses a heap as its underlying data structure.
- Heap is Nearly complete binary tree
  - All levels are complete except possibly last one
  - Heap propetry [later]
  - Height of tree is $\Theta(\log n)$ where n is the number of nodes.



h = # edges longest path from root to leaf



Best case: $h = \log_2(n+1) - 1$

$= \log_2\left(\frac{n+1}{2}\right) \geqslant \frac{1}{2}\log_2 n$

Worst case: $h = \log_2 n$

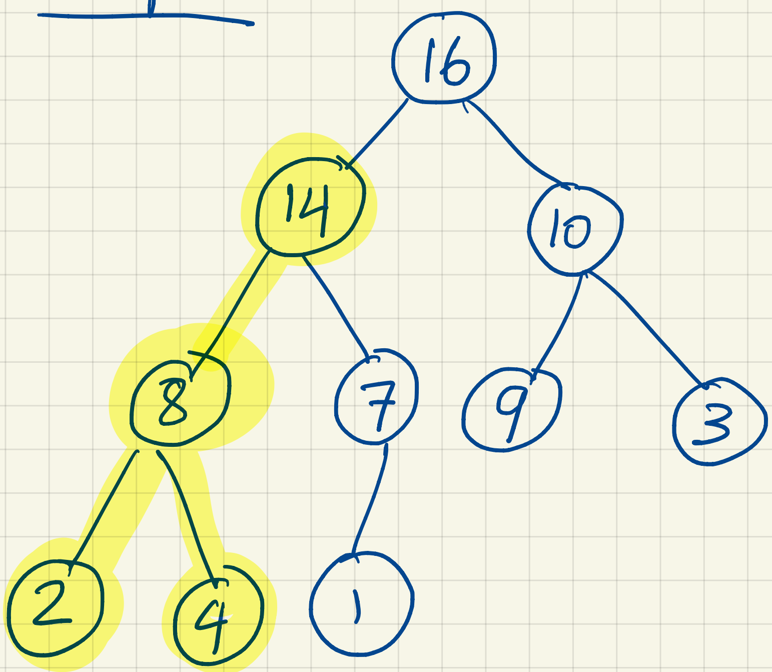
$\frac{1}{2}\log_2 n \leqslant h \leqslant \log_2 n$

Heap Property (Max heap)

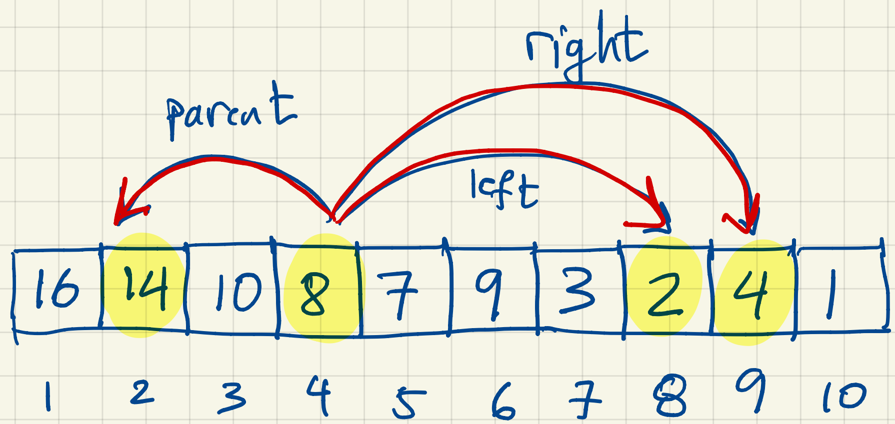
The heap is stored as an array

- $A[1]$ is the root
- Parent of $A[i] = A[\lfloor i/2 \rfloor]$ — $\text{Parent}(i) = \lfloor \frac{i}{2} \rfloor$
- Left child of $A[i] = A[2i]$ — $\text{left}(i) = 2i$
- Right child of $A[i] = A[2i+1]$ — $\text{right}(i) = 2i+1$
- Heap property : $A[\text{parent}(i)] \geq A[i]$

Example:

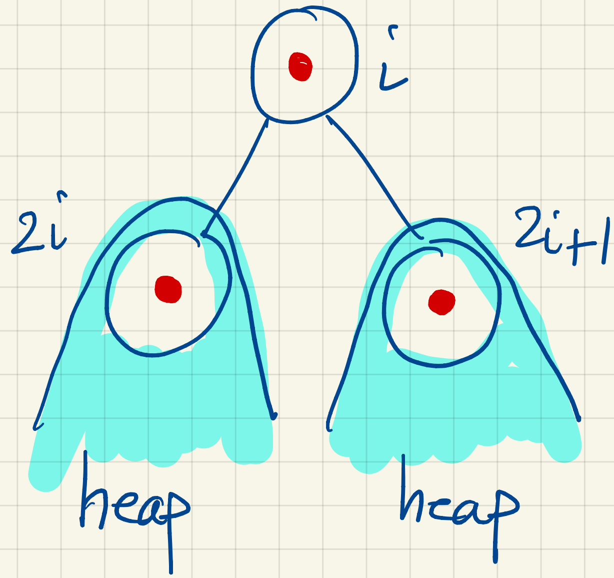


Because it's nearly complete



Max heap property guarantees that max. element is at the root.

Maintaining heap property



Heapify(A, i, n)

  $\ell \leftarrow$ index of largest among $\{A[i], A[2i], A[2i+1]\}$

  if $\ell \neq i$

  then swap $A[i] \Leftrightarrow A[\ell]$

    Heapify(A, ℓ, n) (fix)

Going down a path

Build-heap (A, n)

  for $i \leftarrow \lfloor \frac{n}{2} \rfloor$ down to 1

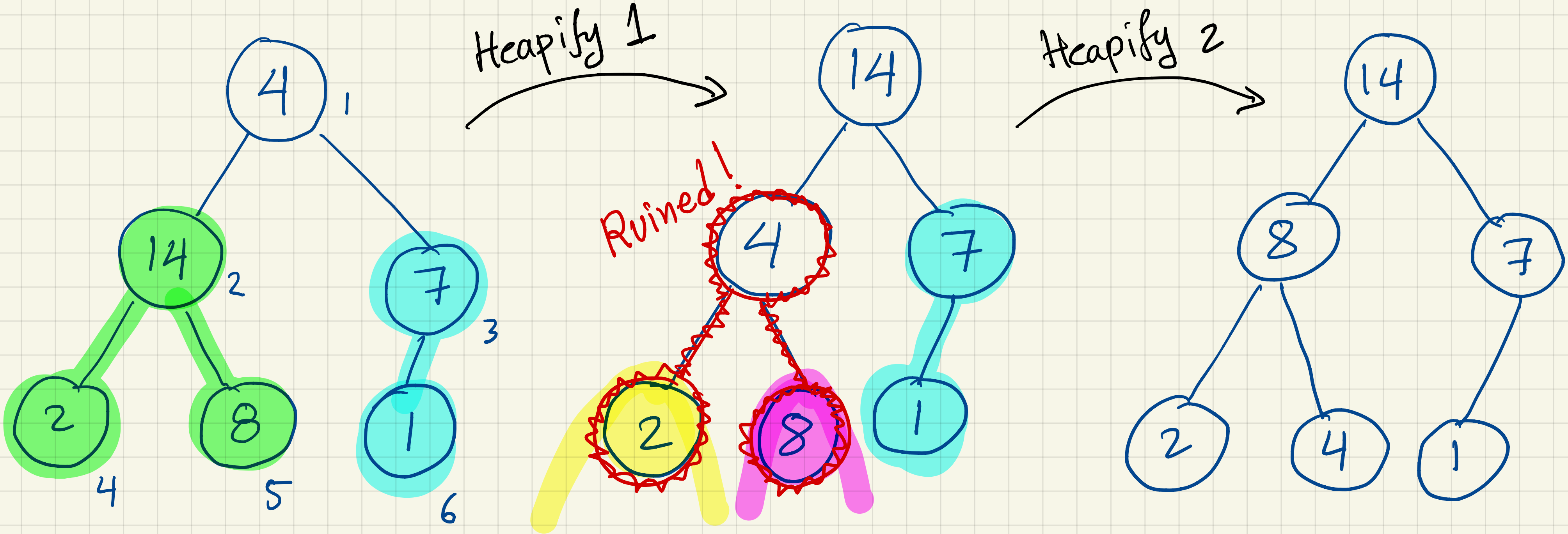
    do Heapify (A, i, n)

(leaves are heaps)

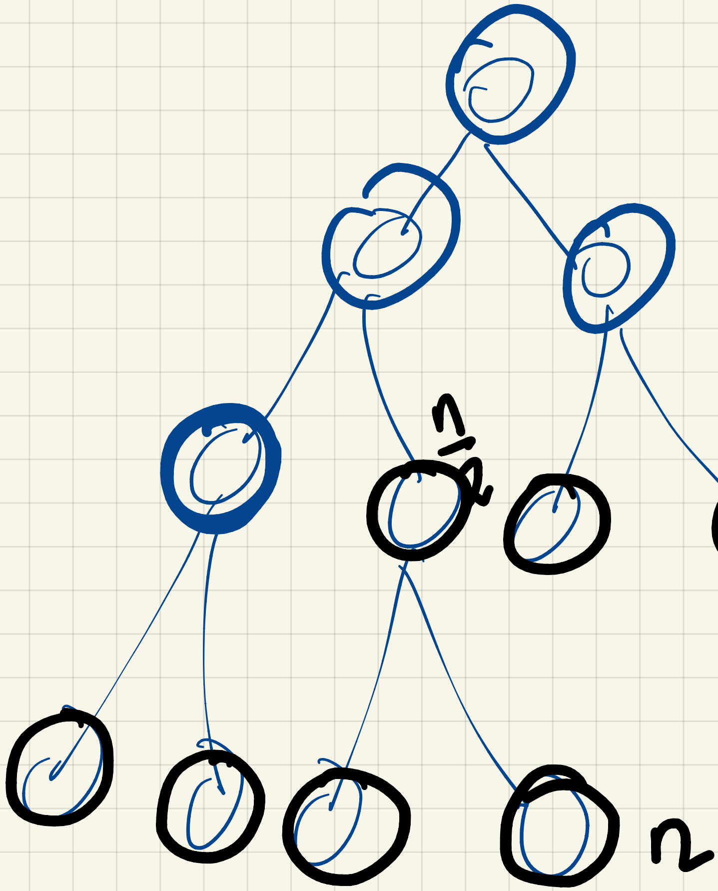
At beginning of iteration i, each node $i+1, i+2, \dots, n$ is root of heap.

loop invariant

Example (illustrating Heapify)



Going down the path $\Longrightarrow$ $O(\log n)$


n/2
n

What is the running time of Buid heap?

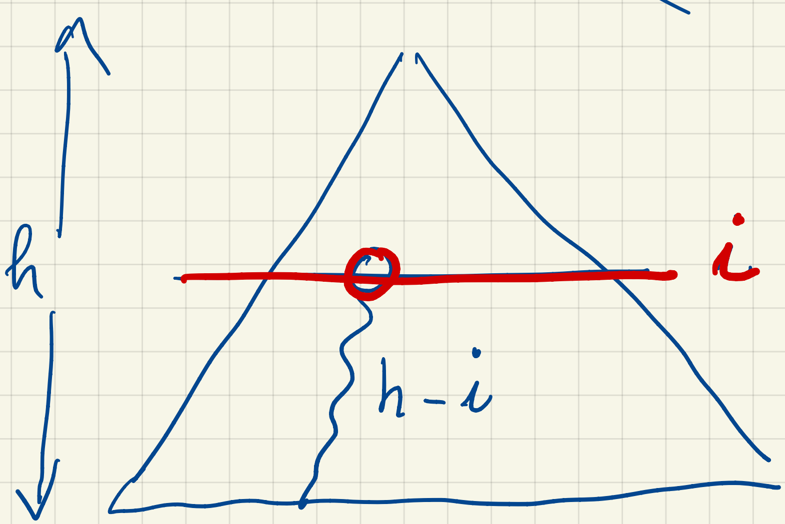
$\Theta(n)$ calls to Heapify, Heapify is $O(\log n)$

$\Rightarrow O(n \log n)$ [not tight]

We can say better!

There are at most 2^i nodes a level i

each has height $(h-i)$



$$\sum_{i=0}^{h} 2^i (h-i) = h + 2(h-1) + 4(h-2) + \dots + 2^h \cdot 0$$

$$= 2^h \left[\frac{h}{2^h} + \frac{h-1}{2^{h-1}} + \dots + 0 \right]$$

$$\leq 2^h \sum_{k=0}^{\infty} \frac{k}{2^k} = \Theta(n) \underbrace{\sum_{k=0}^{\infty} k(0.5)^k}_{\text{constant}}$$

So BuildHeap runs in $\Theta(n)$ time.

Another $O(n \log n)$ sorting algorithm.

Heapsort (A, n)

Build Heap (A, n) - - - - - - - - $\Theta(n)$

for $i \leftarrow n$ downto 2

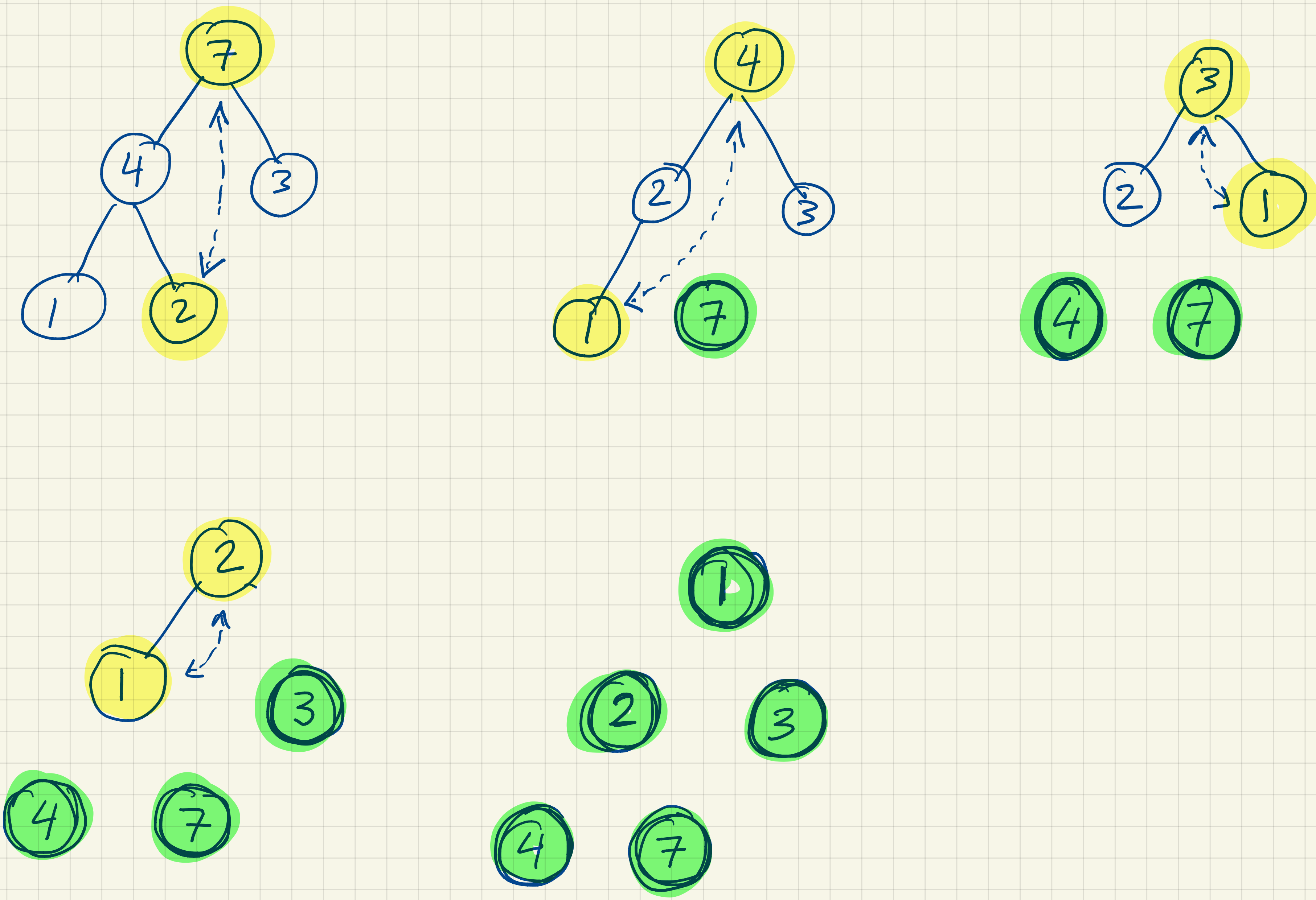
do swap $A[1] \Leftrightarrow A[i]$

Heapify $(A, 1, i-1)$ . . . . $O(\log n)$

Sorts in $O(n \log n)$

Why isn't this $O(n)$ as well ?

Example of Heapsort:



We always heapify at root $\Rightarrow$ sum of distances to root (Not sum of heights)

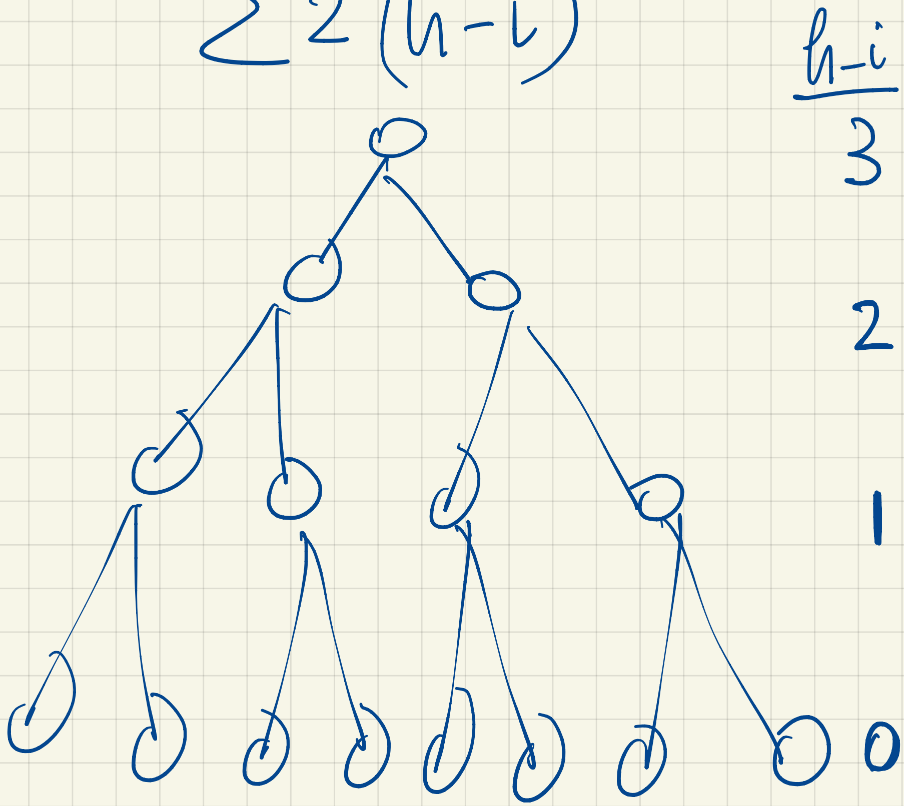
$$\sum_{i=0}^{h} 2^i i = \quad . \; . \; . \quad \Theta(n \log n)$$

(try it)

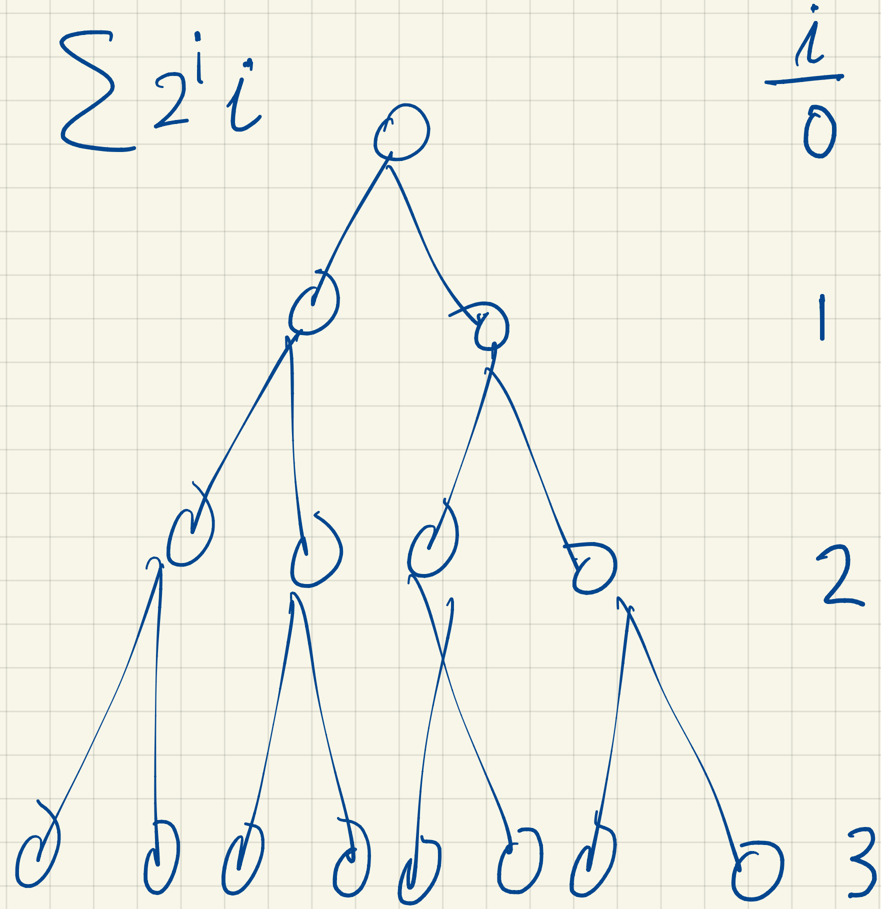
compare with $\sum_{i=0}^{h} 2^i (h-i)$ from before

$\sum 2^i (h-i)$

$\sum 2^i i$



n



$n \log n$

Heap as Priority queue

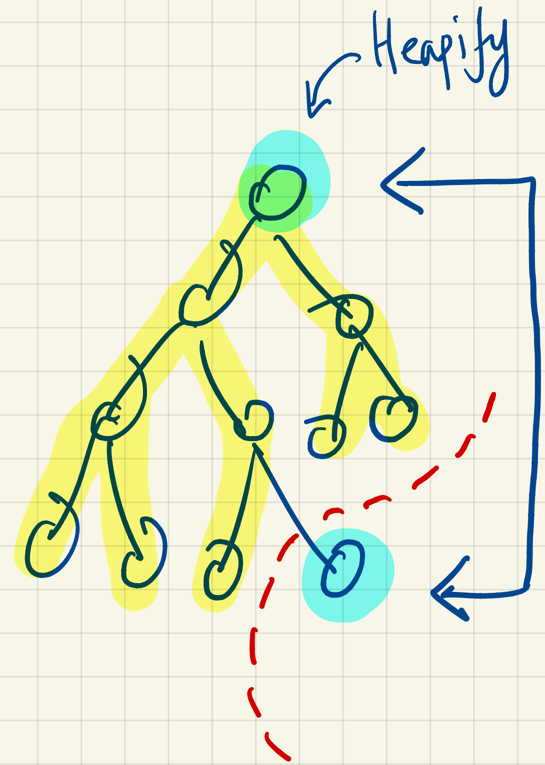
Maintain a dynamic set S of keys supporting the following operations:

- Insert (S, x) : inserts x into S
- Maximum (S) : returns element with largest key
- Extract-max (S): removes & returns elem with " "
- Increase-key (S, x, k): increases x's key to k (Assume $k \geq x$'s current key)

Maximum(A)
    return A[1]

Time $\Theta(1)$

Extract-max(A, n)
    if $n < 1$
        then return "error"
    $m \leftarrow A[1]$
    swap $A[1] \leftrightarrow A[n]$
    Heapify(A, 1, n-1)
    return m



Time : $O(\log n)$

Not showing explicit update of Heap size

Increase-key (A, i, key) ← new value for the key

  if key < A[i]

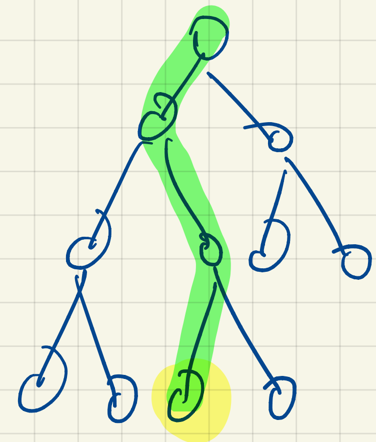
    then return "error"

  A[i] ← key

  while i>1 and A[parent(i)] < A[i]

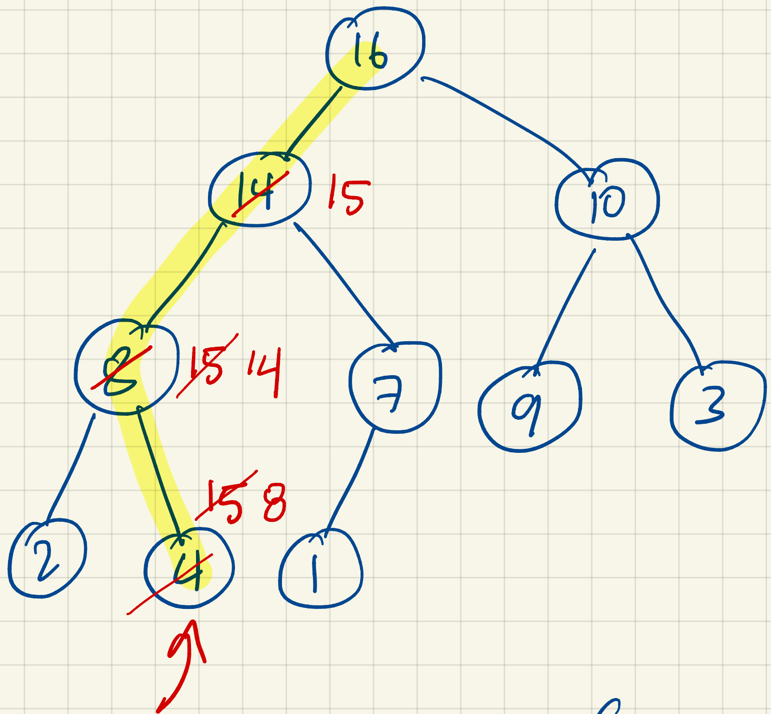
    do swap A[i] ⟺ A[parent(i)]

      i ← Parent(i)



Time: $O(\log n)$

Example of increase key.

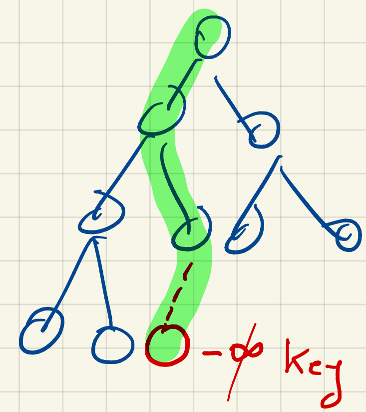


(Only consider nodes on path to root)

Insert (A, key, n)

$A[n+1] \leftarrow -\infty$

Increase-Key $(A, n+1, key)$



Time : $O(\log n)$

Build-heap (A, n)

for $i \leftarrow \lfloor \frac{n}{2} \rfloor$ down to 1

do Heapify (A, i, n)

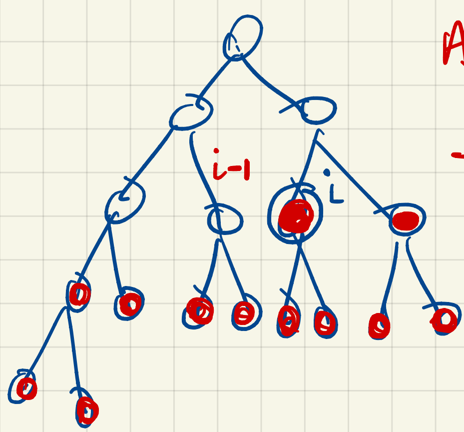
(leaves are heaps)

At beginning of iteration i, each node $i+1, i+2, \ldots, n$ is root of heap.

loop invariant

At beginning of iteration i

nodes $i+1, i+2, \ldots n$ are roots of heaps.

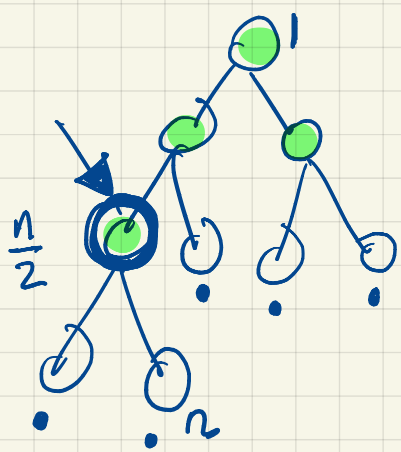


At beginning of iter. $i-1$

We went through iteration i and finished we called Heapify(A, i, n)

nodes $i, i+1, \ldots, n$ are heaps

$(i-1)+1, (i-1)+2, \ldots n$ are heaps



Init: $i = \lfloor \frac{n}{2} \rfloor$

nodes $\lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, \ldots n$ are root of heaps

Maintenance:

loop invariant is true at beginning of iteration i, show it's true at beginning of iteration $i-1$

At termination $i=0$, $\Rightarrow$ all nodes $1, 2, 3, \ldots, n$ are heaps.