Simple sol^{n} → Sort the array in ASC order & return the K^{th} largest element.

$O(n \log n)$

But can we do better ??

array → heap $O(n)$

↳ Max heap → root → largest elem.

$\log n$

$O(K \log n + n)$

$(K-1)$ → K^{th} largest

1

$[3, 2, 1, 5, 6, 4]$ $k = 2$

if the length of the array would have been equal to k, what should be the ans.

Smallest element will be the k^{th} largest.

↳ Min Heap

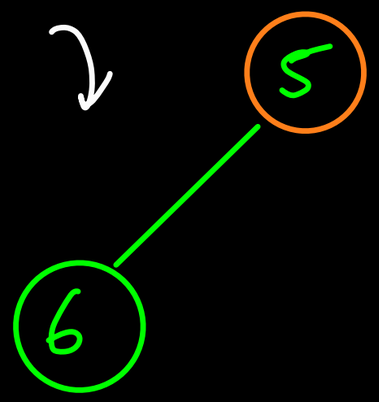
$10^5 \times \log(10^3)$ ← $k \log n$

[3, 2, 1, 5, 6, 4] ← i

k = 2

1 < root

2



Min heap

log k log k

incoming < root

↓

no way

(incoming > root)

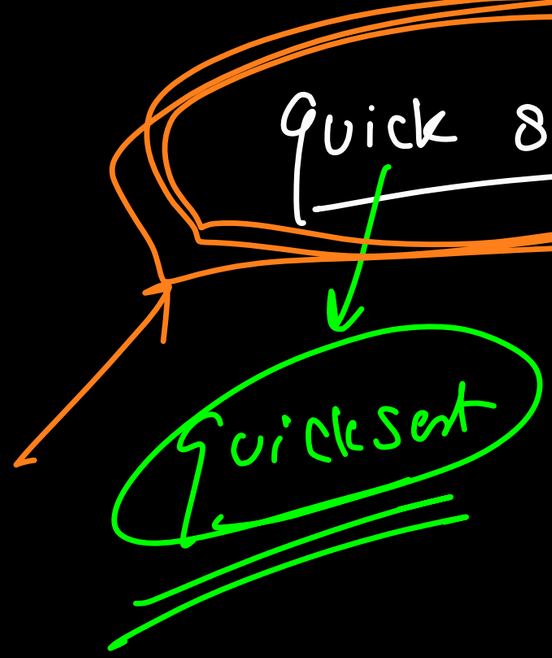
Let's start adding more elements but not inc the size of heap.

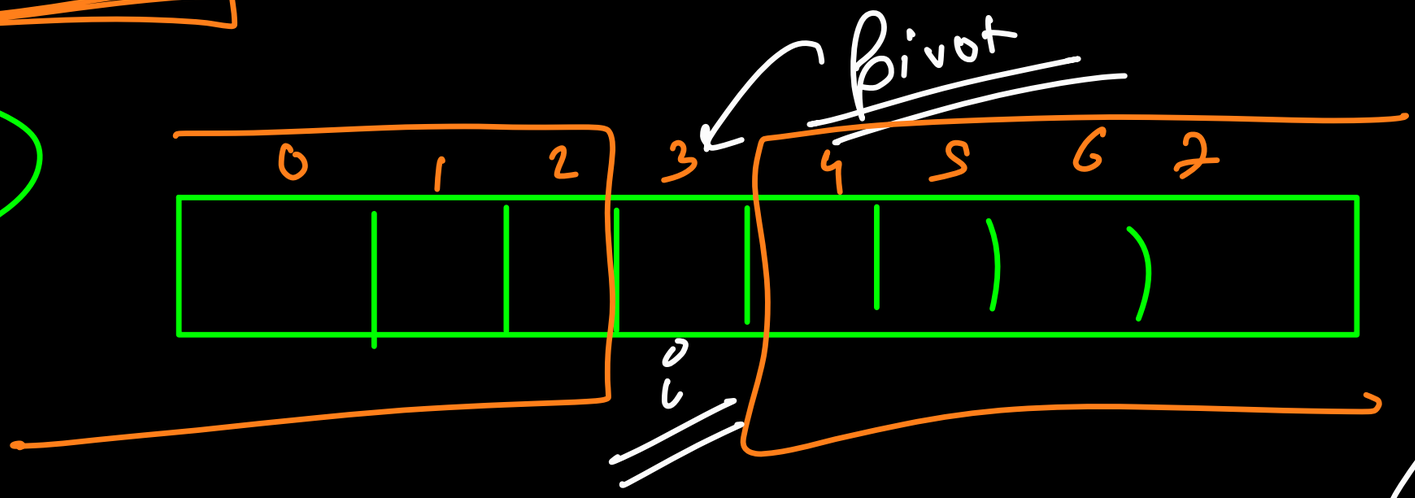
k <<< n

O(n log k) ⟸ $10^8 \times \log(10^3)$

Quick select → O(n)

K^{th} largest element

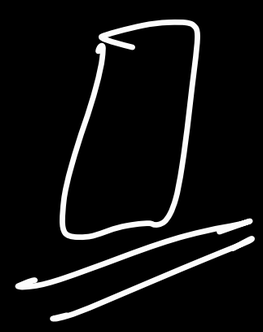




$(i < k)$

$i > k$

after sorting → 6^{th}



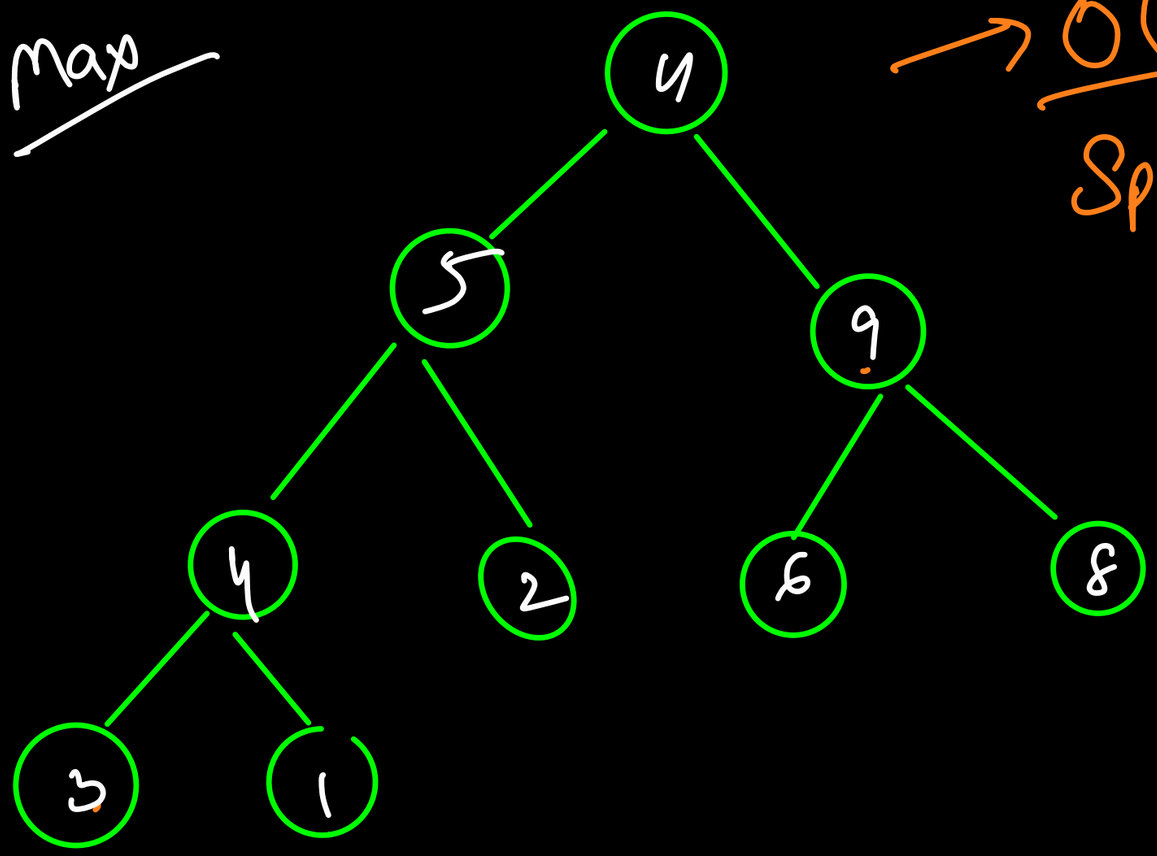
Q ⇒ Given an Array of integers, Convert this into a Heap (Max or Min) in less than O(nlogn).

new heap

0	1	2	3	4	5	6	7	8
4	5	6	1	2	9	8	3	11

assume a new heap

Max



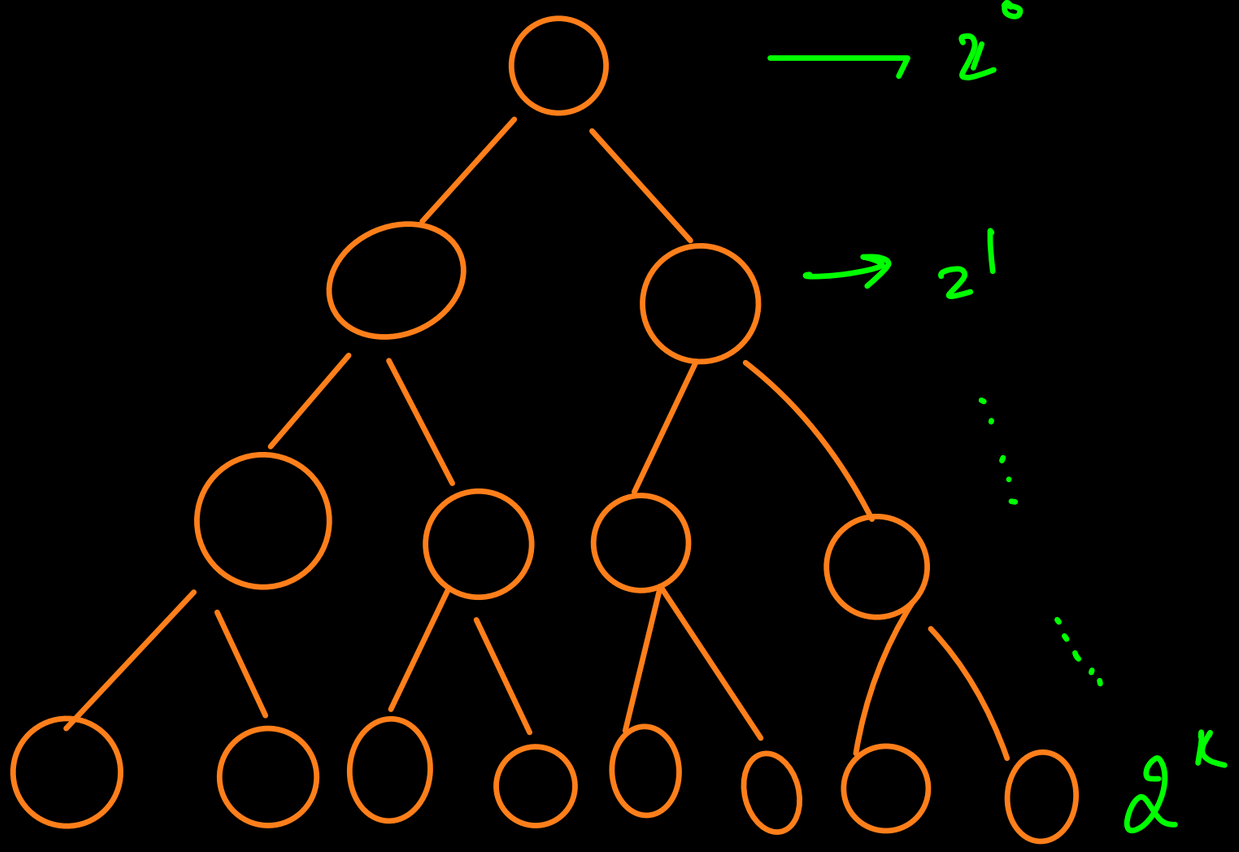
→ O(n) ← Time

Space → O(1) mimic insertion of elements as we would have done in a new heap.

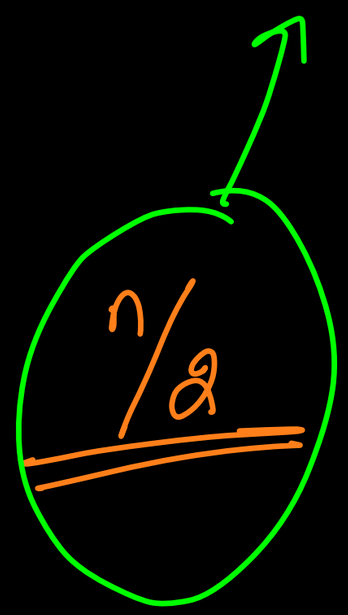
Space → O(1)

Time → O(n log n)

all leaf are already a heap



half of the heap nodes have to traverse the longest way.



$$T = 2^0 \times K + 2^1 \times (K-1) + 2^2 \times (K-2) \cdots\cdots 2^{K-1} \times 1 + 2^K \times 0$$

$$2T = 2^1 \times K + 2^2 \times (K-1) + 2^3 (K-2) \cdots\cdots 2^K \times 1 + \underline{2^{K+1} \times 0}$$

$$2T - T = -2^0 \times K + 2^1 (\underline{K - (K-1)}) + 2^2 (\underline{K-1-(K-2)}) +$$

$$2^3 (K-2-(K-3)) \cdots\cdots 2^K (1-0) +$$

$$T = -\underline{\underline{2^0}} \times K + \underline{2^1 \times 1} + 2^2 \times 1 + 2^3 \times 1 \cdots\cdots 2^K \times 1$$

$$= -K + \left(\frac{2 \times (2^K - 1)}{2-1} \right) \Rightarrow -K + 2^{K+1} - 2$$

$$T = 2^{K+1} - K - 2 \qquad K \approx \log_2 n$$

$$= 2^{\log_2 n + 1} - \log_2 n - 2 \Rightarrow 2n - \log n - 2$$

$$\Rightarrow \underline{O(n)}$$

Worst Case

$$T = \underbrace{2^0 \times 0}_{40} + 2^1 \times 1 + 2^2 \times 2 + 2^3 \times 3 \cdots\cdots 2^{k-1}(k-1) + 2^k \times k$$

$$T = 2^1 \times 1 + 2^2 \times 2 + 2^3 \times 3 \cdots\cdots 2^{k-1}(k-1) + 2^k \times k \quad (1)$$

↳ AGP

multiply LHS & RHS by 2

$$2T = 2^2 \times 1 + 2^3 \times 2 + 2^4 \times 3 \cdots\cdots 2^k(k-1) + 2^{k+1} \times k \quad (2)$$

(2) − (1) (Subtract)

$$2T - T = -2^1 \times 1 + 2^2(1-2) + 2^3(2-3) + 2^4(3-4) \cdots\cdots$$
$$2^k(k-1-k) + 2^{k+1} \times k$$

$$T = -2^1 \times 1 + 2^2(-1) + 2^3(-1) + 2^4(-1) \cdots\cdots 2^k(-1) + 2^{k+1} \times k$$

$$T = -1(2^1 + 2^2 + 2^3 \cdots\cdots 2^k) + 2^{k+1} \times k$$

$$T = -(2^1 + 2^2 + 2^3 \ldots\ldots\ldots 2^k) + 2^{k+1} \times k$$

$$T = -\left(\frac{2^1 \times (2^k - 1)}{2 - 1}\right) + 2^{k+1} \times k$$

$$T = -2(2^k - 1) + 2^{k+1} \times k \Rightarrow \quad (k \approx \log n)$$

$$= -2^{k+1} + 2 + 2^{k+1} \times k$$

$$T \Rightarrow 2^{k+1}(k-1) + 2$$

$$= 2^{\log_2 n + 1}(\log_2 n - 1) + 2 \Rightarrow 2 \times 2^{\log_2 n}(\log_2 n - 1) + 2$$

$$\Rightarrow 2 \times n(\log_2 n - 1) + 2$$

$$\Rightarrow 2n\log_2 n - 2n + 2 \Longrightarrow O(n \log n)$$

$$T = 2^1 \times 1 + 2^2 \times 2 + 2^3 \times 3$$

$$2T = 2^2 \times 1 + 2^3 \times 2 + 2^4 \times 3$$

$$2T - T = \left(2^2 \times 1 + 2^3 \times 2 + 2^4 \times 3\right) - \left(2^1 \times 1 + 2^2 \times 2 + 2^3 \times 3\right)$$

$$T = 2^2 \times 1 + 2^3 \times 2 + 2^4 \times 3 - 2^1 \times 1 - 2^2 \times 2 - 2^3 \times 3$$

$$T = 2^2 \times 1 - 2^2 \times 2 + 2^3 \times 2 - 2^3 \times 3 + 2^4 \times 3 - 2^1 \times 1$$

$$= 2^2(1-2) + 2^3(2-3) + 2^4 \times 3 - 2^1 \times 1$$

12	5	10	10	12	14	15

↑

add (10)

add (2)

add (5)

findMedian ⟶ 5

add (10)

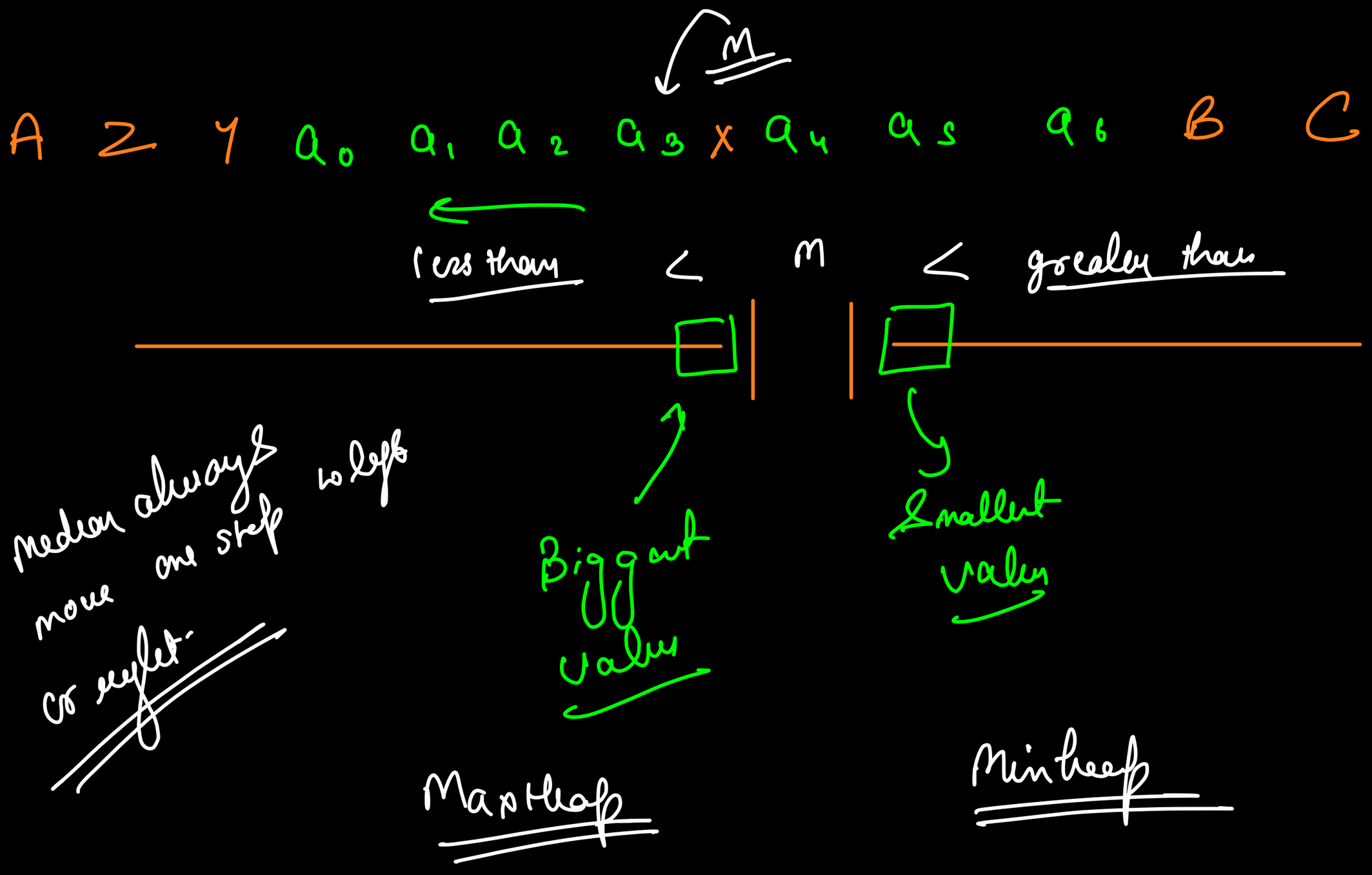
findMedian ⟶ 7.5

add (12)

add (14)

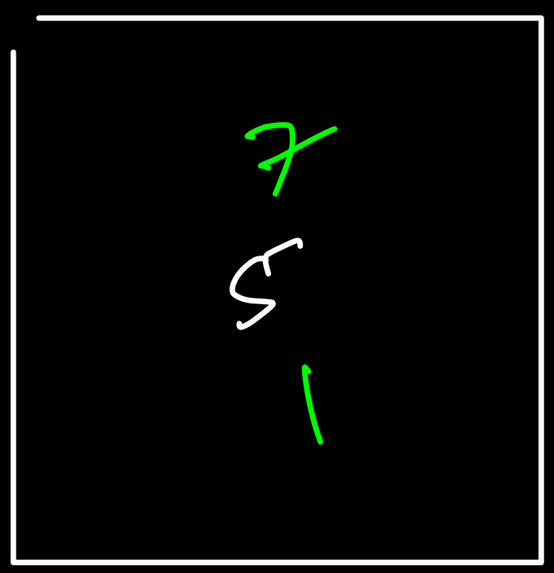
add (15)

add (1)

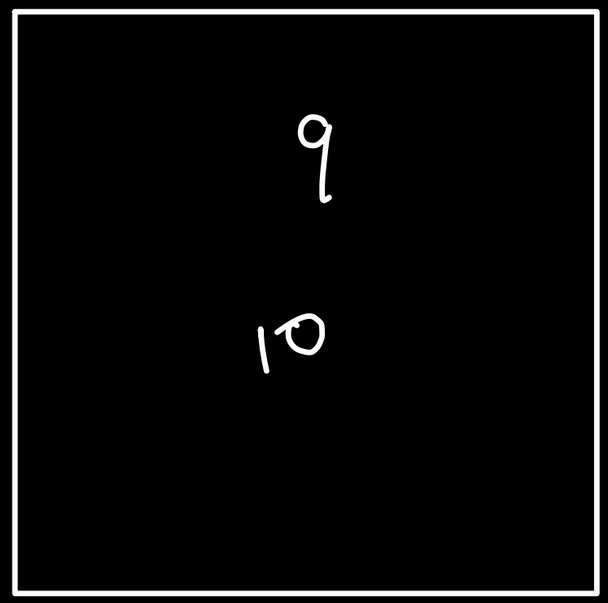


O(1)

n = 7



max heap



min heap

O(log n)

O(n)

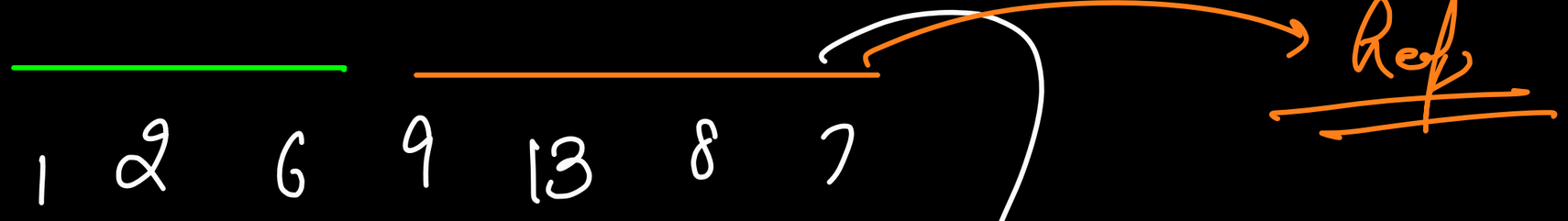
Size of MaxHeap == Size of minheap

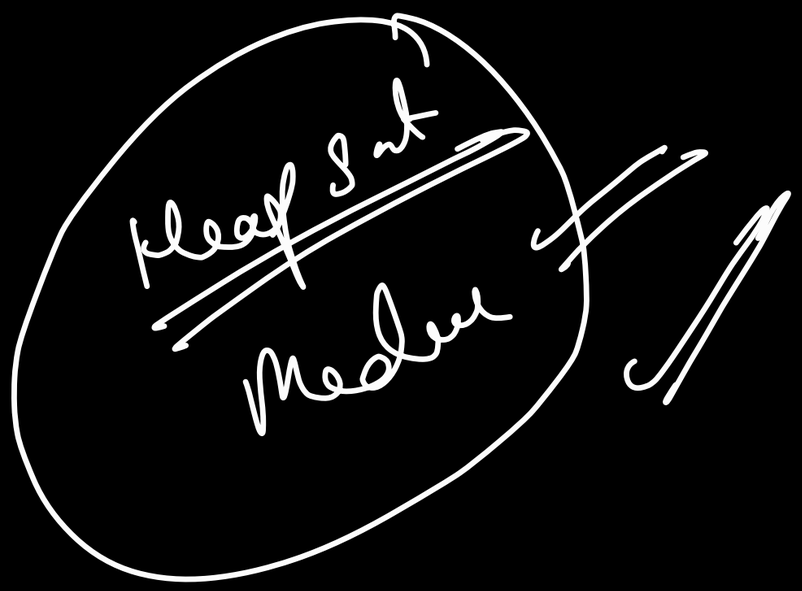
Size of MaxHeap = 1 + Size of minheap

median is avg of both root (even elements)

median is root of maxheap (odd elements)

Selection Sort → $O(n^2)$





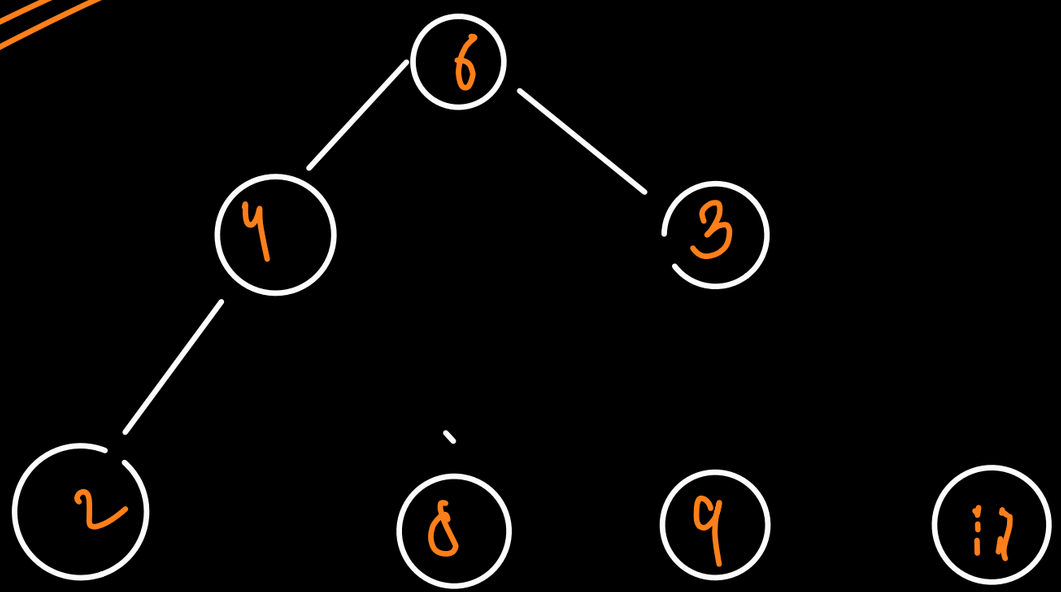
if the data is not already sorted what is one of the best techniques to get a min?

HeapSort

6	4	3	2	8	9	11

i

Max heap



11 9 8 6 4 3 2

rev

2	3	4	6	8	9	11

$O(n + n\log n) \rightarrow O(n\log n)$

Space → O(1)