SCALER

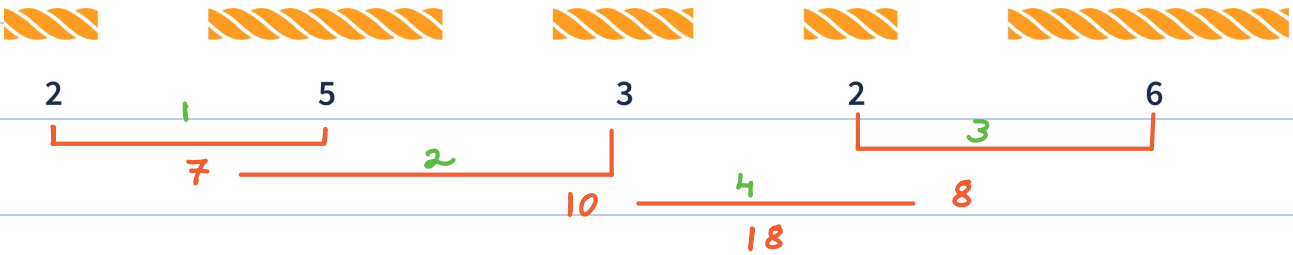
Heaps - 1

TABLE OF CONTENTS

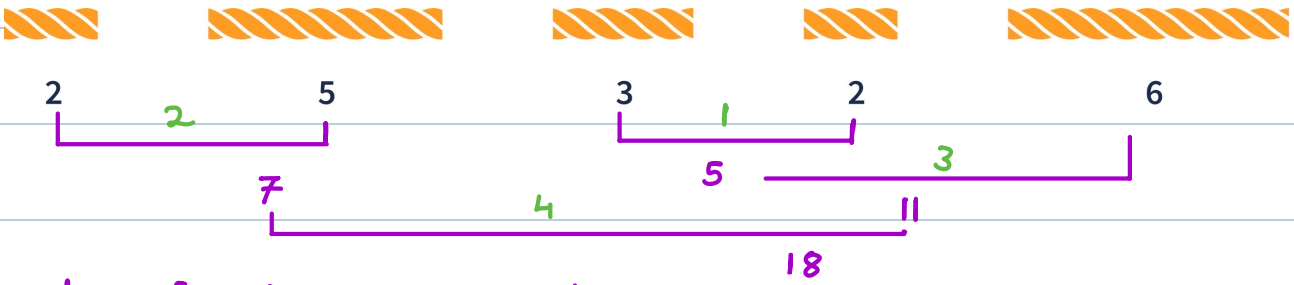
Connecting the Ropes

Given an array that represents the size of different ropes. In a single operation, you can connect two ropes. Cost of connecting 2 ropes is sum of the ropes of length of the ropes you are connecting. Find the minimum cost of connecting all the ropes.

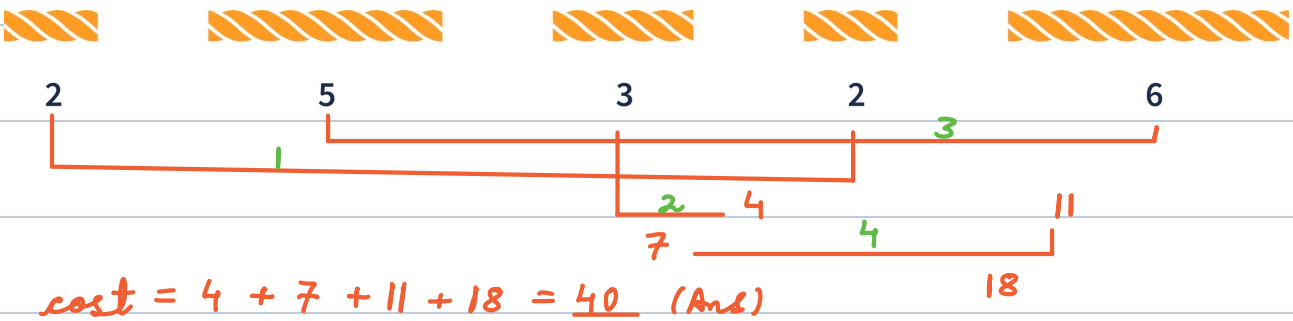
arr[ ] → [2 , 5 , 3 , 2 , 6]



cost = 7 + 10 + 8 + 18 = 43



cost = 5 + 7 + 11 + 18 = 41



cost = 4 + 7 + 11 + 18 = 40 (Ans)

Idea → connect smaller length ropes first.

Let say x < y < z.

$x+y$ < $x+z$ < $y+z$

$(x+y)+z$ $(x+z)+y$ $(y+z)+x$

min cost

Sol → Sort in ascending order & travel L → R.

A = [5 5 8 11 15] (0 1 2 3 4)

10, 18, ✗

cost = 10 + 18 + ...

Insertion Sort

TC = O(N²) SC = O(1)

A = [1 2 3 4] (0 1 2 3)

3, 6, 10

cost = 3 + 6 + 10 = 19

A = [5 6 8 10] (0 1 2 3)

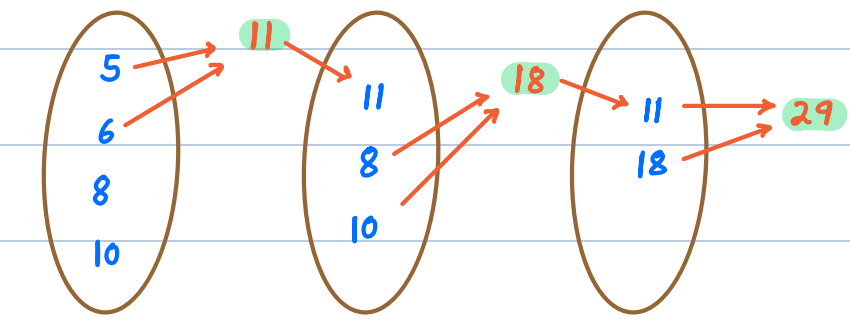
11, 18, 29

cost = 11 + 18 + 29 = 58 (Ans)

Let say, DS → insert element() } TC = O(log(N))

get min()

A = [5 6 8 10] (0 1 2 3)



cost = 11 + 18 + 29 = 58

TC = O((N-1) * 3 log(N))

= O(N log(N))

SC = O(N)

Heap Data Structure (a.k.a Priority Queues)

Binary Tree

Complete Binary Time (C.B.T) + Heap Order Property (H.O.P)

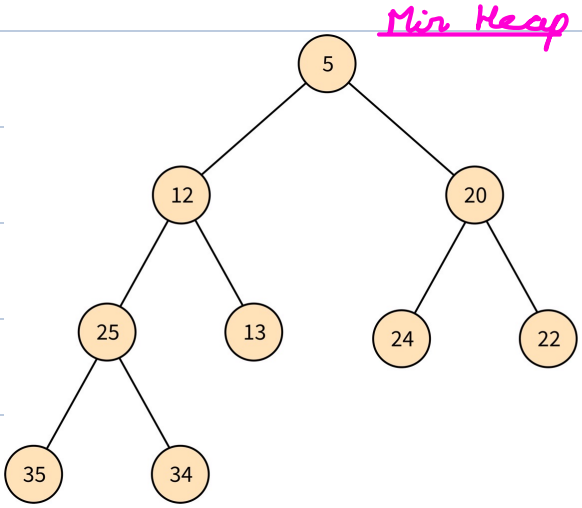
All levels must be completely filled except possibly the last level

and the last level must be filled from left to right.

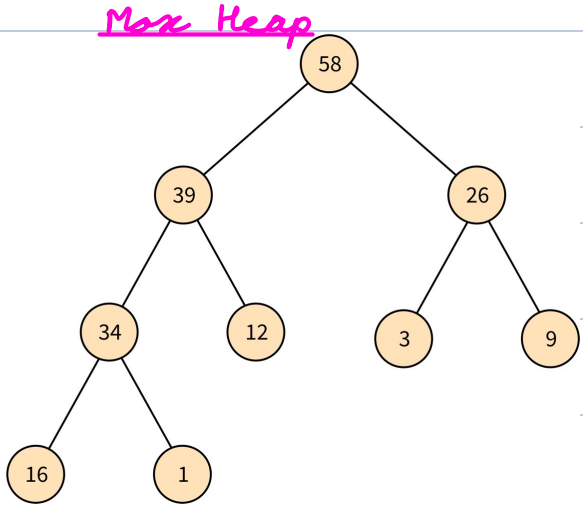
Types → min Heap → ∀ nodes, data <= children data

max Heap → ∀ nodes, data >= children data

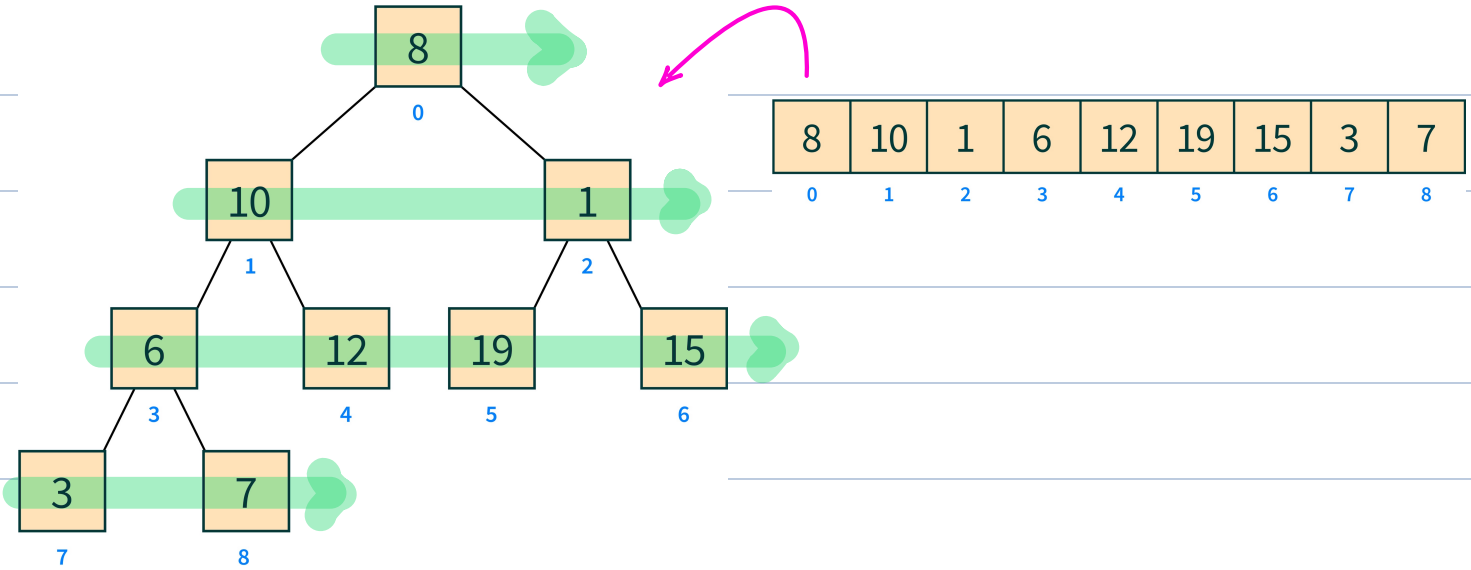
Min Heap



Max Heap



Visualise Arrays as Binary Tree



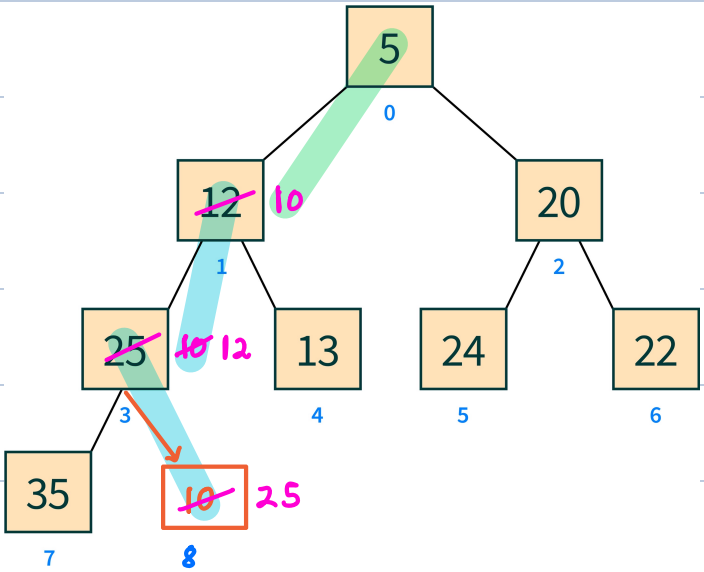
∀ nodes i,

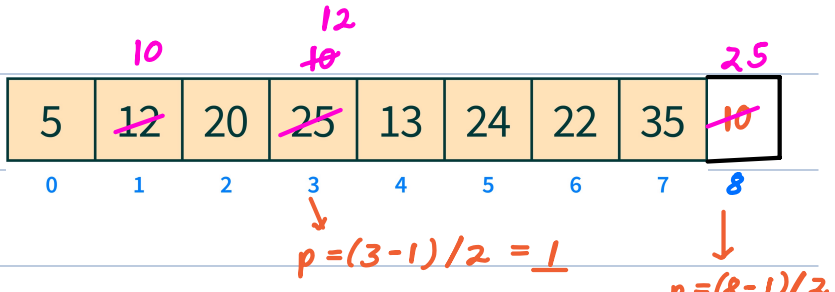
left child = i * 2 + 1

right child = i * 2 + 2

parent = (i - 1) / 2

Min Heap





→ Insert(10) ✓

→ Insert(3) → H.W

∀ nodes, data <= children data

Height of complete binary tree = O(log(N))

new insertion

A[n] = X

n++

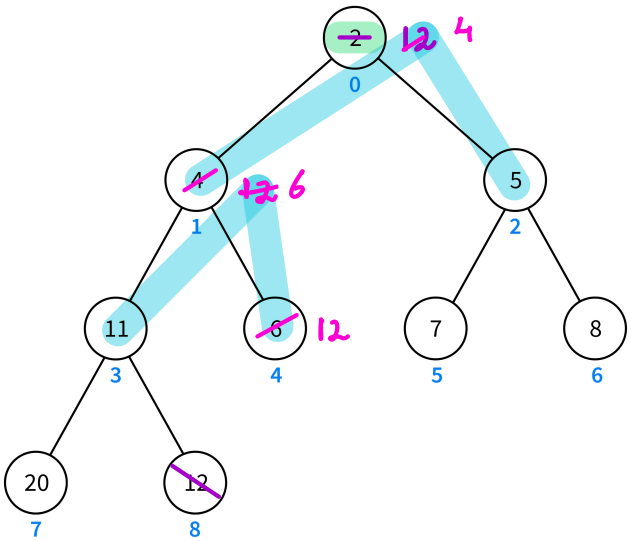
i = n

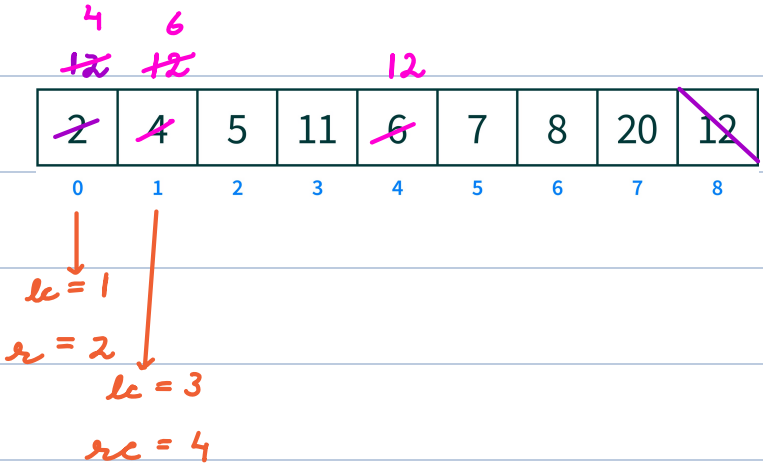
size of heap

```
while (i > 0) {
    p = (i - 1) / 2
    if (A[i] < A[p]) {
        swap(A, i, p)
        i = p
    } else break
}
```

TC = O(log(N)) SC = O(1)

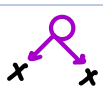
Extract-Min / Remove - Min

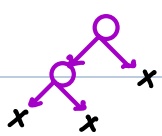




Heapify → Maintaining properties of heap after any operation.

```
swap (A, 0, n-1)          n-- // reducing 1 element
i = 0
while (i < n) {
    lc = 2*i + 1
    rc = lc + 1
    if (lc >= n)  break   // i → leaf node
    if (rc >= n) {
        if (A[lc] < A[i])
            swap (A, i, lc)
        break
    }
```





```
if (A[lc] < A[i] && A[lc] <= A[rc]) {
    swap(A, i, lc)
    i = lc
} else if (A[rc] < A[i] && A[rc] <= A[lc]) {
    swap(A, i, rc)
    i = rc
} else    break
}
```

TC = O(log(N)) SC = O(1)

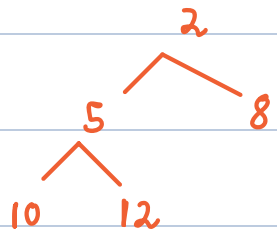
✪ Build a Heap [Interview Problem]

1) Insertion → ∀ elements, insert one by one.

TC = O(N log(N))

2) Sorting

A = [2 5 8 10 12] (indices 0 1 2 3 4)

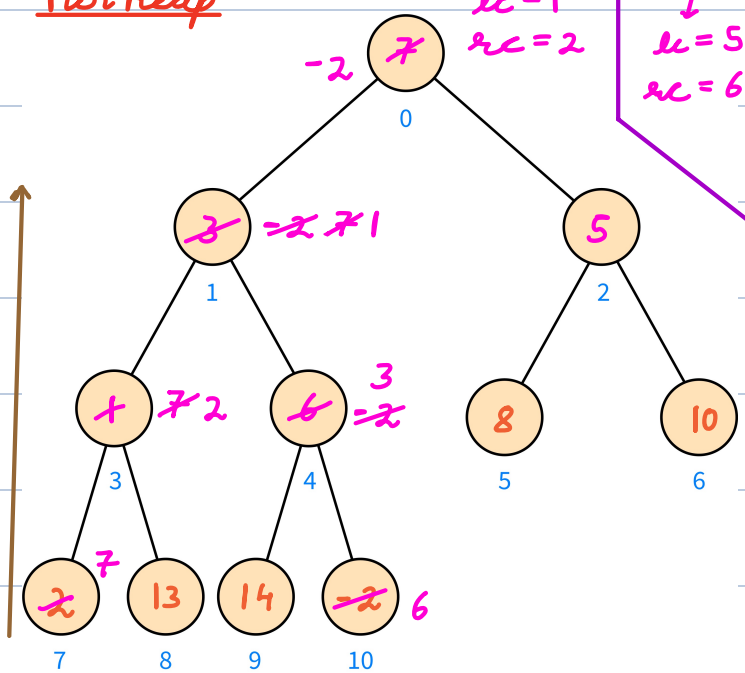


TC = O(N log(N))

3) Bottom Up →

A = [7 3 5 1 6 8 10 2 13 14 -2] (indices 0 1 2 3 4 5 6 7 8 9 10)

Min Heap



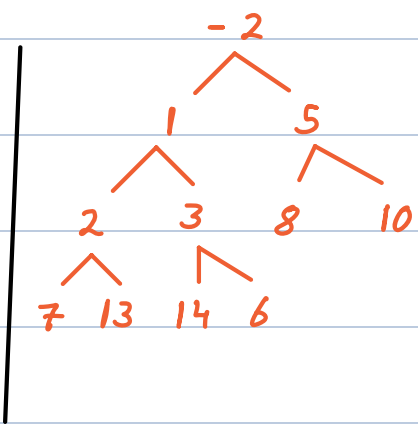
lc = 1, rc = 2

lc = 5, rc = 6

lc = 9, rc = 10

lc = 7, rc = 8

lc = 3, rc = 4



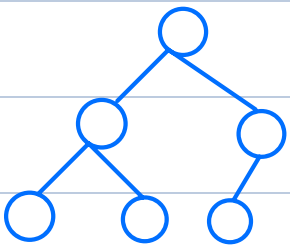
SC = O(1)

(inplace build)

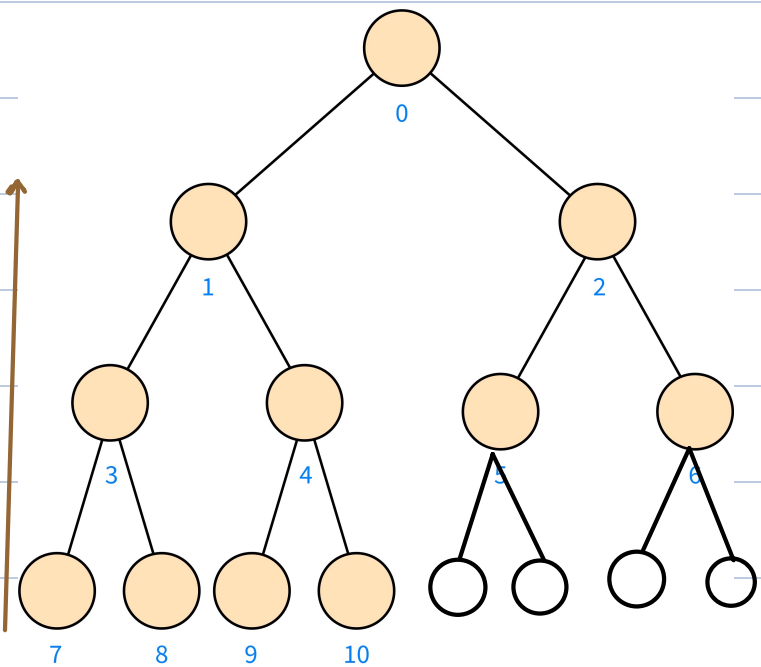
of leaves in complete binary tree

N → 1 2 3 4 5 6 ...

leaves → 1 1 2 2 3 3 ...



$\frac{N+1}{2}$ (half)



#nodes	# swaps
N/16	3
N/8	2
N/4	1
N/2	0

$$TC = \frac{N}{2} * 0 + \frac{N}{4} * 1 + \frac{N}{8} * 2 + \frac{N}{16} * 3 + \dots$$

$$= \frac{N}{2}\left(\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \dots\right) = \frac{N}{2} * 2 = N$$

$TC = O(N)$

$$S = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \cdots$$

$$-\ \frac{S}{2} = \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \cdots$$

$$\frac{S}{2} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = \frac{1/2}{1 - 1/2} = \frac{1/2}{1/2} = 1$$

$$\Rightarrow S = 2$$

SCALER

Heaps - 2

TABLE OF CONTENTS

Sort the array

	TC	SC
Merge Sort →	$O(N \log(N))$	$O(N)$
Quick Sort →	$O(N \log(N)) \rightarrow O(N^2)$	$O(\log(N)) \rightarrow O(N)$

Heap - Sort

Approach 1 → 1) Build Heap.

2) Extract min / max & insert in answer → N times.

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

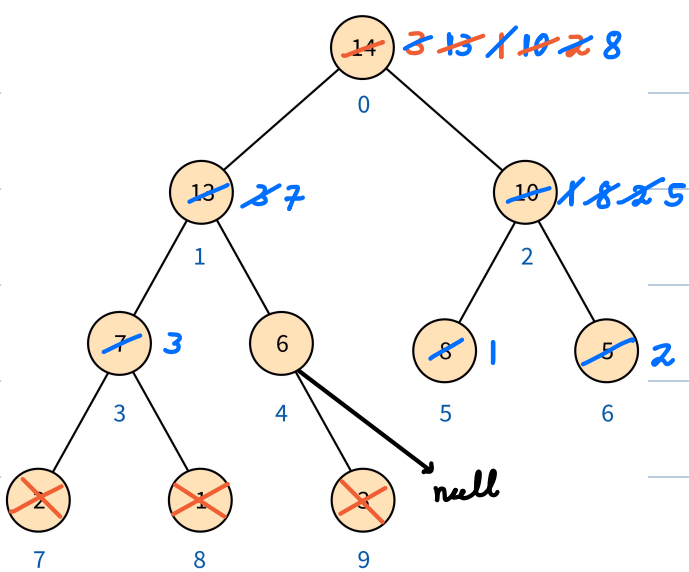
$SC = O(N)$ // Heap

$O(1)$

Approach 2 →

Array → (✓) Heap → (↓) Sorted Array

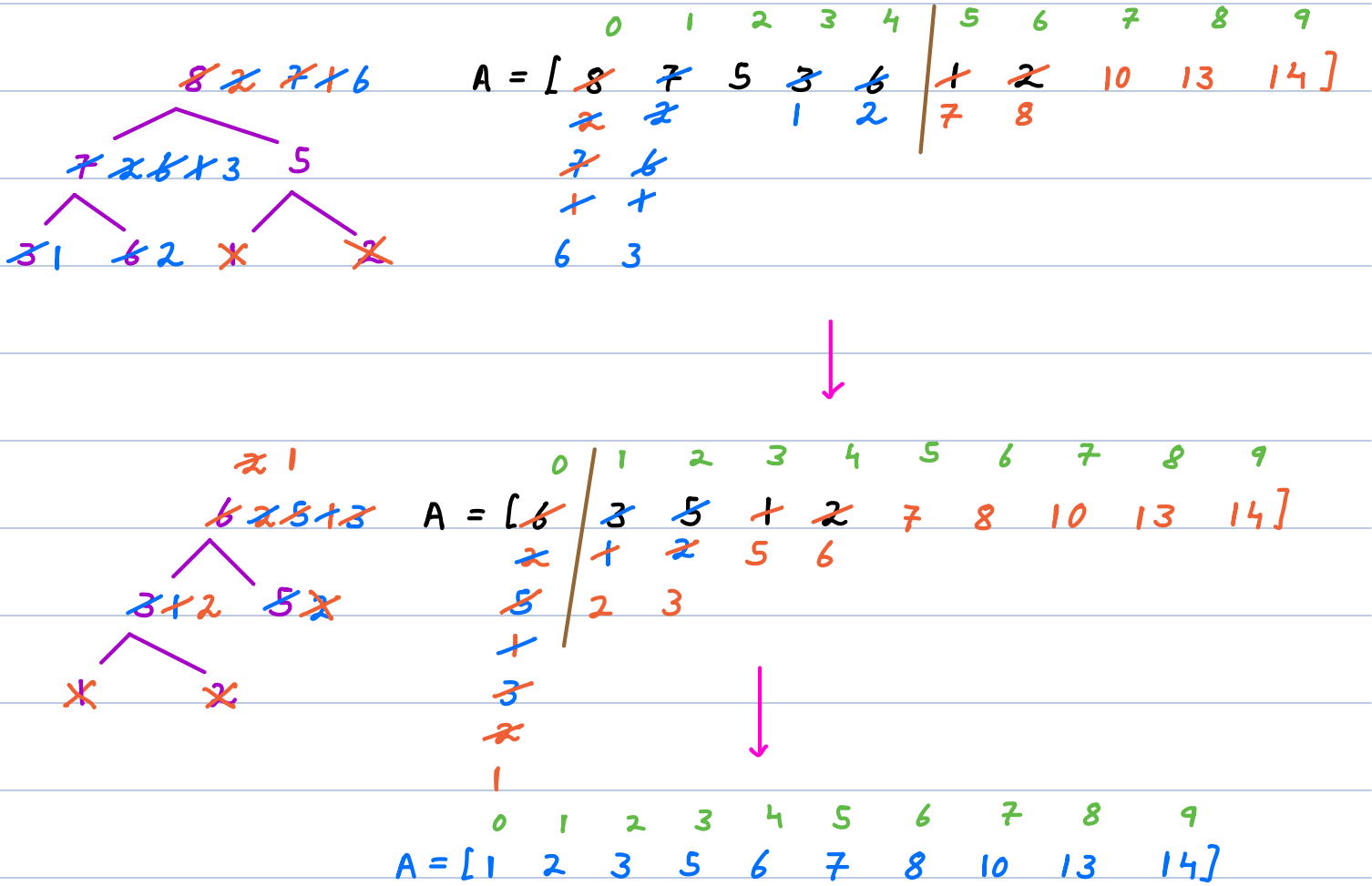
(Max Heap)



max Element

0	1	2	3	4	5	6	7	8	9
A = [14	13	10	7	6	8	5	2	1	3]
3	3	1	3		1	2	10	13	14
13	7	8							
1		2							
10		5							
2									
8									

Extract max & store it at last index.



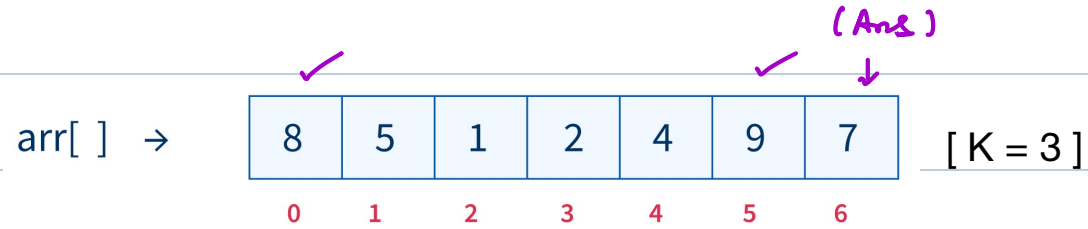
Heap Sort

TC = O (N log (N))

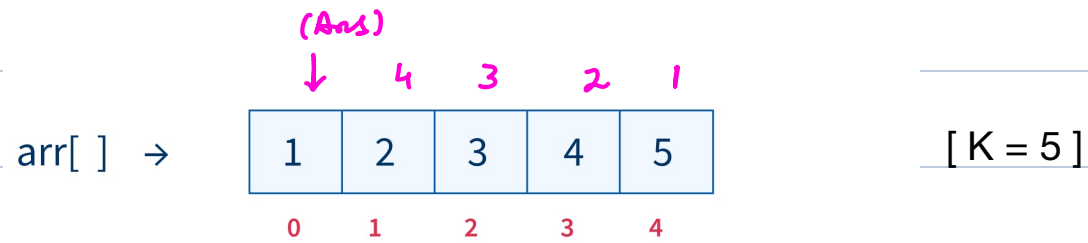
SC = O (1)

Not stable (relative order of equal index).

< Question > : Given arr[N]. Find Kth largest element.



Quiz :



Sol 1 → Sort the array & ans = A[N - K]

TC = O(N log (N))

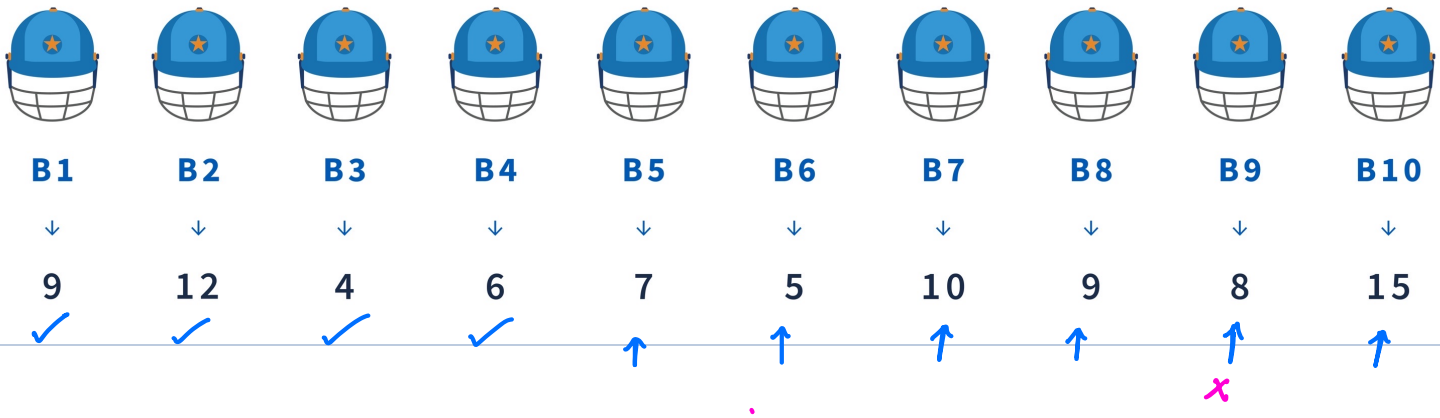
Sol 2 → 1) Build max heap.

2) Extract max K times.

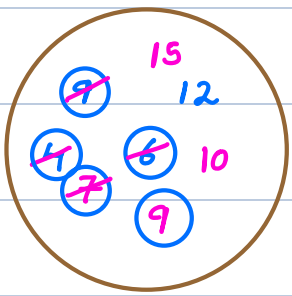
TC = O(N + K log (N)) SC = O(1)

Team Selection

- Select 4 best batsman



Keep track of top 4 players in a min heap.



minHeap size K

```
if (cur > root of min heap) {
    extractMin()
    insert(cur)
} else → Nothing to do
```

Ans = root of minHeap of size K.

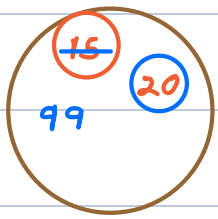
TC = $O(K + (N-K)\log(K))$

SC = $O(K)$

< Question > : Kth largest element for all the windows starting from 0th idx.

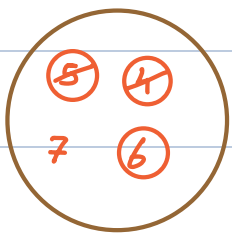
→ min Heap

	0	1	2	3	
A = [	15	20	99	1]	K = 2
ans →	-1	15	20	20	



min Heap

	0	1	2	3	4	
Example → [	5	4	1	6	7],	K = 2
ans →	-1	4	4	5	6	




```
if (cur > root of min heap) {
    extractMin()
    insert(cur)
} else → Nothing to do
ans = root of min Heap
```

$TC = O(K + (N-K)\log(K))$

$SC = O(K)$

Flipkart

Flipkart is currently dealing with the difficulty of precisely estimating and displaying the expected delivery time for orders to a specific pin code.
The existing method relies on historical delivery time data for that pin code, using the median value as the expected delivery time.

As the order history expands with new entries, Flipkart aims to enhance this process by dynamically updating the expected delivery time whenever a new delivery time is added.

The objective is to find the expected delivery time after each new element is incorporated into the list of delivery times.

End Goal

With every addition of new delivery time, requirement is to find the median value.

Why Median ?

The median is calculated because it provides a more robust measure of the expected delivery time.

The median is less sensitive to outliers or extreme values than the mean. In the context of delivery times, this is crucial because occasional delays or unusually fast deliveries (outliers) can skew the mean significantly, leading to inaccurate estimations.

< Question > : Given an infinite stream of integers. Find the median of current set of elements.

↓ middle element

in sorted order

[1 5 (10) 12 15]

[2 8 (10) 20 60 62]

↳ smaller mid

6 → 6

6, (3) → 3

6, 3, 8 → [3 (6) 8] → 6

6, 3, 8, 11 → [3 (6) 8 11] → 6

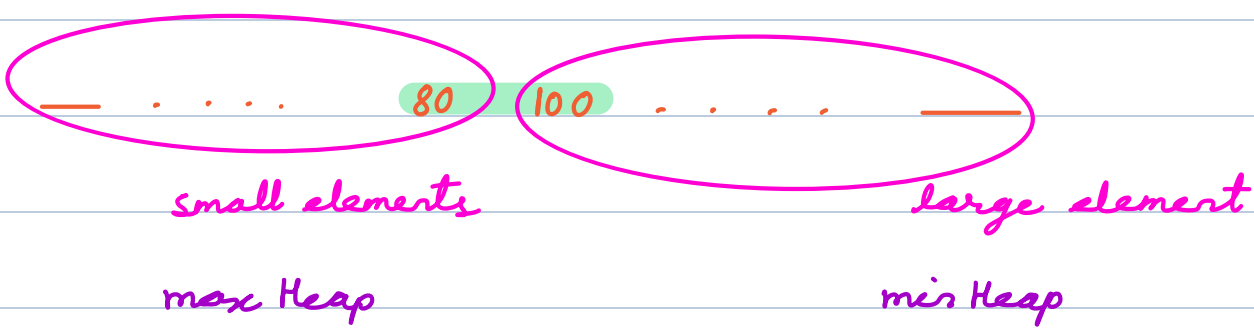
[1 2 4 3] → [1 2 3 4]

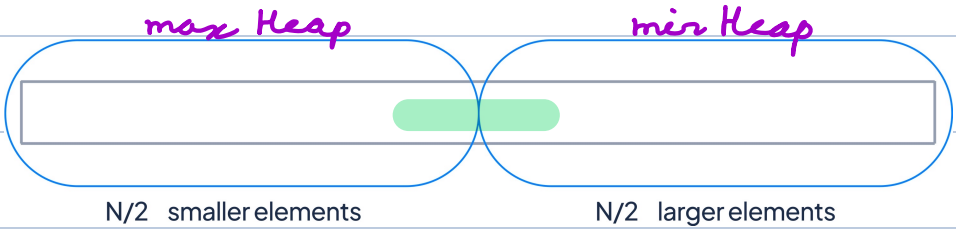
2.5 (Ans)

4

6 3 8 11 → 3 (6) (8) 11

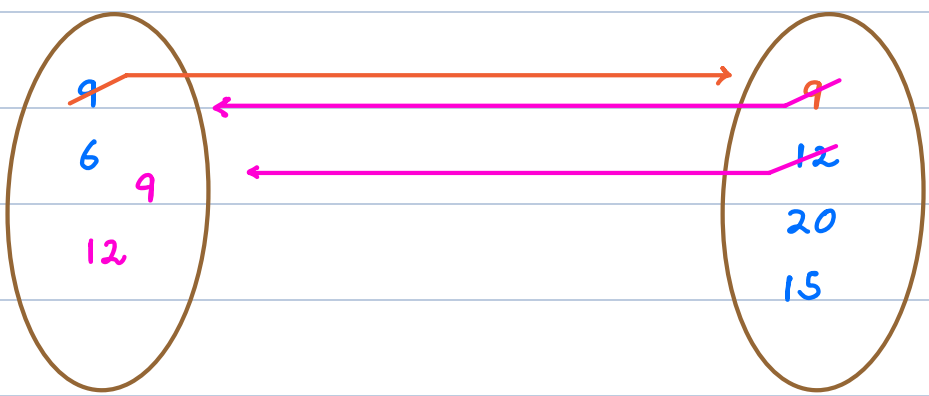
3 4 (6) 8 11





Example : 9 6 12 20 15 . . .

ans → 9 6 9 9 12 . . .



max Heap

min Heap

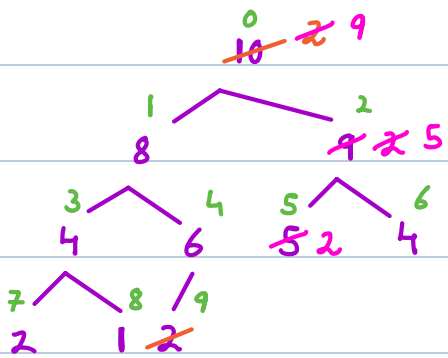
Ans = root of max Heap

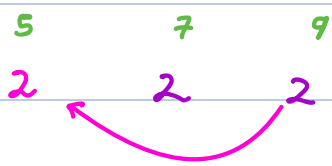
(size of maxHeap - size of minHeap) = {0, 1}

```
Y intake x,
if ( maxHeap.isEmpty() || x <= root of maxHeap ) {
    insertMaxHeap(x)
    if ( maxHeapSize - minHeapSize > 1)
        insertMinHeap ( extractMaxHeap() )
} else {
    insertMinHeap (x)
    if ( maxHeapSize - minHeapSize < 0 )
        insertMaxHeap ( extractMinHeap() )
}
ans = rootMaxHeap
```

TC = O(N log(N))

SC = O(N)





unstable

Agenda:

1. **Meeting Rooms**
2. **Sort the nearly sorted array**
3. **Merge K sorted arrays**
4. **Minimum Distance Equal Pair**
5. **Minimum Window Substring**

Meeting Rooms

You are given an array of meeting time intervals where each interval is represented as [start, end] (start time is less than the end time). Your task is to find the minimum number of conference rooms required to schedule all meetings without overlap.

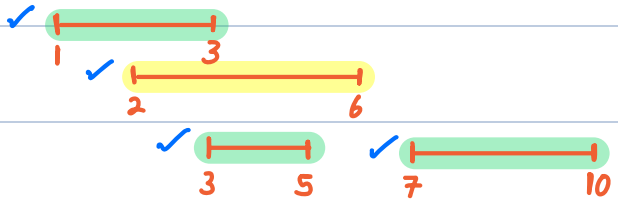
Example 1:

Input: [[0, 30], [5, 10], [15, 20]]
Output: 2

Explanation:

- The first meeting (0, 30) starts at 0 and ends at 30.
- The second meeting (5, 10) overlaps with the first meeting, so a second room is required.
- The third meeting (15, 20) can reuse a room after the second meeting ends, resulting in a total of 2 rooms.

Self try → 10 min



Ans = 2

$0 <= start < end <= 10^6$

0	1	2	3	4	5	6	7	8	9	10	11
0	1	1	-1	0	-1	-1	1	0	0	-1	0

+1

0 1 2 2 2 1 0 1 1 1 0 0 → Ans = 2

```
for i → 0 to (N-1) {
    s = B[i][0]    e = B[i][1]
    rooms[s]++
    rooms[e]--
}

ans = 0
for i → 1 to 10^6 {
    rooms[i] += rooms[i-1]
    ans = max(ans, rooms)
}

return ans
```

TC = O(N + Range) (Range → 10^6) SC = O(Range)

Q → Sort the nearly sorted array i.e every element is at max K distance away from its position in sorted order.

|i − sorted position|

	0	1	2	3	4	5	6	7	8	
sorted →	11	13	20	22	31	45	48	50	60	
A =	13	22	31	45	11	20	48	60	50	K = 4

Sol 1 → Use sorting algo. TC = O(N log(N))

Self try → 10 min 7:55 AM

Sol 2 →

	0	1	2	3	4	5	6	7	8	
A =	13	22	31	45	11	20	48	60	50	K = 4

Smallest element will be between index 0 − K.

min Heap (size K+1)

0	1	2	3	4	5	6	7	8
11	13	20	22	31	45	48	50	60

~~13~~ ~~22~~ ~~31~~

~~45~~ ~~11~~ ~~20~~

~~48~~ ~~60~~ ~~50~~

```
for i → 0 to K {
    insert (A[i])
}

j = 0

for i → (K+1) to (N-1) {
```

```
        A[j] = extractMin()
        j++
        insert(A[i])
    }

while (j < N) {
        A[j] = extractMin()
        j++
}
```

TC = O(N log(K))

SC = O(K)

Q → Merge K-sorted arrays.

A = [2 3 11 15 20]

B = [1 5 7 9]

C = [0 2 4]

D = [3 4 5 6]

Sol 1 → Merge 2 sorted arrays (K-1) times.

Let say len of each array ~ N.

TC → 2N + 3N + 4N + ... + KN

$= N\left(\frac{K(K+1)}{2} - 1\right) \approx O(K^2 N)$

minHeap (size K)

(A[i], A, i)

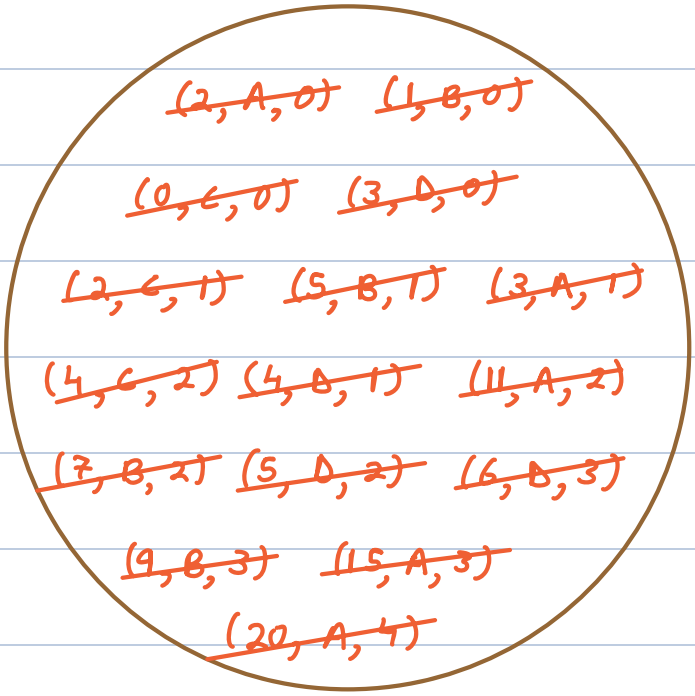
Ans → [0 1 2 2 3 3 4 4 5 5 6 7 9 11 15 20]

SC = O(K)

Let say len of each array ~ N.

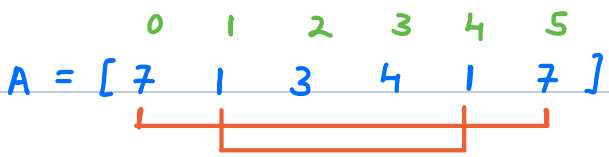
⇒ Total # elements = KN

TC = O(KN log(K))



Minimum Distance Equal Pair

Given an array A, find a pair of indices (i, j) such that A[i] = A[j] and the absolute difference |i - j| is minimised. In simpler terms, you need to find two equal elements in the array that are the closest to each other and return the minimum distance between them.



Ans = 4 - 1 = 3

Self try → 10 min 8:55 AM

Bruteforce → ∀ i, j if A[i] == A[j] find |i-j| & take min.

TC = O(N²) SC = O(1)

0	1	2	3	4	5
7	1	3	4	1	7

A = [7 1 3 4 1 7]

✓ ✓ ✓ ✓ ✓ ✓

HashMap

A[i] → latest index.

ans = ~~∞~~ 4 - 1 = 3

7 → ~~0~~ 5
1 → ~~1~~ 4
3 → 2
4 → 3

ans = ∞

```
for i → 0 to (N-1) {
    if (hm.containsKey(A[i])) {
        ans = min(ans, i - hm.get(A[i]))
    }
    hm.put(A[i], i)
}
if (ans == ∞) return -1
return ans
```

TC = O(N)

SC = O(N)

Minimum Window Substring

Given two strings s and t, find the minimum window in s which contains all characters of t (including duplicates). If no such window exists, return an empty string "".

s = "a d b e c d e b a n c"

t = "a b c b"

Ans = 7

"b e c d e b a"

Bruteforce → ∀ substring of s check if all characters of t are present.

|s| = N $\frac{N(N+1)}{2}$

1) store freq ∀ char in 't' (array / map).

2) store freq ∀ char in substring.

3) ∀ char freq(substring) >= freq(t)

⇒ valid substring to be potential ans.

Sol → Dynamic Sliding Window ⇒ to calculate freq of chars in s.

s = "a d b e c d e b a n c" // N

t = "a b c b" // M

1) freq_t = [1 2 1 0 0 0]
0 1 2 3 4 5 ...

```
for i → 0 to (M-1) {
    freq_t[ t[i] - 'a' ] ++
}
```

2) freq_s using sliding window.

```
ans = ""

l = 0    r = -1          // l — r

while (r < N) {
    if ( check(freq_s, freq_t)) {   // TC = O(26)
        // ∀char freq(s) >= freq(t)
        ans = s.substring(l, r)
        freq_s[s[l] - 'a']--      l++
    } else {
        r++          freq[s[r] - 'a'] ++
    }
} return ans
```

TC = O(N+M) SC = O(2*26) = O(1)

---

t = "abcb" // M

freq-t = [1 2 1 0 0 ...]
0 1 2 3 4

s = " a d b e c d e b a n c " // N
x x x ✓ ✓ ✓ ✓ ✓ ✓
↓ ↓
true

freq-s = [1 1 1 1 2 ...
0 1 2 3 4

for i → 0 to 25 {
    if (freq-s[i] < freq-t[i]) return false
} return true