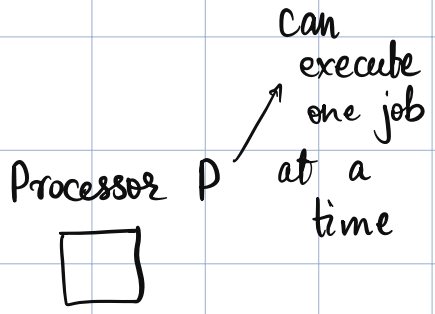
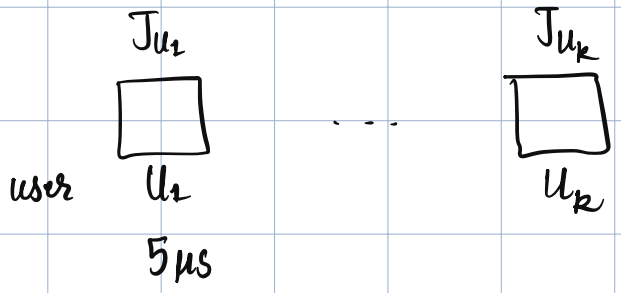
2024|10|15 - Data Structures - Week 11

Priority Queue

Motivation: Scheduling Problem

→ Minimize average waiting time





① Need to maintain remaining processing time of all unfinished jobs at any point of time.

② Need to find job with the shortest remaining time.

③ When a job finishes, need to remove it from the list

④ — ·· — new job arrives, need to add it to the list.

Priority Queue: maintains a set of elements with some associated priority.

Supports the following operations

* Insert (x): insert an element x in the set S. $S \leftarrow S \cup \{x\}$
* Minimum ( ): return element with the smallest priority.
* Delete_min ( ): return and remove elements in S with smallest priority

<u>Naive implementation</u>

① Maintain a sorted array

$O(\log n)$ $O(n)$
find correct location + shifting

Insert (x) — $O(n)$

Minimum() — $O(1)$

Delete_min() — $O(1)$

② Maintain unsorted array

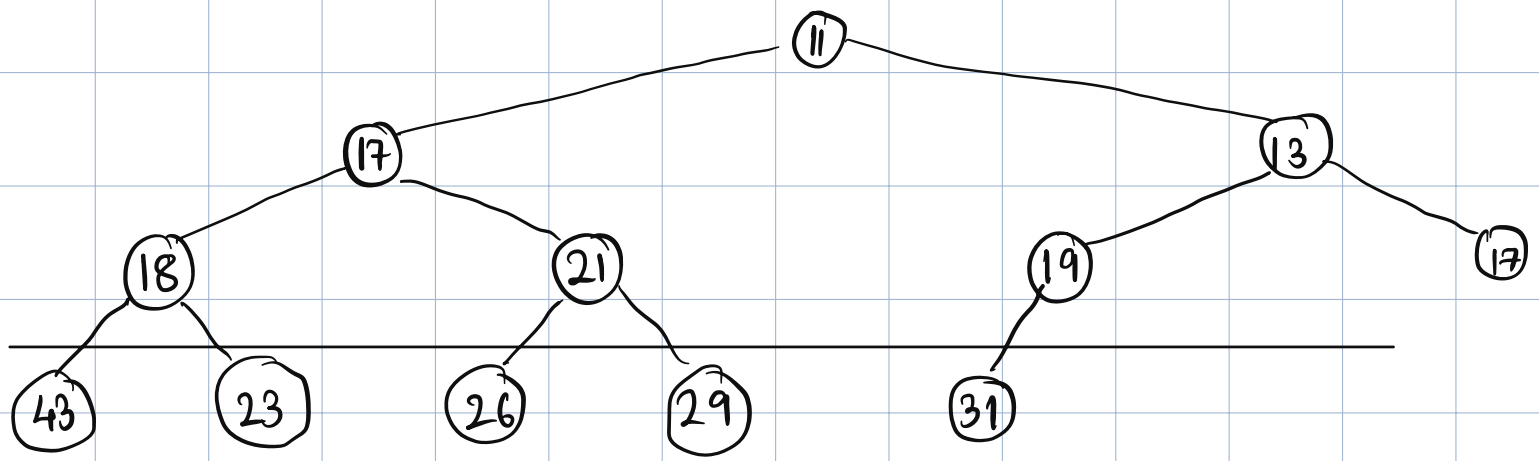
Insert (x) — $O(1)$

Minimum () — $O(n)$

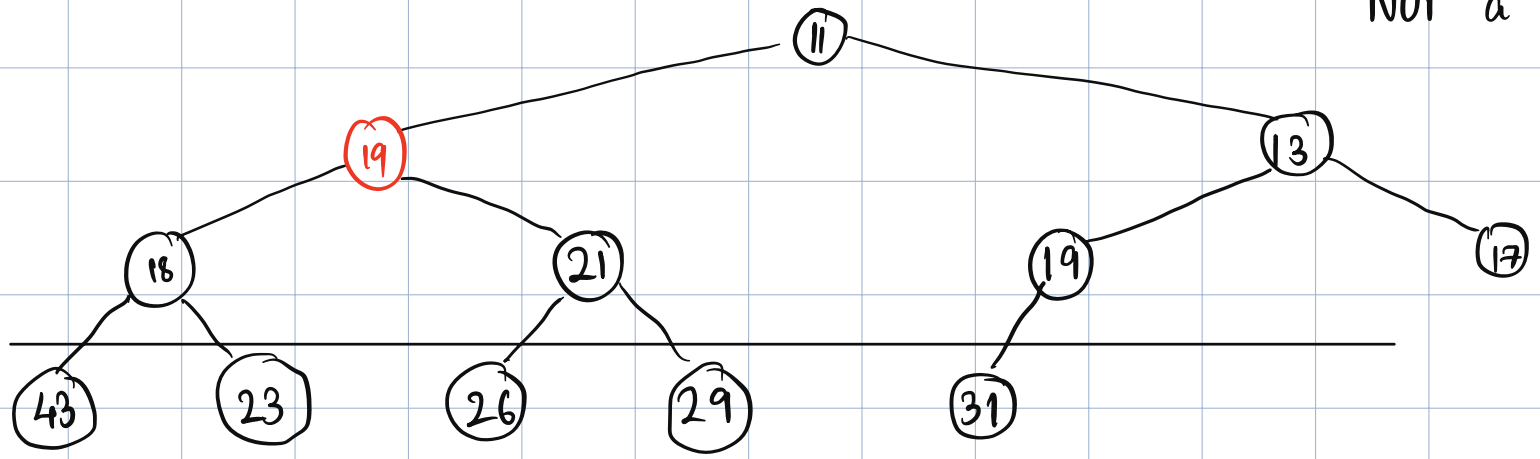
Delete_min() — $O(n)$

Clever approach: Heap data structure (array object that we can view as nearly complete binary tree)

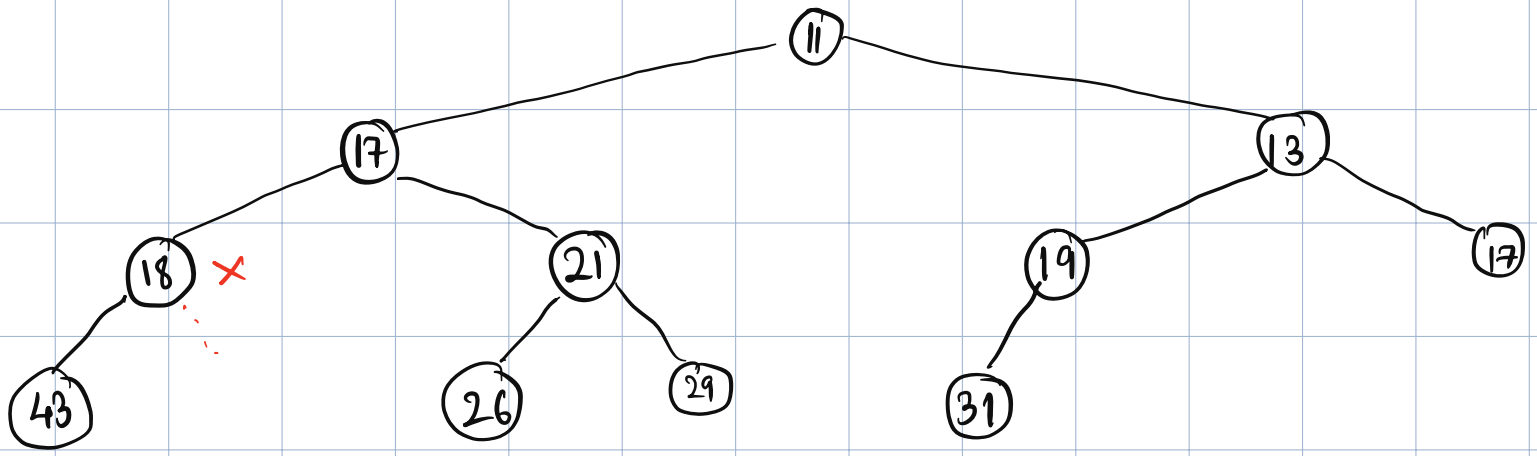
- A binary tree that stores priority at nodes.
- All levels except the last are full.
- Last level is left filled.
- Priority of a node should be smaller than its children



NOT a heap tree



NOT a heap tree



Finding the minimum element

① The element with smallest priority always sits at the root of the heap

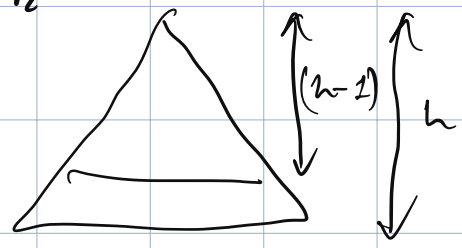
Prove by contradiction 

Compute the minimum in $O(1)$ time.

Height of heap

Suppose a heap over n nodes has height h

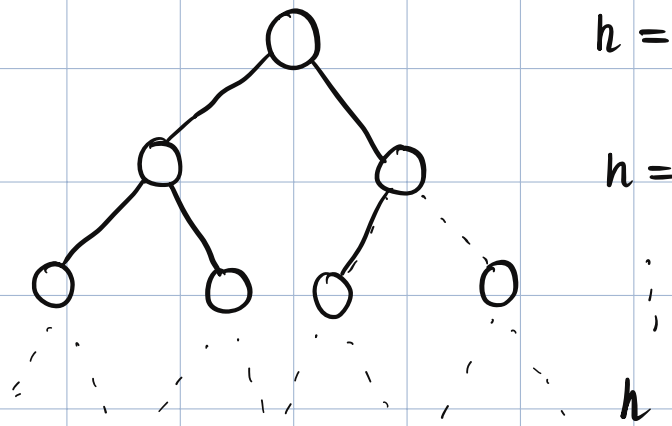
$$\underbrace{(2^h - 1)}_{\#\text{ of nodes till level } (h-1)} + 1 \leq n \leq 2^{h+1} - 1$$



height of a node in a heap = no. of edges on the longest simple downward path from node to the leaf

height of the heap := height of the root

6.1.2 Show that an n-element heap has height $\lfloor \lg n \rfloor$



$$n = 2^0 + 2^1 + \cdots + 2^{h-1} + x \qquad 1 \le x < 2^h$$

$$2^h \le n \le 2^{h+1} - 1 \quad \} \text{ 6.1.1}$$

$$n = 2^h - 1 + k \qquad 1 \le k < 2^h$$

$$\lg(n - k + 1) = h \qquad 0 \le \lg k < n$$

$2^h - 1$

$$h = \lg(n - k + 1)$$

$$1 \leq k < 2^h$$

$$-1 \geqslant -k > -2^h$$

$$0 \geqslant -k+1 > -2^h + 1$$

$$n \geqslant n-k+1 > -2^h + 1 + n$$

$$\lg n \geqslant h > \lg(n - 2^h + 1)$$

$$2^h \leq n \leq 2^{h+1} - 1$$

$$\Rightarrow \quad h = \Theta(\log n)$$

Implementing heaps

→ parent (i)

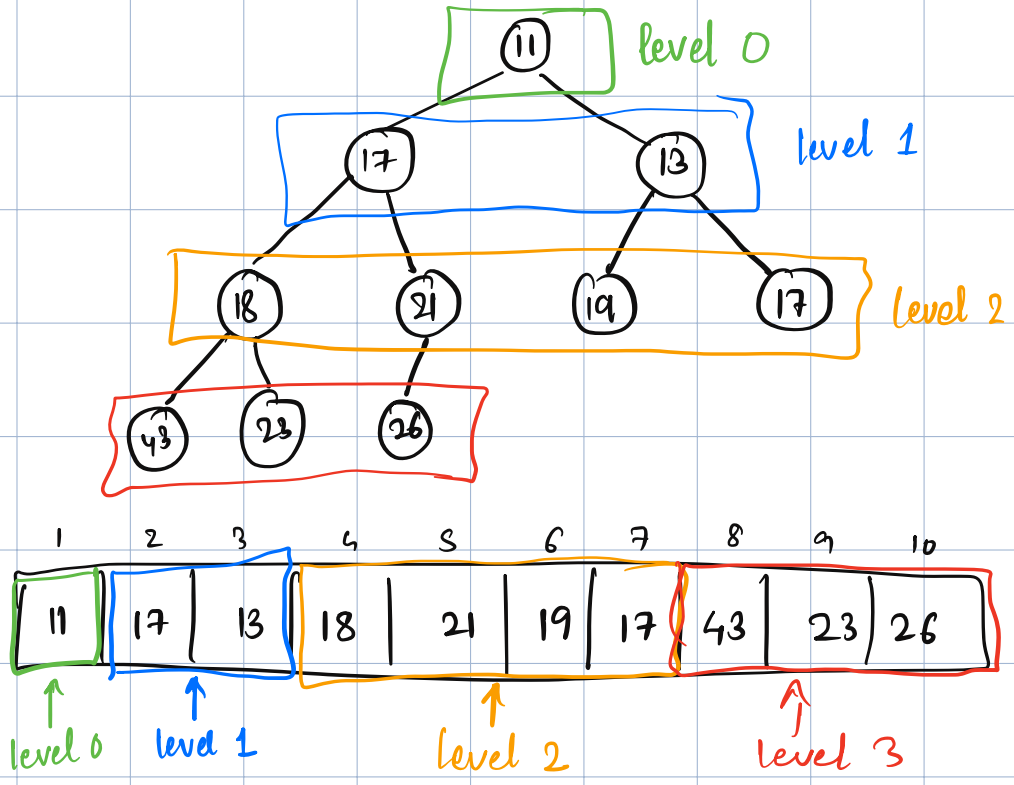
return $\lfloor i/2 \rfloor$

→ left (i)

return $2i$

→ right (i)

return $2i + 1$



$h = 0 \rightarrow$ 1

$h = 1 \rightarrow$ 2 to 3

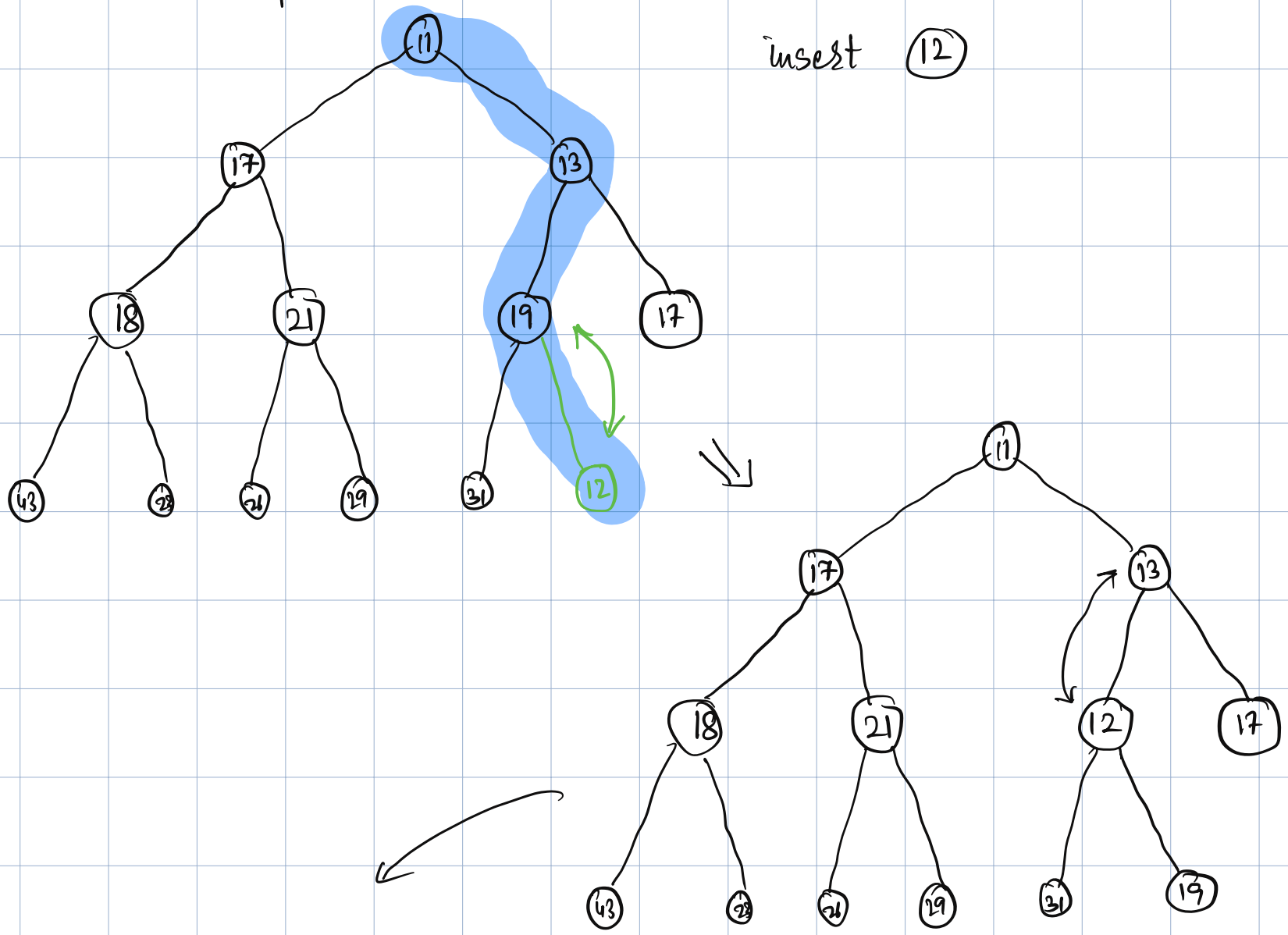
$h = 2 \rightarrow$ 4 to 7

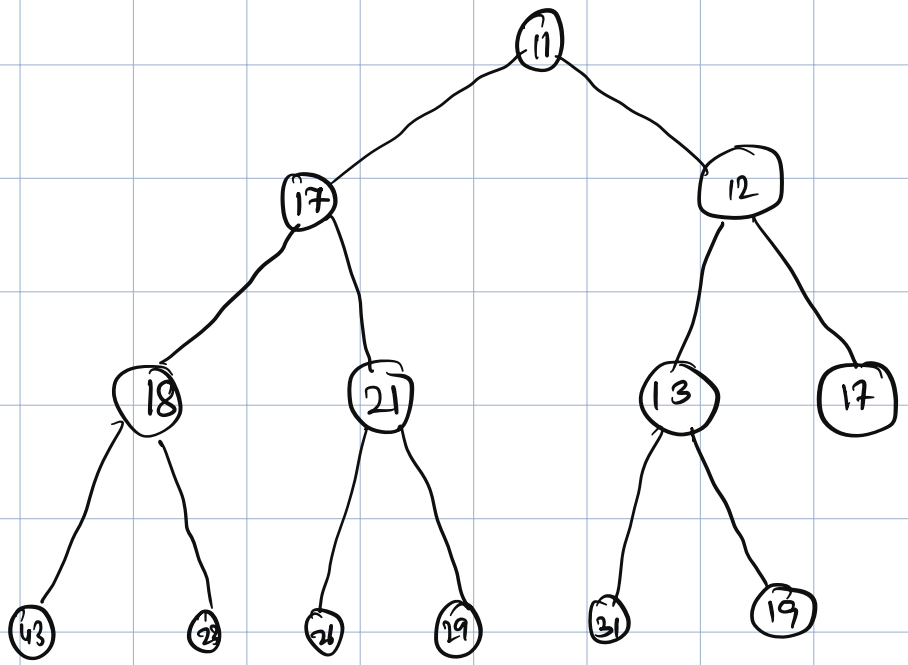
$h = 3 \rightarrow$ 8 to 15

$x < \lfloor x \rfloor \leq x + 1$

2^h to $2^{h+1} - 1$

Insertion in a heap





Proof of correctness

- The only node whose content changes are on the path from newly inserted node to root

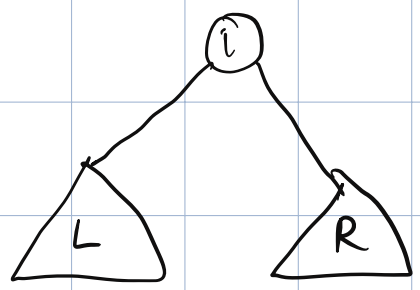
- We swap nodes (if required) on this path

- Heap property is not violated.

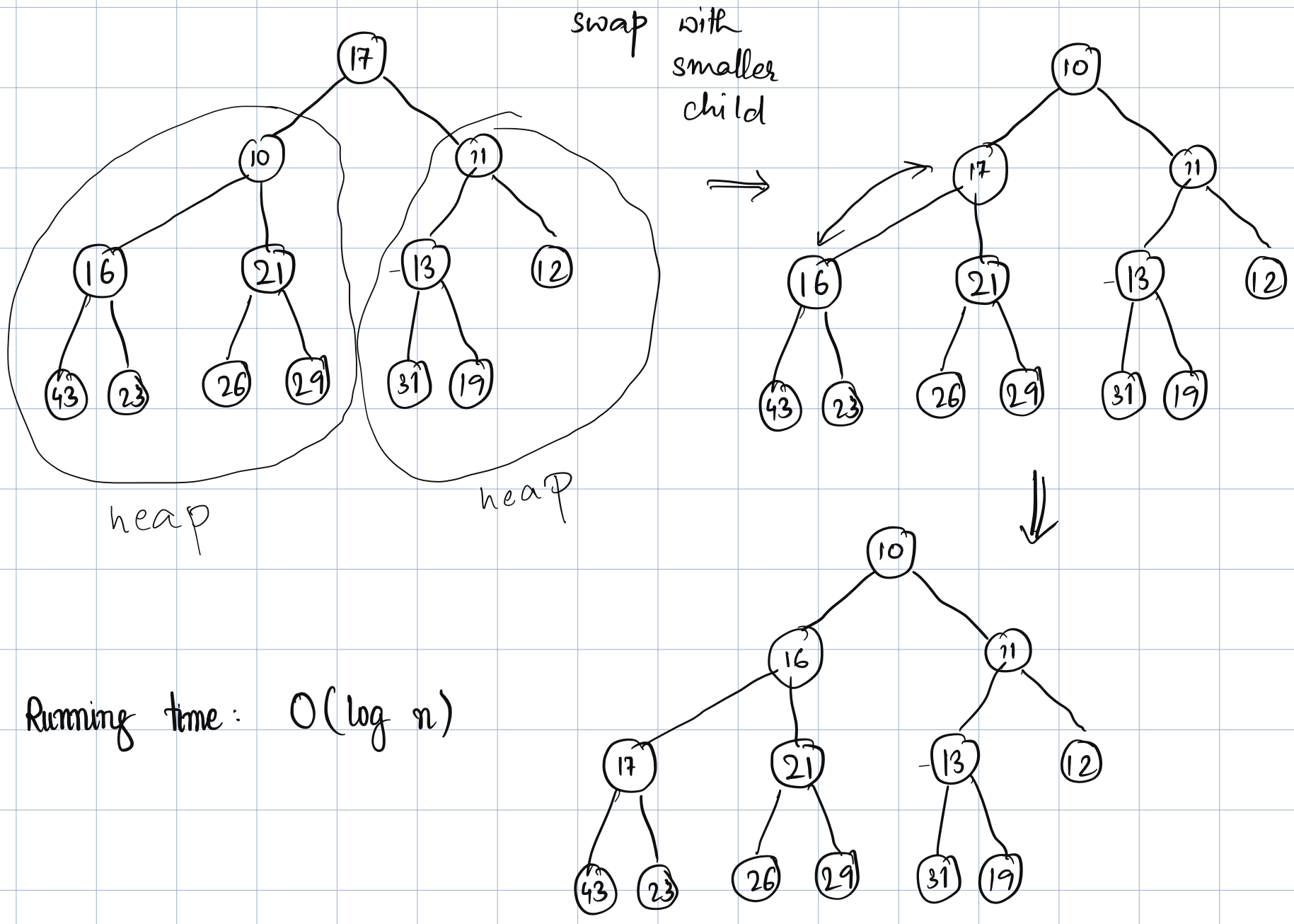
Time complexity : $O(\log n)$

Heapify

- i be the index into array A
- Binary tree rooted at left(i) and right(i) are heaps (L, R are heaps



- A[i] may be bigger than its children, violating heap property



Running time: $O(\log n)$

```
Min - heapify (A, i)
    l  = left (i)
    r  = right (i)
    if ( l < A.heap_size  &&  A[l] < A[i] )
        smallest = l
    if ( r < A.heap_size  &&  A[r] < A[smallest] )
        smallest = r
    if ( smallest ≠ i )
        swap ( A[i], A[smallest] )
        Min_heapify ( A , smallest )
```

$\Theta(1)$

Worst case: recurse all the way to a leaf.

Assume: left is complete binary tree of height $h-1$

right is complete binary tree of height $h-2$.

$$n = 1 + (2^h - 1) + (2^{h-1} - 1)$$

$$\frac{\text{number of nodes in left subtree}}{\text{number of nodes}} = \frac{2^h - 1}{2^{h-1}(2+1) - 1}$$

$$= \frac{2 \cdot 2^{h-1} - 1}{3 \cdot (2^{h-1} - 1)}$$

$$T(n) \leq T(2n/3) + \Theta(1)$$

$$T\left(\frac{2n}{3}\right) \leq T\left(\frac{4n}{9}\right) + \Theta(1)$$

$$T\left(\frac{4n}{9}\right) \leq T\left(\frac{8n}{27}\right) + \Theta(1)$$

$$\vdots$$

$$T\left(\frac{2^{k-1}n}{3^{k-1}}\right) \leq T\left(\frac{2^k n}{3^k}\right) + \Theta(1)$$

$$T(n) \leq T\left(\frac{2^k n}{3^k}\right) + k\,\Theta(1)$$

$$\left(\frac{2}{3}\right)^k n = 1$$

$$\Rightarrow \left(\log_{3/2} n\right) = k$$

$$T(n) \leq O(1) \cdot \log_{3/2} n$$

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

$T(n) = \Omega(\log n)$

$\rightsquigarrow$ worst case: go all the way upto leaf

Building a heap

Build_heap (A)

for $i = \lfloor n/2 \rfloor$ down to 1

heapify (A, i)

$\rightarrow$ $O(n \log n)$ $\rightarrow$ tighter bound possible

Correctness: induction on i,

all trees rooted at $m > i$ are heaps

Assume $n = 2^k - 1$ (a complete binary tree)

→ For the $n/2$ nodes of height 1 ⟶ at most 1 swap

→ For the $n/4$ nodes of height 2 ⟶ at most 2 swaps

→ For the $n/2^i$ nodes of height i ⟶ at most i swaps

$$\therefore T(n) = O\left(\frac{n+1}{2} + \frac{n+1}{4}\cdot 2 + \frac{n+1}{8}\cdot 3 + \cdots + 1\cdot k\right)$$

$$= O\left((n+1)\underbrace{\sum_{i=1}^{\lfloor \lg n \rfloor} \frac{i}{2^i}}_{\leq 2}\right)$$

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-|x|}, \quad |x| < 1$$

$$\downarrow \frac{d}{dx}$$

$$\sum_{i=1}^{\infty} i\, x^{i-1} = \frac{x}{(1-|x|)^2}$$

2024|10|17

Data Structure for set membership query

Problem Statement: Given a set S of universe U

$S \subseteq U$, $S = \{x_1, \ldots, x_n\}$

construct a data structure to answer queries

Is $y \in S$?

U = set of alphabets

S = some subset of alphabet

σ = abaceea ...

$S = \{a, b, c, d, e\}$

stream (σ)

once an element is read, it cannot be read again

$p \in S$ No

$d \in S$ Yes

Data structure should be

- space efficient
- fast

1 solution

↓

array → read from stream, if duplicate, don't add

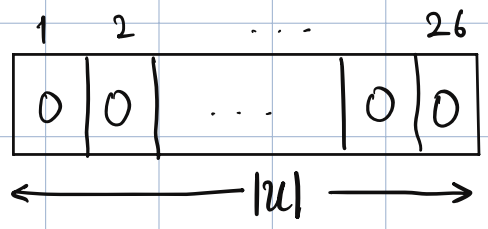
2nd solution

Bit vector

← 1 bit for each possible stream element

~ hashing (very primitive)

① Bit vector: One bit for each possible elements of U



Space complexity $O(|U|)$

Time complexity $O(1)$

② BST approach: Create a BST of elements in S.

space complexity $|S| \log |U| = O(n \log |U|)$

Assume ordering in $|U|$

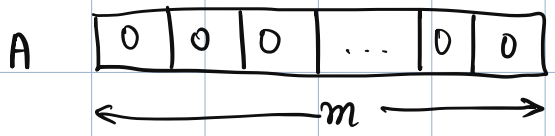
Ignore duplicates

Time complexity $O(\log n)$

Bloom filter (1970)

- Universe $\mathcal{U}$, set $S \subseteq \mathcal{U}$, $|S| = n$
- Maintain a bit vector (array that stores 0/1) of size m.

- Initialize bit vector to all zeroes



- k universal hash functions $h_1, \ldots, h_k : \mathcal{U} \to \{1, \ldots, m\}$

Query time complexity

Bit vector → faster
↳ but space complexity ↑

We need less space complexity with similar time complexity

$$\underset{h \sim H}{P}\left[h_i(a) = j \right] \leq 1/m$$

$\forall\ i \in \{1, \dots k\}$
$j \in \{1, \dots, m\}$
$a \in S \subseteq U$

ex: dot product hash family

To insert $x \in S$

$A[h_1(x)] \leftarrow 1$
$A[h_2(x)] \leftarrow 1$
$\vdots$
$A[h_k(x)] \leftarrow 1$

To answer a query $y \in S$:

Check whether $A[h_i(y)] = 1$ $\forall\ i = 1 \dots k$

if $\forall i\ A[h_i(y)] = 1$ then report $y \in S$

otherwise report $y \notin S$

Example: $m = 10$, $k = 3$

1	2	3	4	5	6	7	8	9	10

$H(x) = \{ h_1(x), h_2(x), h_3(x) \}$

$H(x_1) = \{1, 4, 9\}$

1	2		4	5	6	7	8	9	10
1		0	1	0	0	0	0	1	0

$H(x_2) = \{4, 5, \}$

1	2	3	4	5	6	7	8	9	10
1	0	0	1	1	0	0	1	1	0

Query Y_1

$H(Y_1) = \{2, 4, 8\}$ No ✓

Query Y_2

$H(Y_2) = \{1, 5, 8\}$ Yes ✗ (May be reporting false +ve FP)

Outputting

no ⇒ sure that answer is correct

yes ⇒ may be incorrect

Come up with expression for no. of false positives

Use calculus to minimize

Want to choose parameters such that F.P probability is minimized.

Once all elements are hashed then probability that a particular bit remains zero

$$p = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

$\frac{1}{m}$ → for hash

$\left(1 - \frac{1}{m}\right)$ → for 0

each element is hashed k times and there are n elements

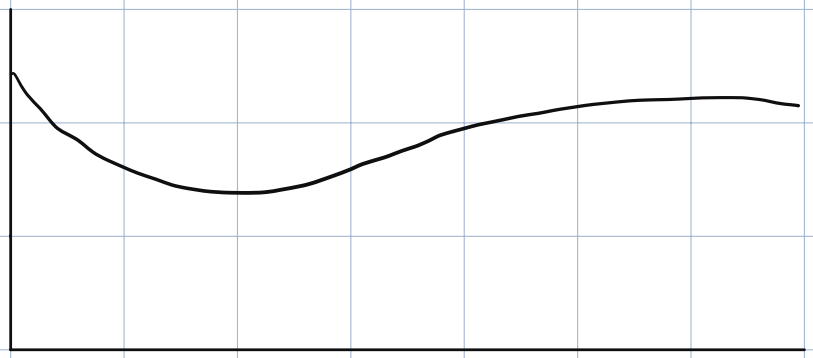
Probability of false +ve : from prob that a set of k specific bits are 1

$$f = (1-p)^k = \left(1 - e^{-kn/m}\right)^k$$

$$= \left(1 - \underbrace{\frac{1}{e^{kn/m}}}_{k\uparrow \ FP\uparrow}\right)^{k}$$

$k\uparrow$ $FP\uparrow$

$k\uparrow$ $FP\uparrow$



$$f = \left(1 - e^{-kn/m}\right)^k$$

$$= \exp\left(k \underbrace{\ln\left(1 - e^{-kn/m}\right)}_{g}\right)$$

$$\frac{\partial g}{\partial k} = 0 \qquad k = \left(\frac{m}{n}\right) \ln 2$$

$$\frac{\partial^2 g}{\partial k^2} > 0$$

f and g are monotonic

Bloom filter
- → browsing history
- → web crawler
- → ML applications