Coding Blocks

HEAP

PAGE NO. icon
DATE : / /

Priority Queue - Elements will be removed from the queue on the basis of the element's priority.

Heap / Priority Queue

	Sorted Array	Unsorted Array	Heap
Add	O(n)	O(1)	O(logn)
Delete	O(n)	O(n)	O(logn)
Get	O(1)	O(n)	O(1)

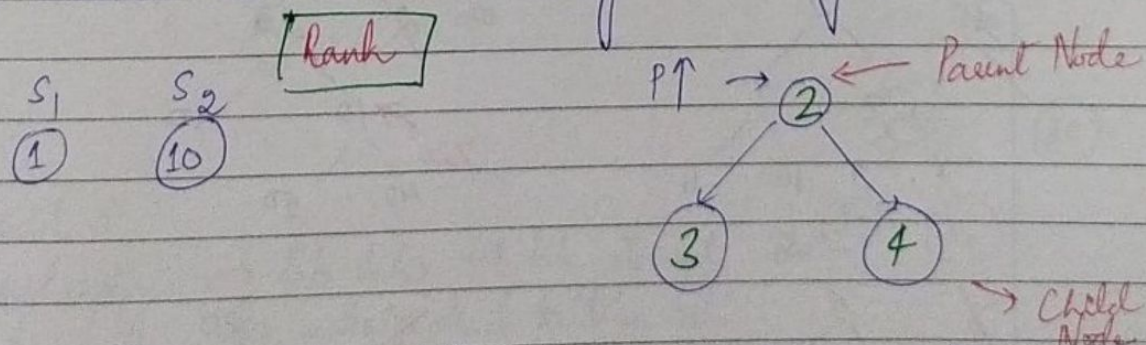
Heap

Heap is a binary tree which follows the following properties :-

1. Priority of parent is always more than children
2. It is a complete binary tree

1 PRIORITY

a) Smaller Value S.V. is having ↑ Priority P.



Rank → Parent Node will have a smaller value → P↑
Child Node will have a greater value → P↓

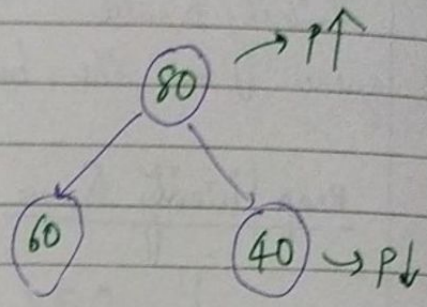
MIN HEAP

PAGE NO. icon
DATE : / /

b) larger value is Priority P↑

Marks

S_1	S_2
80	60

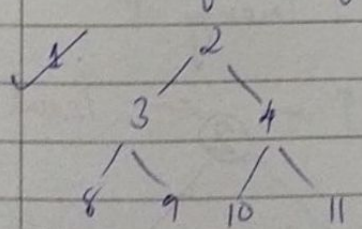


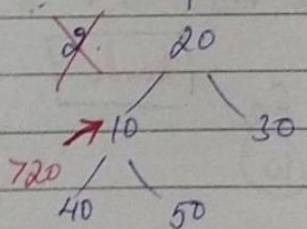
→ Greater marks → larger Priority → ∴ Parent Node will have greater value

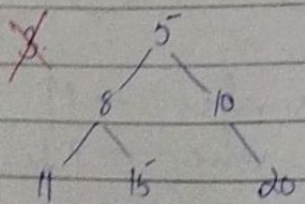
MAX HEAP → Parent value is more

2 COMPLETE BINARY TREE
In which every level except the last level, is completly filled ✓ and the last level is filled from left to right →

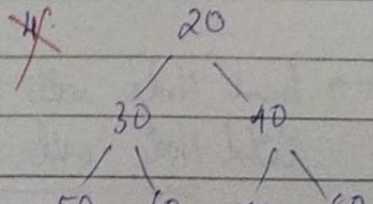
Q Which of the foll is a min heap?



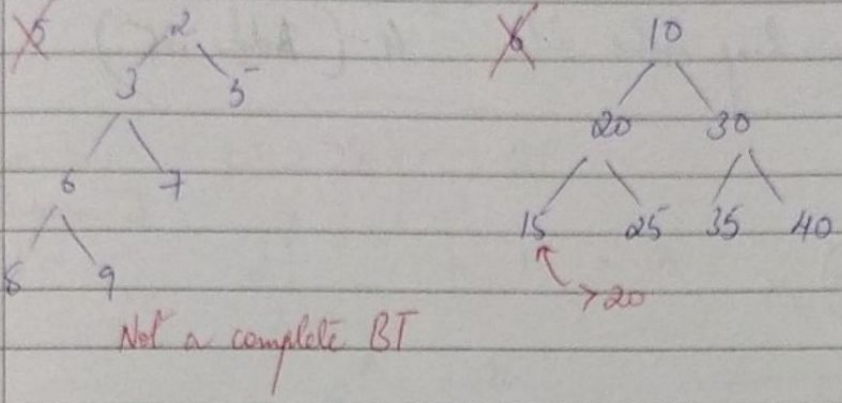




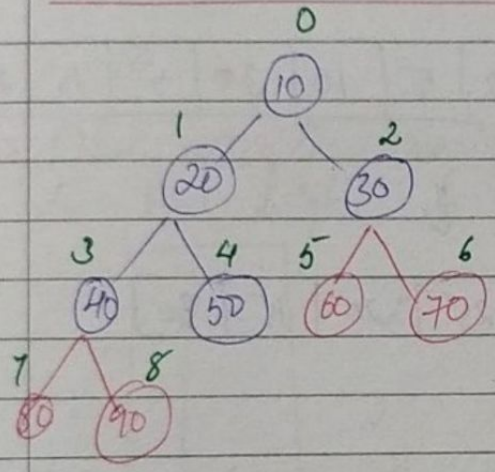
not a complete BT



not a complete BT



ADDITION & DELETION IN HEAP



$p_i = 1$

$l_i = 2p_i + 1 = 3$

$r_i = 2p_i + 2 = 4$

$p_i = 3$

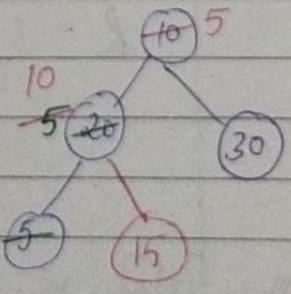
$l_i = 7$

$r_i = 8$

$$l_{ci} = 2p_i + 1 \qquad r_{ci} = 2p_i + 2$$

$$p_i = \frac{l_{ci} - 1}{2} \qquad p_i = \frac{r_{ci} - 2}{2}$$

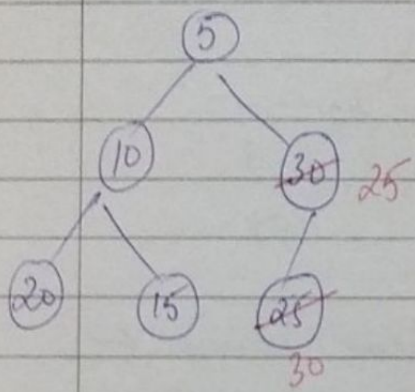
data - | 0: ~~10~~ 5 | 1: ~~20~~ ~~5~~ 10 | 2: 30 | 3: ~~5~~ 20 | 4: 15 |



1. add 5
2. upheapify → tab tak jab tak element ko uska correct position nhi mil jaata
3. add 15

PAGE NO. icon
DATE : / /

Now, the heap looks like – 4. (Add 25)



(25 < 30 : we need to swap)

dataList :- data → | 5 | 10 | 30 | 5 | 15 | 25 |
25 30

⇓

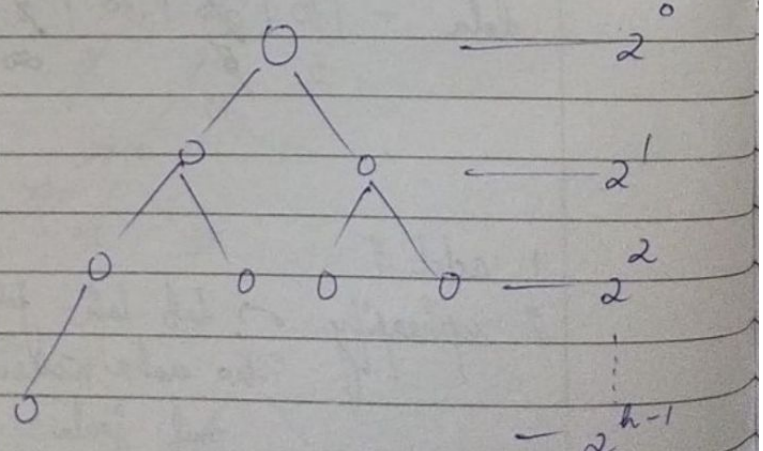
data :- | 5 | 10 | 25 | 20 | 15 | 30 |

(Code → MinHeap (add^n, deletion etc.) see Github)

COMPLEXITY ANALYSIS –

Q1. What are the minimum no. of nodes required in CBT to have height h ?

h = 3



n : Total no. of nodes in BT

PAGE NO. icon
DATE : / /

$$\underbrace{2^0 + 2^1 + 2^2 + \cdots + 2^{h-1}}_{GP} + 1 = n$$

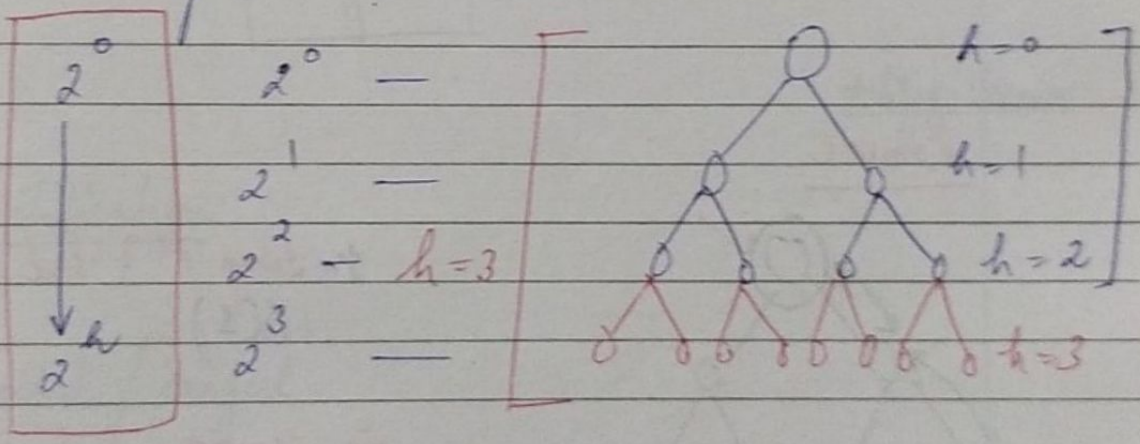
$$\frac{2^{h-1}}{2-1} + 1 = n$$

$$2^h = n$$

$$h = \log_2 n$$

$$\boxed{h = O(\log n)}$$

Q2. What are the maximum no. of nodes required in CBT to have height h?



$$\underbrace{2^0 + 2^1 + 2^2 + \cdots + 2^h}_{GP} = n$$

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

$$2^{h+1} = n+1$$

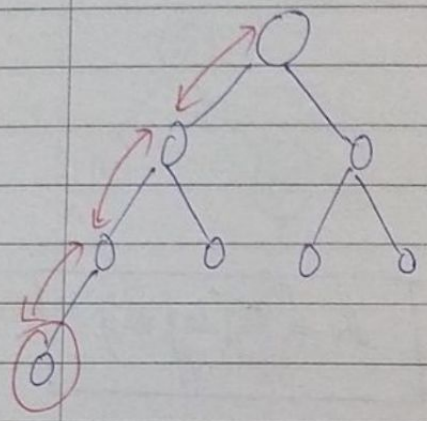
$$\log_2 2^{h+1} = \log_2 (n+1)$$

$$h+1 = \log_2 (n+1) \Rightarrow h = \log_2 (n+1) - 1$$

$$\Rightarrow \boxed{h = O(\log n)}$$

PAGE NO. icon
DATE : / /

Add

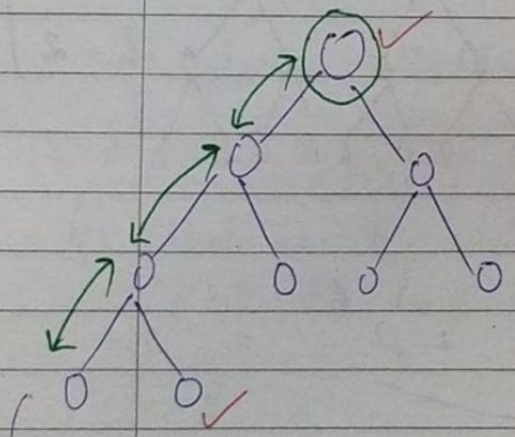


$O(1) + O(h)$

- $O(1)$ → to add the element at the end of the array
- $O(h)$ → upheapify (worst case me swap kete kete we reach the root element)

$O(h) = \boxed{O(\log n)}$

Remove



1. Swap 1st & last element
   $O(1)$

2. remove last me jo element hai (after swapping)

3. In worst case → downheapify kete kete we reach the last

$O(1) + O(h) = O(h)$

- $O(1)$ → swap
- $O(h)$ → downheapify

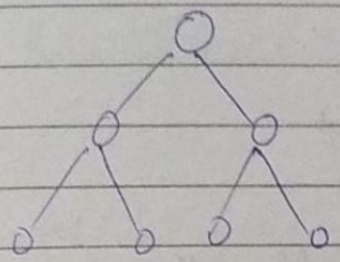
Yahan 3 swaps hue hai, 3 → dependent on h

CBT : $h = O(\log n)$

$\boxed{\text{Remove} : O(\log n)}$

PAGE NO.
DATE : / /

Get



data.get(0)

O(1)

Summarize :-

1. Add : O(log n)
2. Remove : O(log n)
3. Get : O(1)

Aditya Verma

Identification

1. K
2. Smallest / Largest

K + smallest ⇒ Max heap
K + largest ⇒ Min heap

Sorting — n log n (merge sort)

Heap K — n log k (Improved complexity)

PAGE NO. icon
DATE : / /

Q. K^{th} smallest element

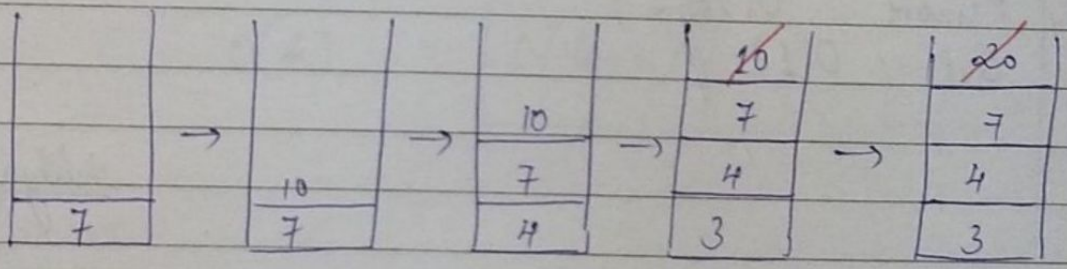
K smallest] → Qualify for heap

↓

Which heap? Max Heap

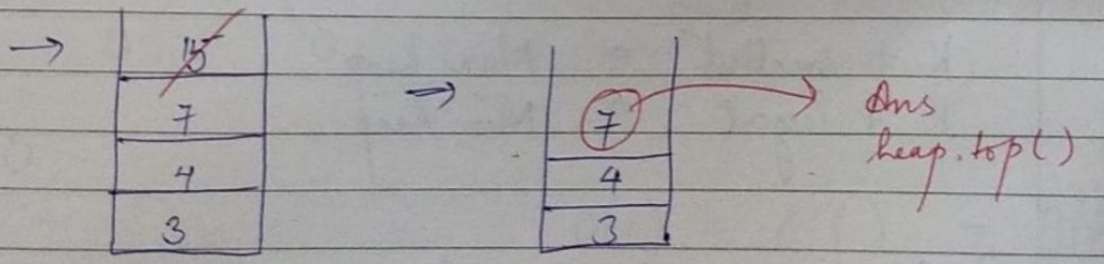
arr[] : 7 10 4 3 20 15

K = 3



size = 1 size = 2 size = 3 size > k size > k

= K : pop : pop



size > k

: pop

NOTE:- Max Heap makes sure the the largest element is always at the top (it does not guarantee that all elements are pushed in sorted order in the heap)

PAGE NO. icon
DATE : / /

Code : using max heap

```
PriorityQueue<Integer> pq = new PriorityQueue<>(Comparator.reverseOrder());

for(int i=0; i<size; i++){
    pq.add(arr[i]);
    if(pq.size() > k){
        pq.pop()/peek;
    }
}
return pq.top()/peek();
}
```

→ to build max heap

Q. Return K largest elements in Array

arr[] : 7 10 4 3 20 15

k = 3

| 20 | 15 | 10 | ⇒ o/p

① Approach 1 : Sorting

| 3 | 4 | 7 | 10 | 15 | 20 |

return

PAGE NO. icon
DATE : / /

But, sorting 3, 4 & 7 is extra work that we're doing

Hence, we need to optimize it

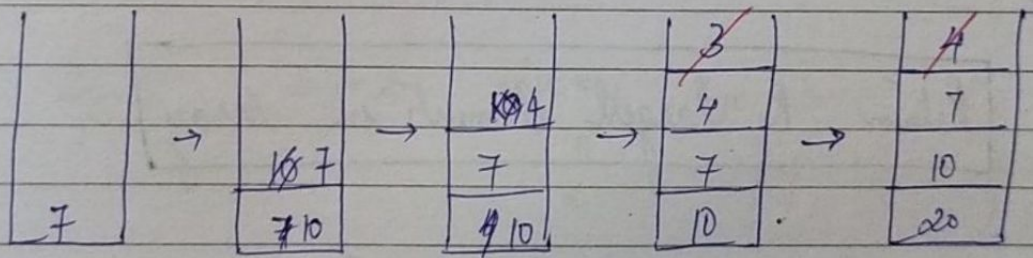
That we can do with help of heap

Identification

1. k
2. Largest

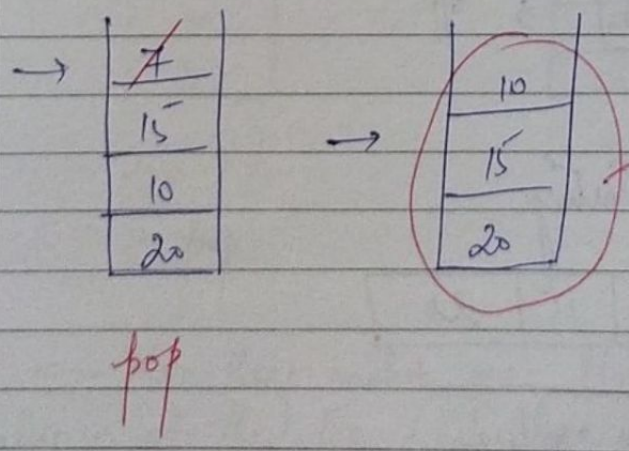
} ⇒ Min Heap

Approach 2 :



size > k ∴ pop

size > k ∴ pop



pop

→ Emptying the Heap

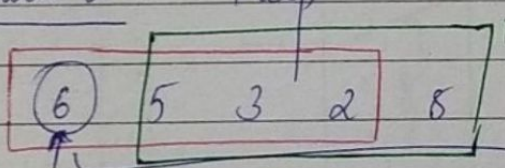
```
while (pq.size() > 0) {
    Sysout (pq.peek());
    pq.pop();
}
```

Q: Sort a K Sorted Array | Sort Nearly Sorted Array

Inp: arr: [6 | 5 | 3 | 2 | 8 | 10 | 9]
k = 3

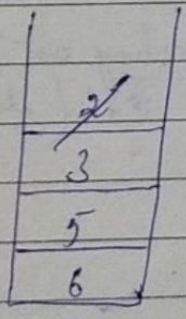
Approach 1: Sorting
Merge Sort O(n log n)
But we didn't use (K)

Approach 2: Heap



ye toh pata hai ki value tak index 0 would be till 6532 we don't need to see aage

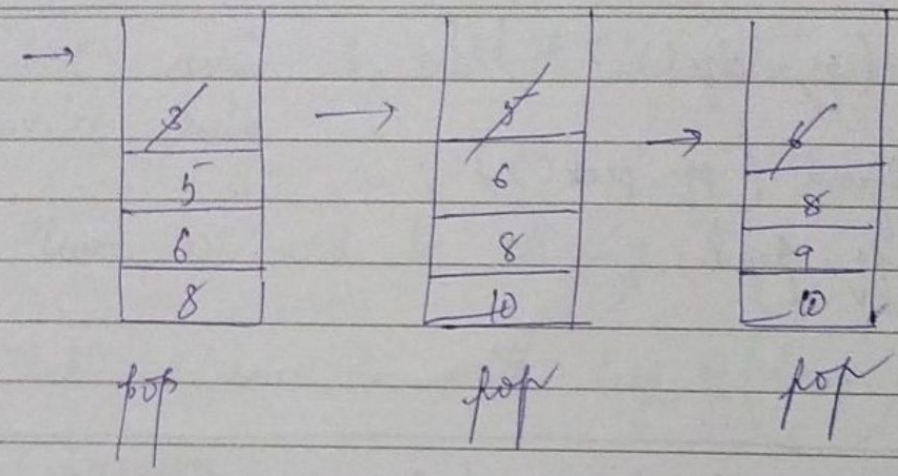
Min Heap

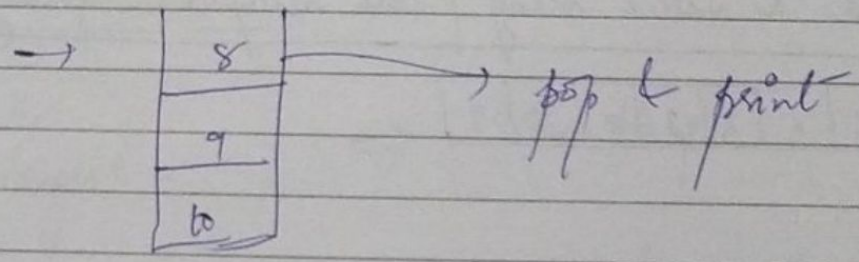


pop

nlogk → kyunki 'k' se upar no. of elements ke liye sort nhi kar rahe

Op: 2 3 5 6 8 9 10

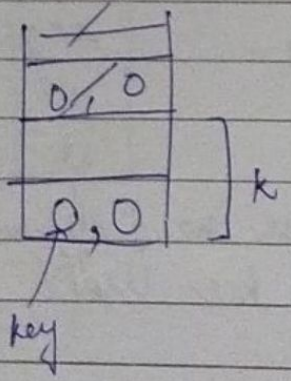




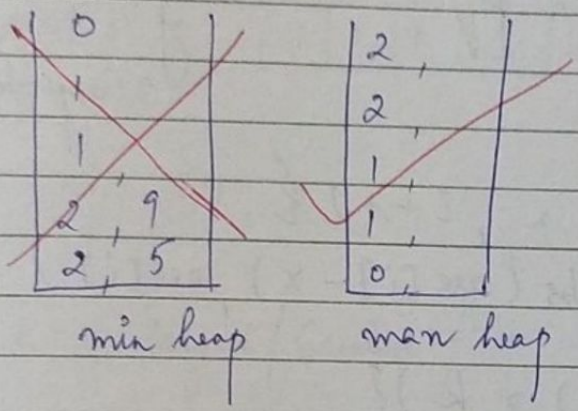
Q. **K Closest Numbers**

arr[] : 5 6 7 8 9
k = 3
x = 7

5 6 7 8 9
7 7 7 7 7
---
diff ⇒ 2 1 0 1 2 → sorting on the basis of that (diffarence)



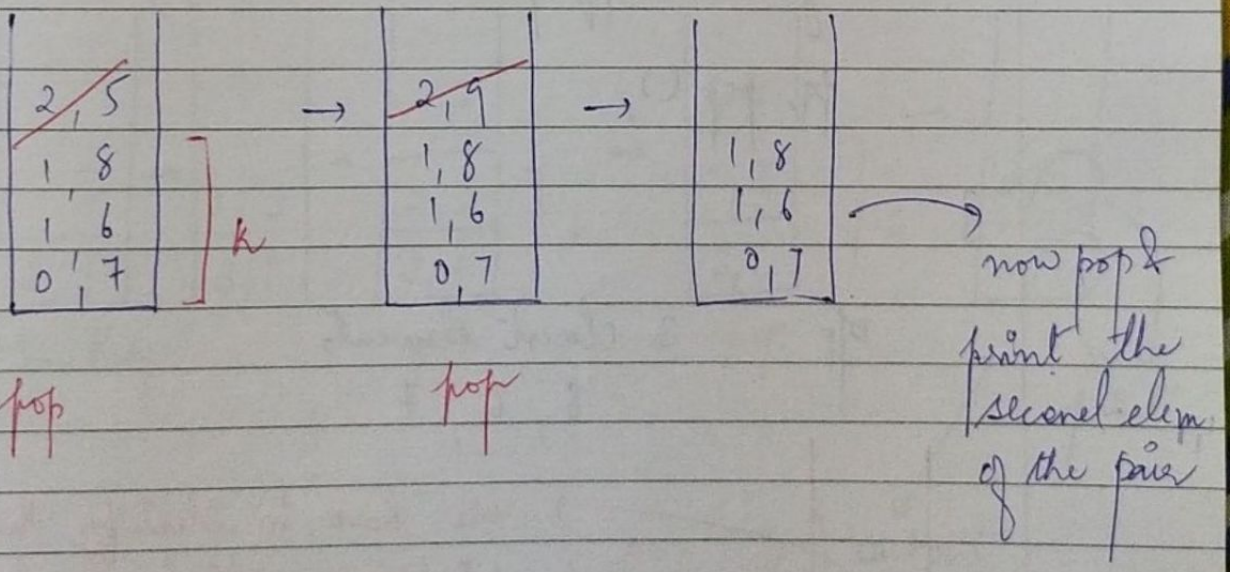
Which heap ??



(Suitable is max heap)

we always pop the useless once → Here useless is 2, 2,
∴ we cannot make max heap.

Pair class ⇒ `abs(arr[i] - x), arr[i]`



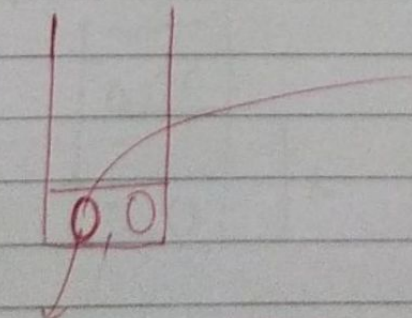
PAGE NO. icon
DATE : / /

Code :

```
PriorityQueue<Pair<Integer>> pq = new PriorityQueue<>(Comparator.reverseOrder());

for (int i=0; i<n; i++) {
    pq.add(abs(arr[i]-x), arr[i])
    if (pq.size() > k){
        pq.pop();
    }
}
while(pq.size() > 0) {
    sysout(pq.peek().second);
    pq.pop();
}
```

o/p : 3 closest elements
8, 6, 7



→ we have to identify this key ki kispe sort karna hai

key → # itself
→ frequency
→ difference

PAGE NO. icon
DATE : / /

Q. **Top K frequent numbers**

arr[] = | 1 | 1 | 1 | 3 | 2 | 2 | 4 |

k = 2

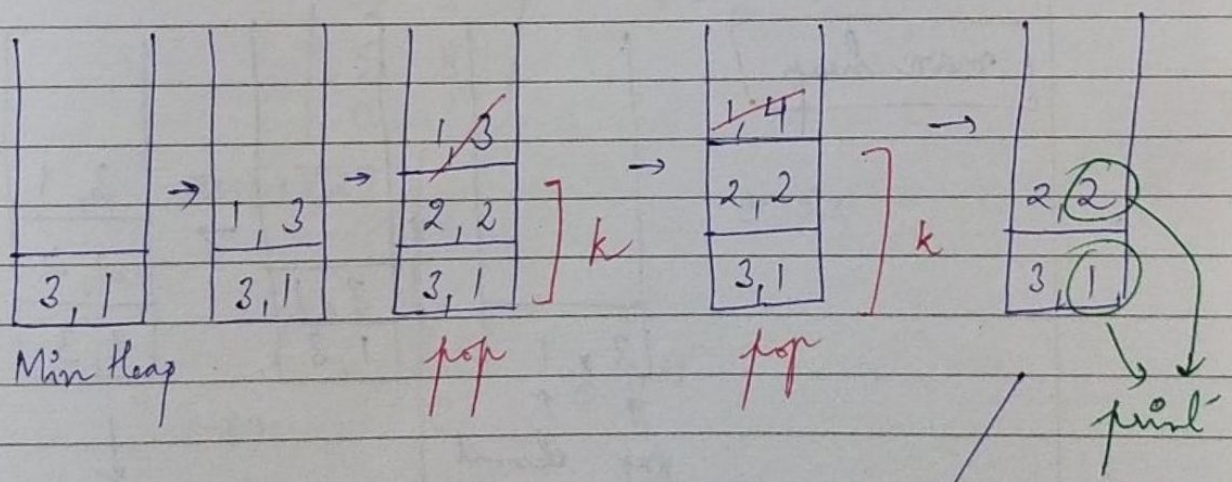
o/p : 1, 2

feel karo

k + Largest / Greatest / Top → **Min Heap**

HashMap

1	3
3	1
2	2
4	1
↑ arr[i]	↑ freq



(For the syntax of priority queue → check github repo) Output

PAGE NO. icon
DATE : / /

Q. Frequency Sort

arr[] : | 1 | 1 | 1 | 3 | 2 | 2 | 4 |

op : | 1 | 1 | 1 | 2 | 2 | 3 | 4 |

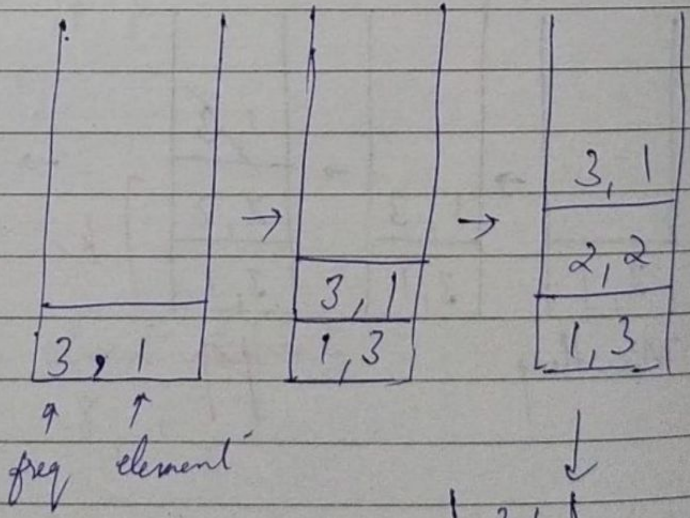
3 2 1 1

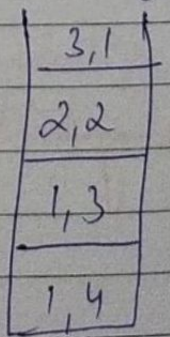
Sort based on frequency.

Map

key	value
1	3
3	1
2	2
4	1

max heap



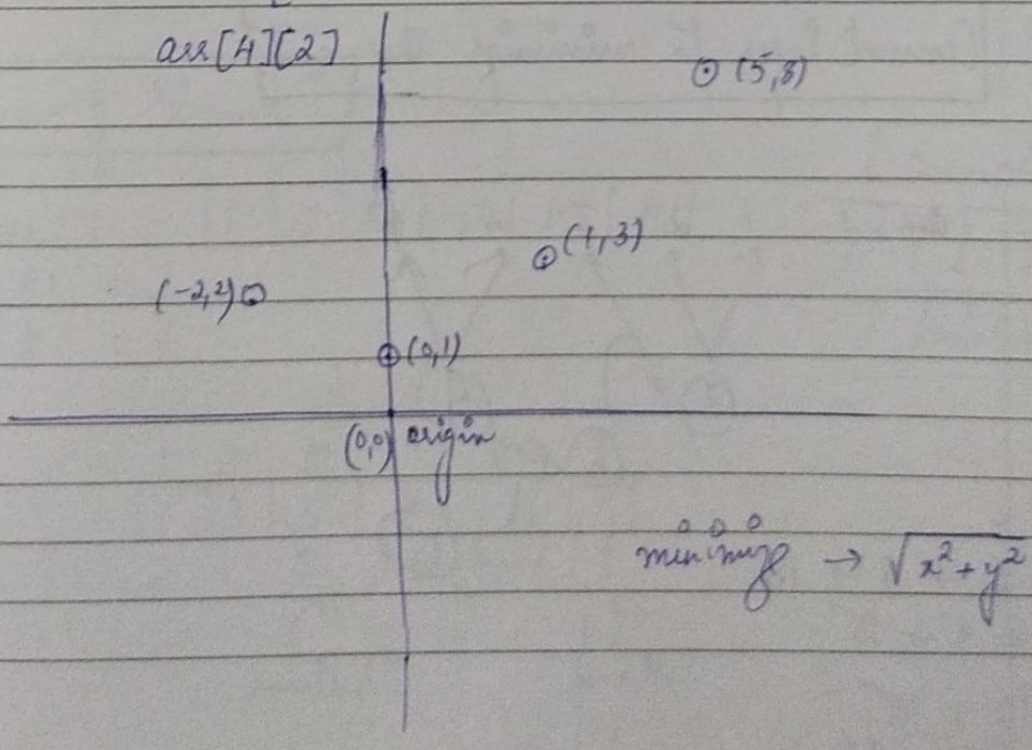


```
while (pq.size() > 0) {
    int freq = pq.top().first;
    int elem = pq.top().second;
    for (int i = 0; i <= freq; i++) {
        sysout (elem);
    }
    pq.pop();
}
```

Q. **K - Closest Points to Origin**

$$arr[] = \begin{bmatrix} 1 & 3 \\ -2 & 2 \\ 5 & 8 \\ 0 & 1 \end{bmatrix} \qquad K = 2$$

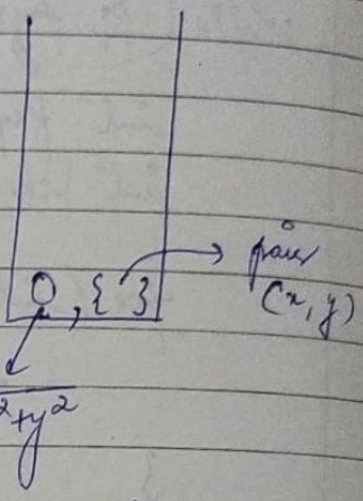
arr[4][2]



minimize → $\sqrt{x^2+y^2}$

k = 2

2 closest points from origin



pair < int < pair < int, int >> p ;

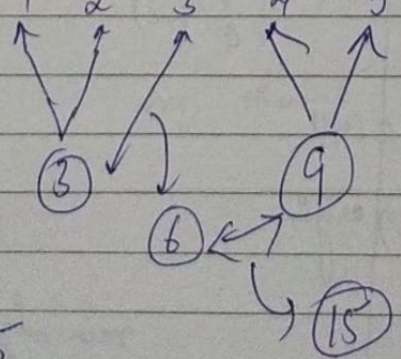
To ease out, we'll sort acc to $(\sqrt{x^2+y^2})^2 = x^2+y^2$

Max Heap Bana lenge

(jab samajh na aaye ki konsa heap banana hai toh make both & check konsa suitable hai & then implement that heap)

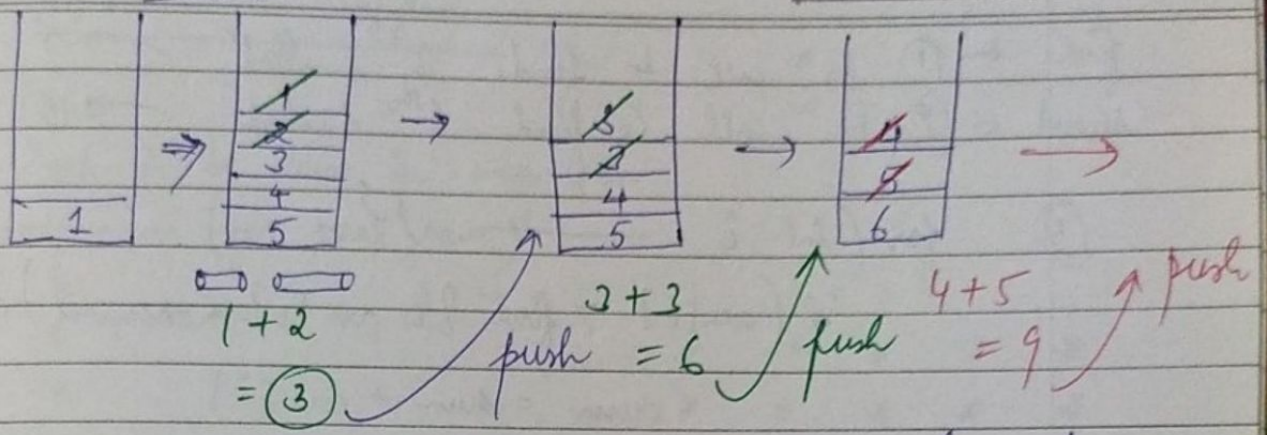
Q. Connect Ropes to minimize the cost

arr[] : 1 2 3 4 5

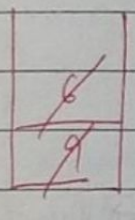


Cost =
3 + 6 + 9 + 15
9 + 9 + 15
18 + 15
= 33

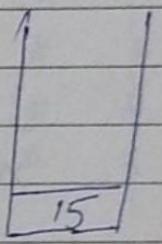
Min Heap



Sum Tcost
= 3+6
+9+15
= 33



pop ⇒ 6+9 =15 push



>=2 nhi hai

pop krke & add to Tcost

Return Tcost

Ans

Q. **Sum of Elements**

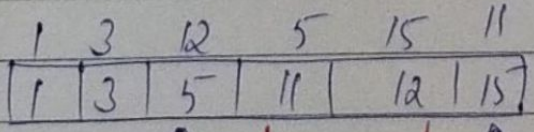
arr[] : | 1 | 3 | 12 | 5 | 15 | 11 |

k1 = 3

k2 = 6

concept

Variation



3rd smallest

6th smallest

11 + 12 = 23 Ans

PAGE NO. icon
DATE : / /

first ← ① Fxn call to find 3rd smallest → 5
second ← ② Fxn call to find 6th Smallest → 15

③ for(int i ———){
    if(arr[i] > first && arr[i] < second){
        sum = sum + arr[i];
    }
}
return sum;

Variations ⇒

k_1 → Largest	k_1 → Smallest
k_2 → Largest	k_2 → Largest

k_1 → Largest	k_1 → Smallest
k_2 → Smallest	k_2 → Smallest

Ase variations ban sakte hain ques ke

But CONCEPT wahi rahega !!!

ALL CODES : GITHUB.