

NOTE BOOK

Rokshana Ahmed

PRIORITY QUEUES / HEAPS

A heap makes it possible for you to store N (key, value) pairs, where N is to be fixed a priori, in such a way that

- The item with the lowest key can be accessed in $O(1)$ time;
- The item in position "i" can be removed from the heap in $O(\log n)$ time;
- A new item can be inserted into the heap in $O(\log n)$ time.



ARRAY

- Removing and adding takes $O(1)$ time

- Finding the minimum takes $\Omega(n)$ time

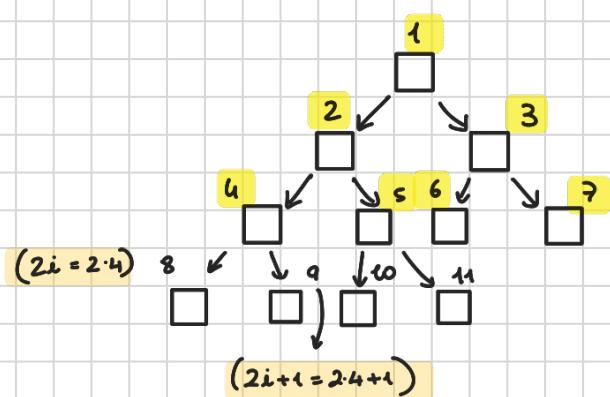


SORTED ARRAY

- Finding the minimum takes $O(1)$ time.

- Remove and adding takes $\Omega(n)$ time.

Heaps "interpolate" between the two extremes

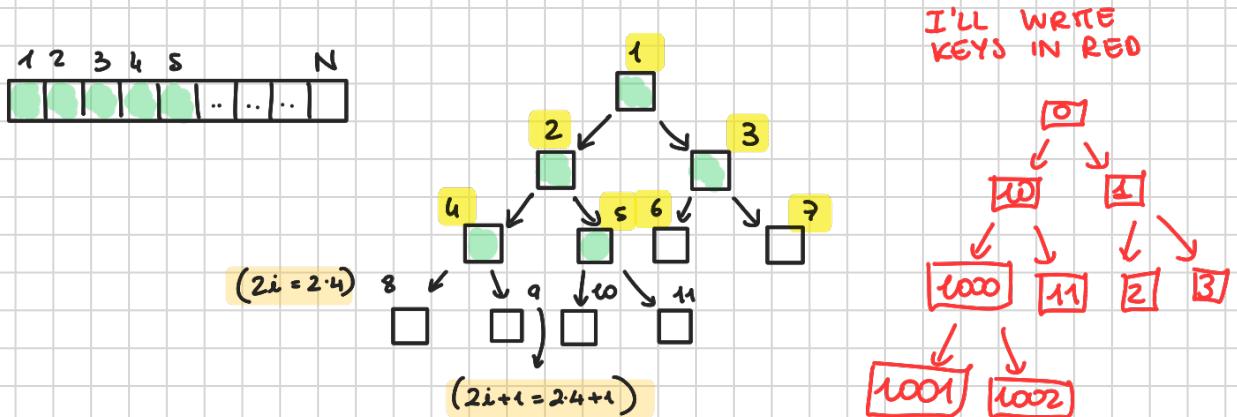


This "logical" representation of the array has a number of useful properties

- The left child of node "i" has index/position $2i \triangleq \text{left}(i)$
- The right child of node i has index " $2i+1 \triangleq \text{right}(i)$ "
- The parent of node i has index $\lfloor \frac{i}{2} \rfloor \triangleq \text{parent}(i)$

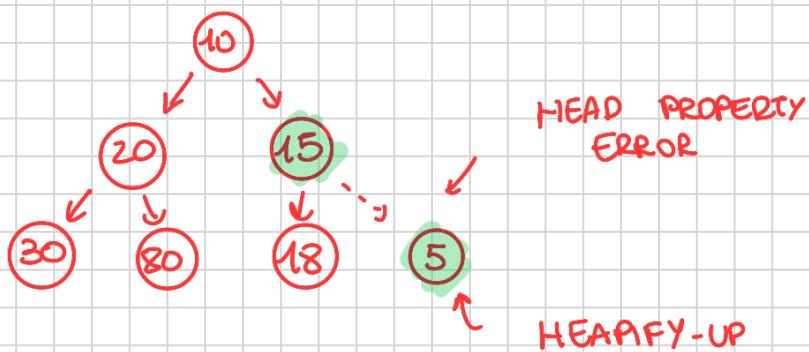
We will be representing this binary tree with an array.

If at some point, the heap only contains $n \leq N$ items then only the first "n" positions of the array will be filled



HEAP PROPERTY :

A heap satisfies the HEAP PROPERTY if, for each filled position i in the heap, if "v" is the item in position i , and if w is the item in position $\text{parent}(i)$ ($= \lfloor \frac{i}{2} \rfloor$), then
 $\text{KEY}[w] \leq \text{KEY}[v]$



HEAPIFY-UP(H, i)

// This function will try to "push up" the error
// in position i

IF $i \geq 2$:

// i has a parent

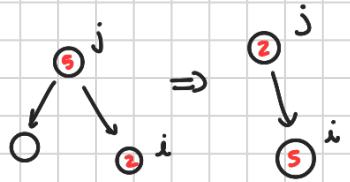
$$j = \lfloor \frac{i}{2} \rfloor$$

IF $\text{KEY}[H[i]] < \text{KEY}[H[j]]$:

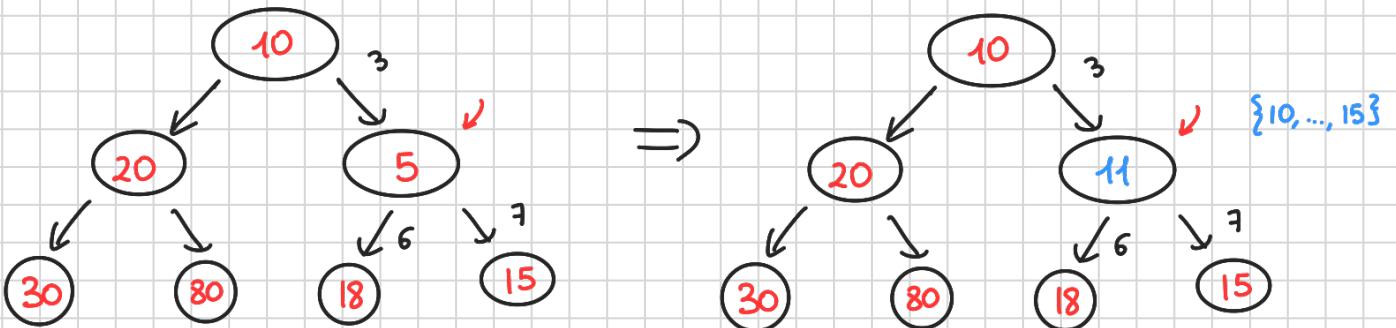
// the heap property fails at i

SWAP $H[i]$ with $H[j]$

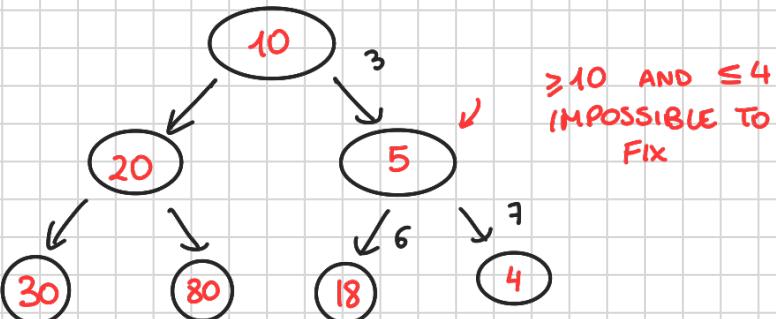
HEAPIFY-UP(H, j)



DEF: We say that H is an almost - head - with - $H[i]$ - too - small if it is possible to increase the key of $H[i]$ to a value $\alpha \geq \text{KEY}[H[i]]$, so that the resulting structure has the heap property.



This H has the almost - head - with - $H[3]$ - too - small prop.



THIS HEAP DOES
NOT HAVE THE
ALMOST - HEAP - WITH -
 $H[3]$ - TOO - SMALL
PROPERTY

L: The function $\text{HEAPIFY-UP}(H, i)$ fixes the heap property at H (provided it needs fixing), if H has the ALMOST-HEAP-WITH- $H[i]$ -TOO-SMALL PROPERTY.

The runtime of $\text{HEAPIFY-UP}(H, i)$ is $O(\log n)$.

P: We prove the claim by induction on i .

If $i=1$, the claim is true since H already has the HEAP-PROPERTY (i has no parent).

If $i \geq 2$, then the algorithm swaps $H[i]$ with $H[j]$ where $j = \text{parent}(i) = \lfloor \frac{i}{2} \rfloor$ if $\text{KEY}[H[i]] < \text{KEY}[H[j]]$.

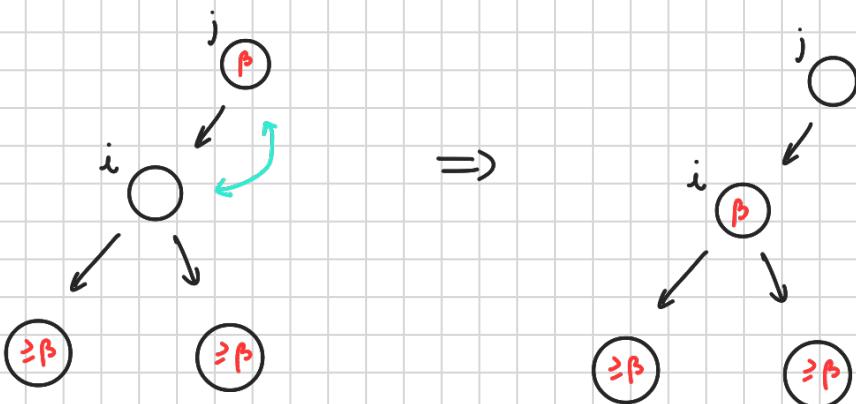
Let β be equal to $\beta = \text{KEY}[H[j]]$, before the swap

Because of the A-H-W-H[i]-T-S property $\exists \alpha \geq \text{KEY}[H[i]]$ such that if the key of $H[i]$ becomes α , the heap is fixed.

But, then, such a α has to satisfy $\alpha \geq \beta$, i.e., α cannot be smaller than the key of the parent of i , and, moreover, α has to satisfy

$$\alpha \leq \text{KEY}[H[\text{left}(i)]] \text{ AND } \alpha \leq \text{KEY}[H[\text{right}(i)]]$$

Thus, $\beta \leq \alpha \leq \min(\text{KEY}[H[\text{left}(i)]], \text{KEY}[H[\text{right}(i)]])$



Thus, after the swap, $\text{left}(i)$ will have a key not smaller than that of i . Same is true for $\text{right}(i)$. Since the algorithm made the swap, β was larger than the key of $H[i]$. After the swap, then, the key of i will be larger than the key of $j = \text{parent}(i)$.

Then, after the swap, the heap is going to have the A-H-W-H[j]-T-S PROPERTY.

Since $j = \lfloor \frac{i}{2} \rfloor$, we have $j < i$. And the claim for $j < i$ is true by induction.

As for the runtime, each call to HEAPIFY-UP takes $O(1)$ time plus, possibly, an extra call to HEAPIFY-UP.

Since the number of calls is bounded by $O(\log n)$, the height of the tree, the runtime is $O(\log n)$ ■

Using HEAPIFY-UP, we can easily add/insert items to the heap.

ADD(h, v, n)

// n is the number of items currently in the heap

IF $n \leq N - 1$: // then the HEAP has enough space for v

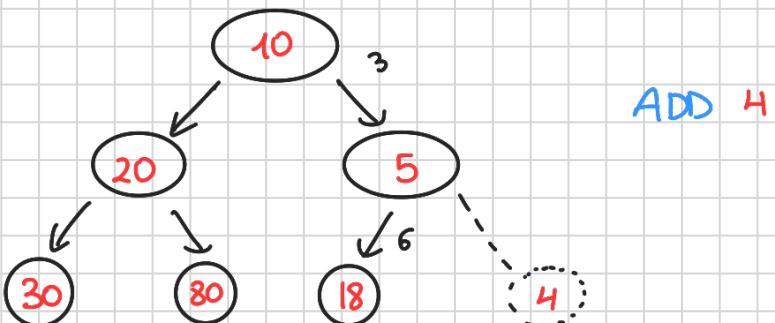
$n += 1$

$H[n] = v$

// now the HEAP might not satisfy the HEAP PROPERTY

// But the HEAP satisfies the ALMOST-HEAP-WITH-TOO-SMALL Prop.

HEAPIFY-UP(H, n)



THM : ADD adds an item to the HEAP, while keeping the HEAP Property, in time $O(\log n)$

DEF: We say that H is an almost-head - with $H[i]$ - too-large if $\exists \alpha \leq \text{KEY}[H[i]]$ s.t. if we decrease $\text{KEY}[H[i]]$ to α , then H satisfies the HEAP Property.

HEAPIFY-DOWN(H, i):

i is a position in the heap, $1 \leq i \leq n$

LET n BE THE CURRENT SIZE OF THE HEAP

IF $2i > n$:

i has no left, and no right, child

RETURN

ELIF $2i == n$:

i has only the left child

$j = 2i$

ELSE:

i has both children

IF $\text{KEY}[H[\text{right}(i)]] < \text{KEY}[H[\text{left}(i)]]$:

$j = \text{right}(i)$

ELSE:

$j = \text{left}(i)$

IF $\text{KEY}[H[j]] < \text{KEY}[H[i]]$:

SWAP $H[i]$ AND $H[j]$

HEAPIFY-DOWN(H, j)