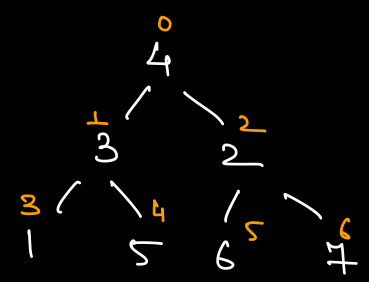
CBT ✓

CBT can be implemented using Array

Heap
- Min Heap
- Max Heap.

$$A : \{ \underset{4}{\overset{0}{}} \quad \underset{3}{\overset{1}{}} \quad \underset{2}{\overset{2}{}} \quad \underset{1}{\overset{3}{}} \quad \underset{5}{\overset{4}{}} \quad \underset{6}{\overset{5}{}} \quad \underset{7}{\overset{6}{}} \}$$



Parent
2 → 5, 6

Parent
1 → 3, 5

$i \rightarrow 2i+1,\ 2i+2$

$$\boxed{x \Longrightarrow \frac{(x-1)}{2}}$$

Heap :

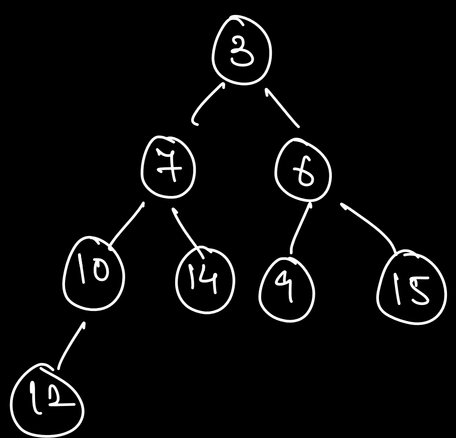
→ A Binary Tree is said to be a heap if

i) it is a CBT

ii) every node's value >= Both children
↳ MAX Heap

OR

every node's value <= Both children
↳ MIN Heap



CBT ✓

Min Heap

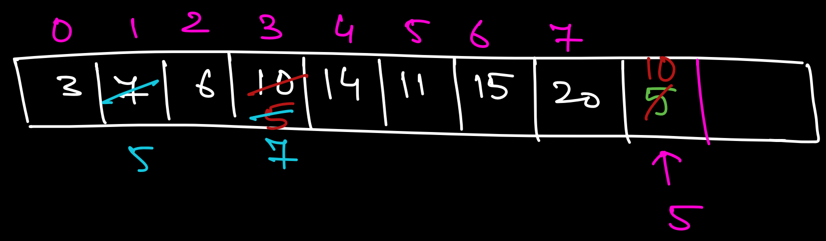
* Heap Operations

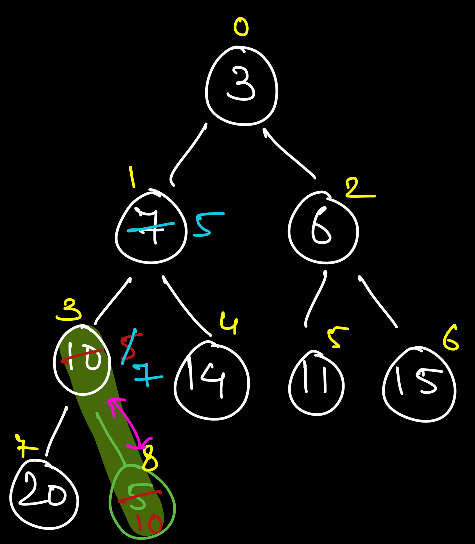
Max Heap

Min Heap

⇒ Min Heap Operations

list :





⇒ heapify

Min Heap

⇒ Insert 5 in min heap

index	parent	if A[parent] > A[index]
8	3	Swap
3	1	Swap
1	0	⇒ × Break.

3, 5, 6, 7, 14, 11, 15, 20

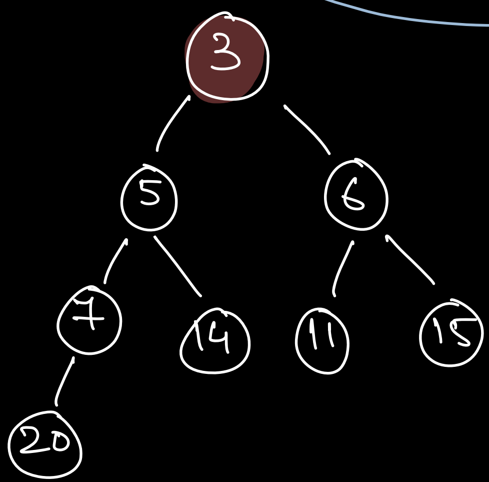
TC: O(log N)

* getMin() ⇒ return A[0]

TC: O(1)

* deleteMin()

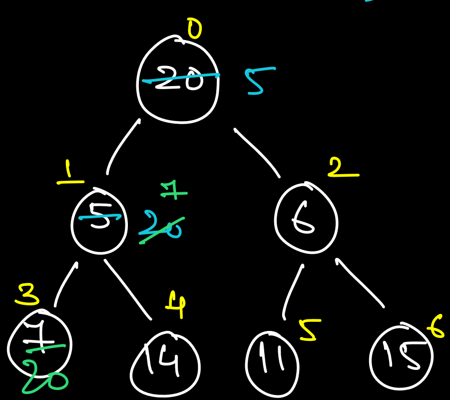
list : 3, 5, 6, 7, 14, 11, 15, 20



swap(A[0], A[N-1])

20, 5, 6, 7, 14, 11, 15, [3

x



index	l	r	min_index	A[index] > A[min_index]
0	1	2	1	swap
1	3	4	3	swap
3	7	8		

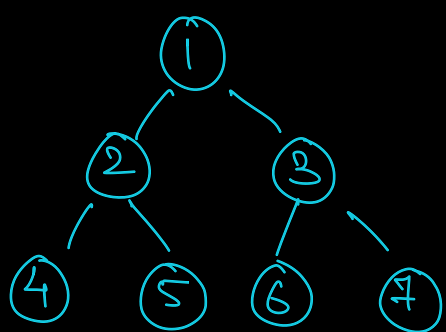
$\underbrace{7 \quad 8}_{\text{doesn't exist in the list}}$

⇒ **Break.**

TC: $O(\log N)$

⇒ **Search TC in Min Heap** ⇒ $O(N)$

1 2 3 4 5 6 7



Min Heap

⇒ Sorted & Unsorted array, both can be **Min Heap**

* Min Heap

insert() → $O(\log N)$

deleteMin() → $O(\log N)$

getMin() → $O(1)$

Search() → $O(N)$

delete(x) → search + delete

$O(N) + O(\log N)$

→ $O(N)$

* Max Heap

insert() → $O(\log N)$

deleteMax() → $O(\log N)$

getMax() → $O(1)$

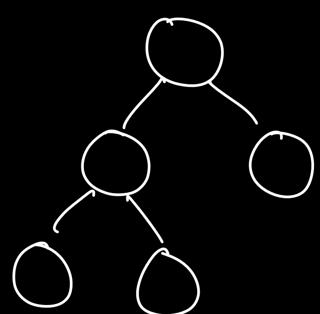
Search() → $O(N)$

delete(x) → $O(N)$

* Balanced Binary Search Tree

⇓

$|Ht(LST) - Ht(RST)| <= 1$

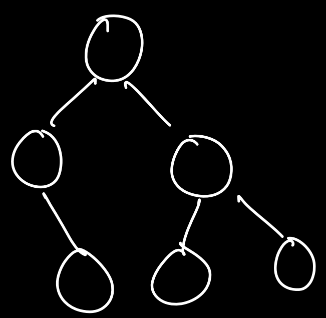


All operations in BBST → $O(\log N)$

* CBT vs Balanced Binary Tree

⇒ Every CBT is a Balanced Binary Tree

⇒ Every Balanced ~~Binary~~ Tree is a CBT ?
↳ NO



Balanced B.T ✓

CBT ✗

Height(CBT) $\simeq \log N$

$\simeq$ Height(Balanced B.T)

	Heap	BBST
insert	$\log N$	$\log N$
getMin() | getMax()	$O(1)$	$\log N$
deleteMin() | deleteMax()	$\log N$	$\log N$
Search()	$O(N)$	$\log N$
delete(x)	$O(N)$	$\log N$

delete(x)
↓
delete a random element

⇒ search + delete
search ↓ N, delete ↓ $\log N$

* insert()
* getMin() | getMax()
* deleteMin() / deleteMax()

} Heap.

* Pre-defined Library for Heap DS :

1. C++ : priority_queue (STL)
2. Java : PriorityQueue <-> Min Heap / Max Heap.
3. Python : heapq
4. C# : —
5. JS : 

Q: Given N **distinct** elements, find K smallest elements in Array. **K < N**

$$A[10] : \begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8 & 3 & 10 & 4 & 11 & 2 & 7 & 6 & 5 & 1 \end{array}$$

K = 4

↳ 4 smallest elements.

⇒ {1, 2, 3, 4}

$$A[9] : \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ -3 & 6 & 2 & 0 & 8 & 7 & 10 & 4 \end{array}$$

K = 3

↳ {-3 0 2}

I In every iteration, get the smallest element & swap it with index i.

⇒ Repeat the process K times.

TC : O(K * N)
SC : O(1)

} Selection sort

II Sort

Sort the array & return first k elements.

TC: $N \log N$

SC: $O(N) \rightarrow$ Merge sort

$O(\log N) \rightarrow$ Quick sort

III Min Heap

$\Rightarrow$ Insert all array elements into Min Heap & call getMin() & deleteMin() k times.

$$A[9]: \{ \overset{0}{-3} \;\; \overset{1}{6} \;\; \overset{2}{2} \;\; \overset{3}{0} \;\; \overset{4}{8} \;\; \overset{5}{7} \;\; \overset{6}{10} \;\; \overset{7}{4} \} \quad k=3$$

Min Heap

~~-3~~ 6 ~~2~~ ~~0~~

8 7 10 4

* Create a Min Heap

$\rightarrow$ getMin() : -3

$\rightarrow$ deleteMin()

$\rightarrow$ getMin() : 0

$\rightarrow$ deleteMin()

$\rightarrow$ getMin() : 2

$\rightarrow$ deleteMin()

k

TC: $N \log N + K * 1 + K * \log N$

: $N \log N + K \log N$ $(K < N)$

: $N \log N$

SC: $O(N)$

↳ If we are creating a Heap in new Array.

(IV) Max Heap

A[10]:

	0	1	2	3	4	5	6	7	8	9
A[10]:	{ 8	3	10	4	11	2	7	6	5	1 }

K = 4

* Create a Max Heap of size = 4.

	0	1	2	3	4	5	6	7	8	9
A[10]:	{ 8	3	10	4	11	2	7	6	5	1 }

~~8~~ 3 ~~10~~ 4
2 ~~7~~ ~~6~~
~~5~~ 1

Max Heap of size = K

Obs 1 : If ele > getMax() ⇒ ele can't be the ans

↳ $O(1)$ ⇒ Skip

Obs 2 : If ele < getMax

i) Insert in Heap

ii) deleteMax()

} $\log K$

TC : $\underbrace{K \log K}_{\text{Create a Max Heap of size = 4}} + \underbrace{(N-K) \log K}_{(N-K) \Rightarrow \text{insert + delete}}$

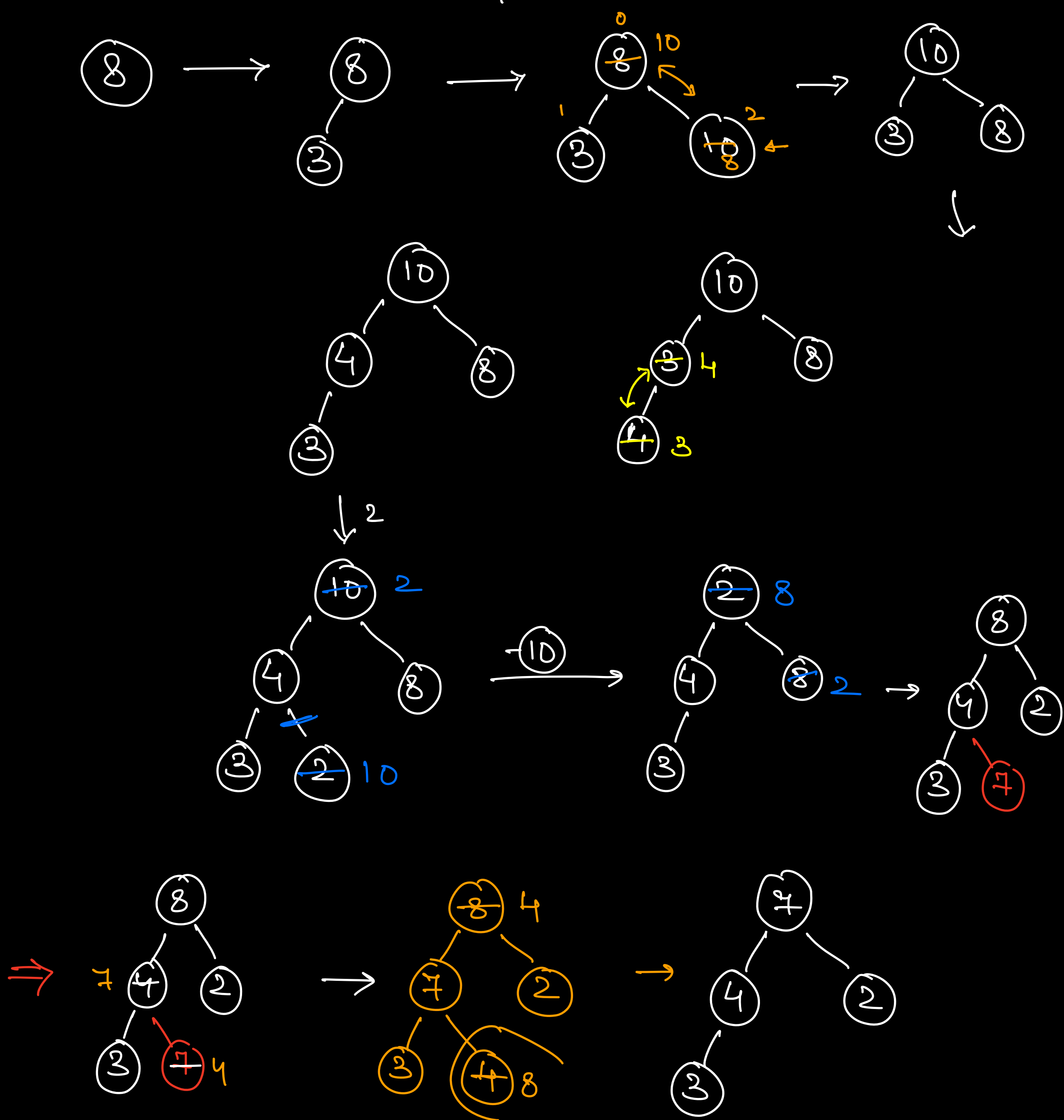
$N \log K$

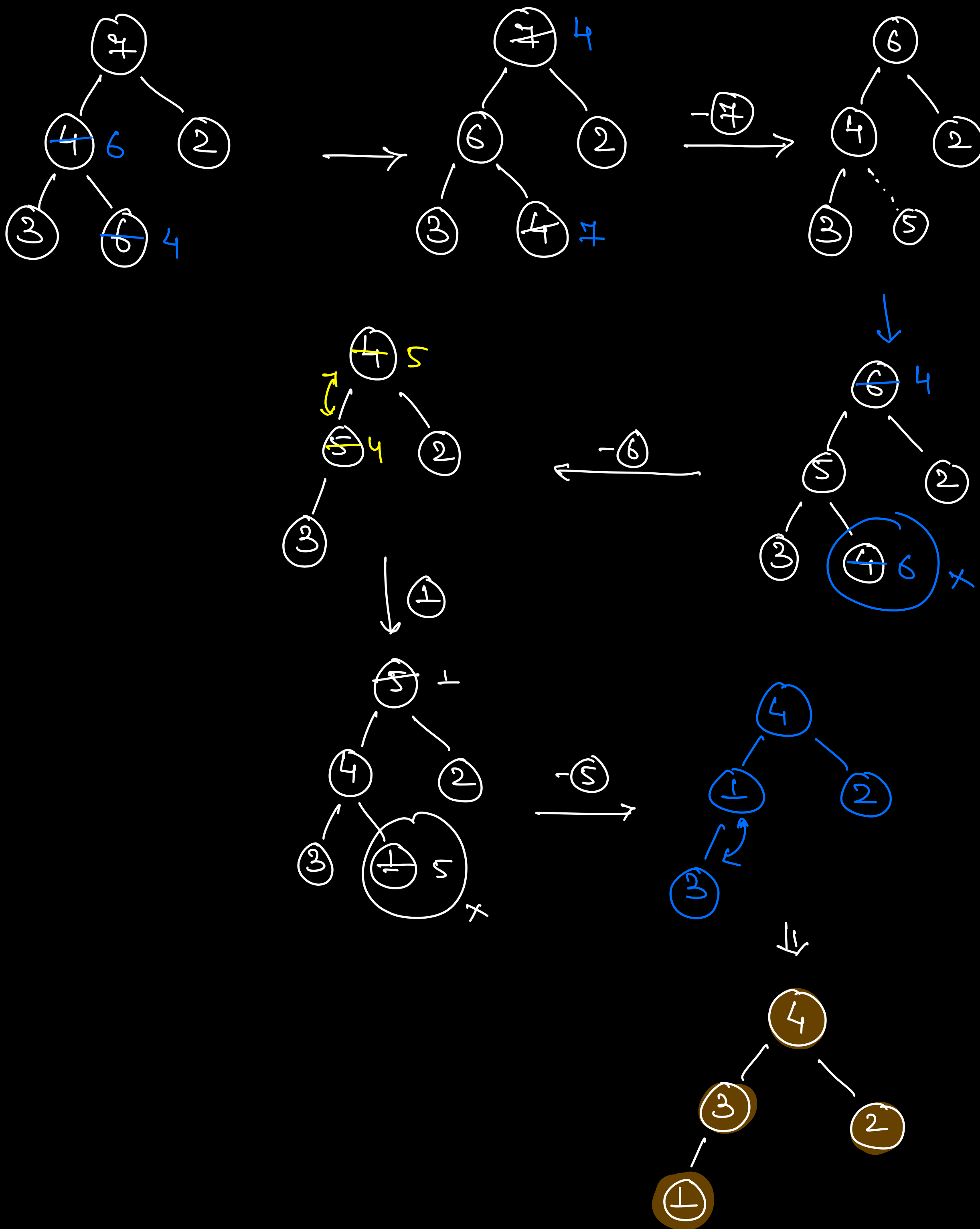
SC : $O(K)$

	0	1	2	3	4	5	6	7	8	9	
A[10] : {	8	3	10	4	11	2	7	6	5	1	}

k = 4

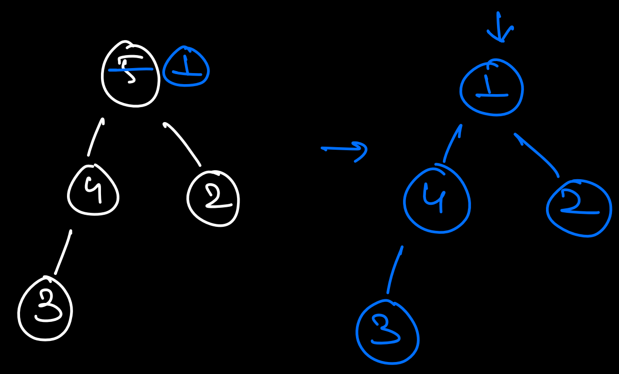
Max Heap

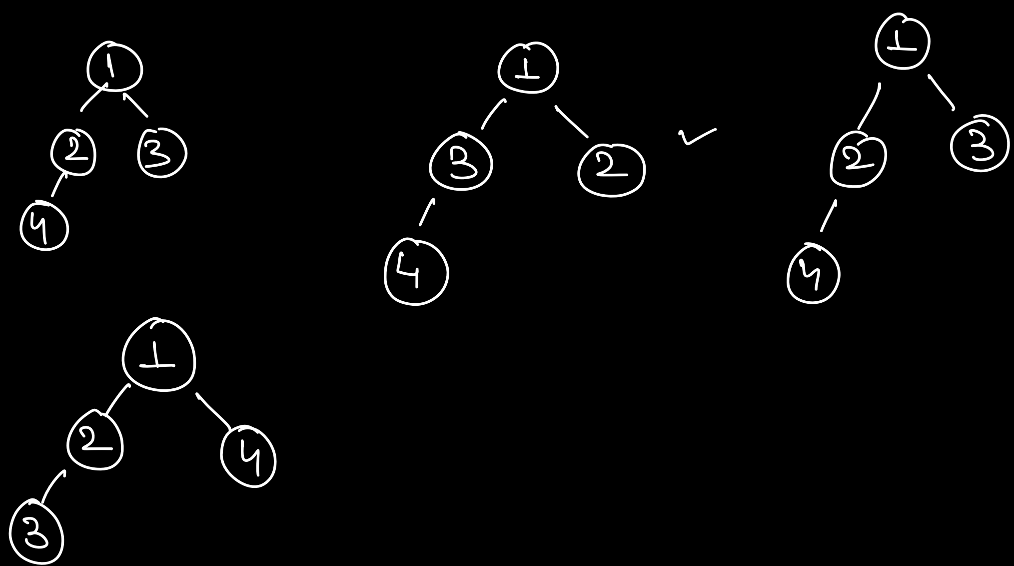




insert

deleteMax





Note

* How many heaps we can create from N distinct elements.

↳ Google this.