Time Complexity : $O(N \log N)$

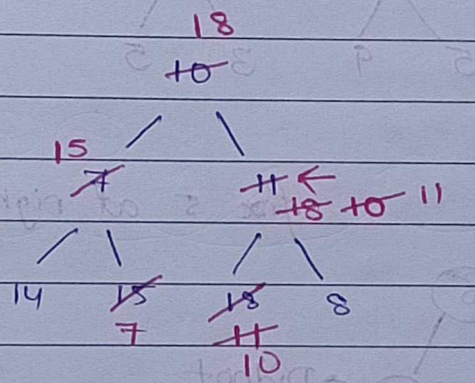
Lecture - 50

(Heap)

Heap sort : (sorting)

Max Heap
Min Heap

Heapify : Correct position
O(N) create.



Max Heap

for sorting :

1) Take Element from top.

2) put the last element in place of top.

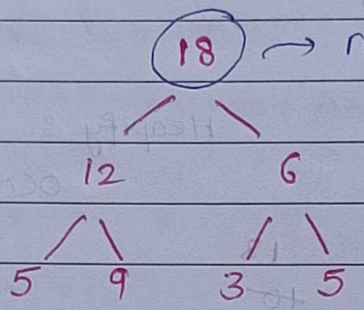
3) top element find its correct position and fix it.

4) Apply these 3 steps repeatedly till Heap is empty.

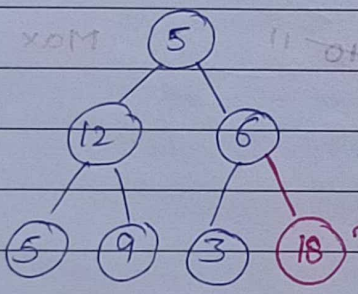
0	1	2	3	4	5	6
18	12	6	5	9	3	5

Sort these

Make a tree :



→ remove and put here bottom right most element.



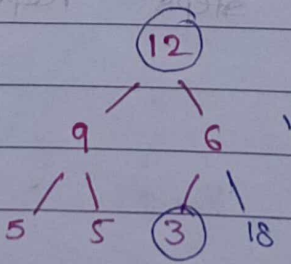
fix 5 at right place.

→ Didnot Consider this

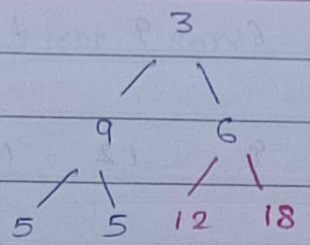
5	12	6	5	9	3	18

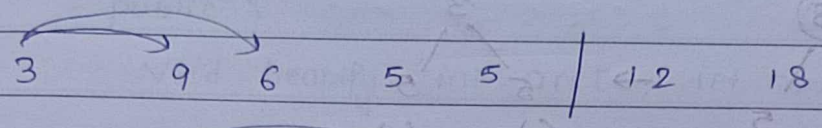
12	5	6	5	9	3	18

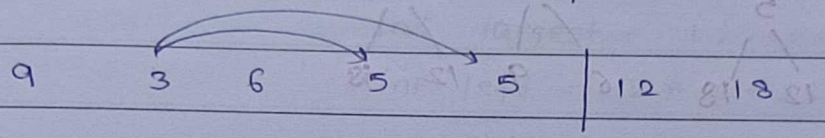
12	9	6	5	5	3	18

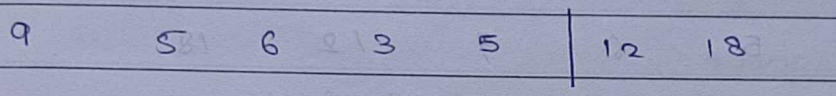


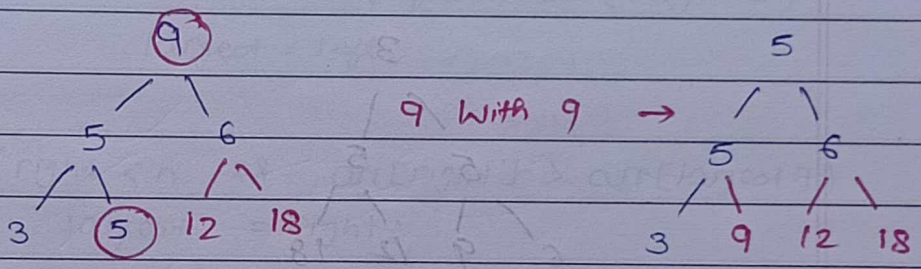
12 With 3

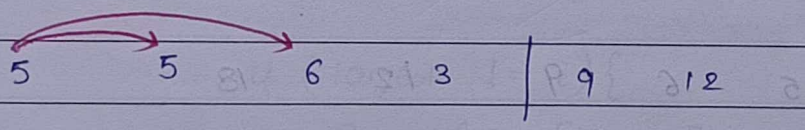

3
9 6
5 5 12 18

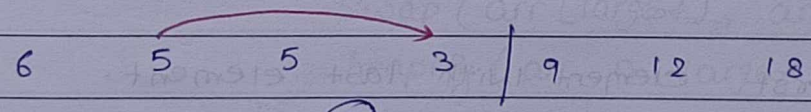

3 9 6 5 5 | 12 18

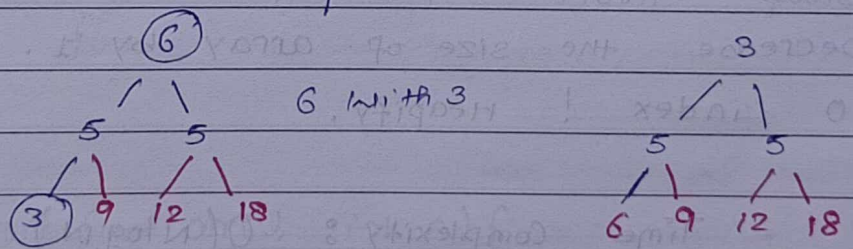

9 3 6 5 5 | 12 18

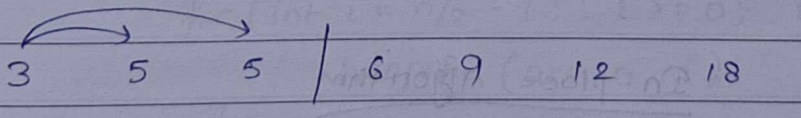

9 5 6 3 5 | 12 18

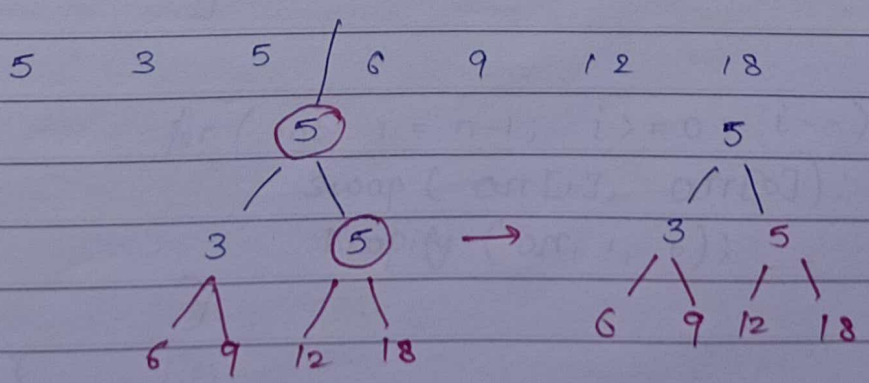

9
5 6
3 5 12 18
9 with 9 →
5
5 6
3 9 12 18


5 5 6 3 | 9 12 18


6 5 5 3 | 9 12 18

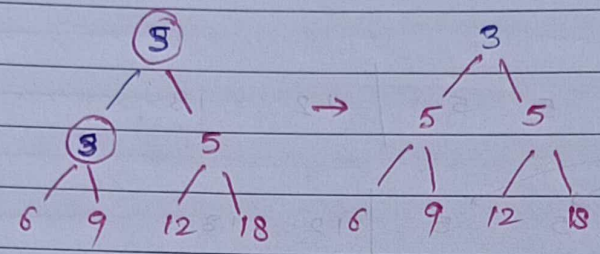

6
5 5
3 9 12 18
6 with 3
3
5 5
6 9 12 18


3 5 5 | 6 9 12 18

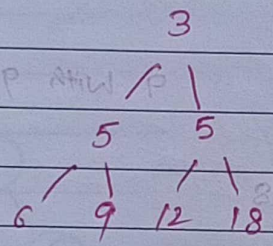

5 3 5 | 6 9 12 18
5
3 5
6 9 12 18
→
5
3 5
6 9 12 18

5 3 | 5 6 9 12 18

5 3 | 5 6 9 12 18



3 | 5 5 6 9 12 18



3 5 5 6 9 12 18

① Swap first element with last element.
② Decrease the size of array by 1.
③ 0 index ! Heapify.

Time Complexity : $O(N \log N)$

In-place Algorithm

Problem - 1 : Heap sort

Code :

```
class solution {
    public :
        Void heapify (int arr[], int n, int i) {
            int largest = i;
            int left = 2 * i + 1;
            int right = 2 * i + 2;

            if (left < n && arr[left] > arr[largest])
                largest = left;

            if (right < n && arr[right] > arr[largest])
                largest = right;

            if (largest != i){
                Swap (arr[largest], arr[i]);
                heapify (arr, n, largest);
            }
    }

Void heapsort (int arr[], int n){
        for (int i = n/2 - 1; i >= 0; i--){
            heapify (arr, n, i);
        }

        for (int i = n-1; i >= 0; i--){
            swap (arr[i], arr[0]);
            heapify (arr, i, 0);
        }
}
```

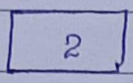
~~#Heap~~

Heap (Code using STL)

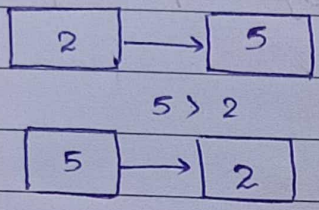
```
#include <iostream>
#include <queue>        // STL for Heap
Using namespace std;

int main(){
    // max heap
    priority_queue<int> p;
    p.push(4);
    p.push(7);
    p.push(2);
    p.push(6);

    Cout<<p.top();
    P.pop();
    Cout<<p.top();

    Cout<<p.empty();    // 0 because heap is not empty
    // min Heap
return 0;
    priority_queue<int, vector<int>, greater<int>> p;
}
```

Heap creation by linked list :

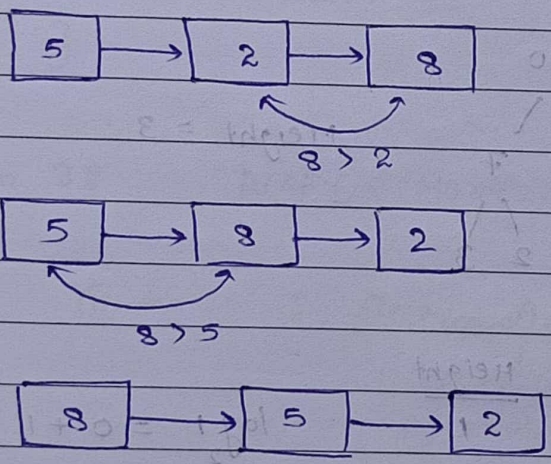
→ 2



→ 5



→ 8



min Heap :

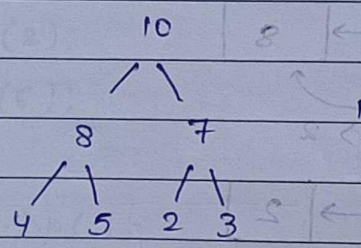
```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

int main (){
    // min Heap
    priority_queue <int, vector <int>, greater <int> > p;
        p.push (9);
        p.push (6);
        p.push(3);
        p.push(2);
```

```
cout<<p.top();
p.pop();
cout<<p.top();
return 0;
}
```

* problem 2 : ~~Height of a tree~~ (Height of Heap)

Tree in form of array

arr : 10 8 7 4 5 2 3



Element	Height	
1	1	$\log_2 1 = 0 + 1 = 1$
2	2	$\log_2 2 = 1 + 1 = 2$
3	2	$\log_2 3 = 1 + 1 = 2$
(4 - 7)	3	$\log_2 4 = 2 + 1 = 3$
(8 - 15)	4	$\log_2 9 = 3 + 1 = 4$

find log with base 2 and then add 1.

$\log_2 15$: $\frac{15}{2} \rightarrow \frac{7}{2} \rightarrow \frac{3}{2} \rightarrow 1$

$3 + 1 = 4$

On question, they ask 0 based height, so the answer is only $\log_2 n$.

Code:

```
int heapHeight ( int N, int arr[]) {
        int Count = 0;
        While ( N != 0) {
            N = N/2;
            Count++;
        }
        Return Count -1;
}
```

* problem 3: profix Maximization (interview bit)

Given : Array A
↓
represents seats in each row of stadium.

sell Ticicet to B people.

Seat Cost = No. of vacant seats in row it belong to

Maximize the profit.

A = {2, 3}
0 1

B = 3

Row 0 __ __ cost = 2

Row 1 B1 __ __ (Cost = 3) Max

(B = 3) → people.

people --

B = 2

2 Row 0 _ _ price = 2

1 Row 1 _ _ price = 2 Max

B = ~~2~~ 1

profit = 3 + 2 = 5

Row 0 = 2 → Max

Row 1 = 1

B = ~~1~~ 0

profit = 3 + 2 + 2 = 7.

* Example 2:

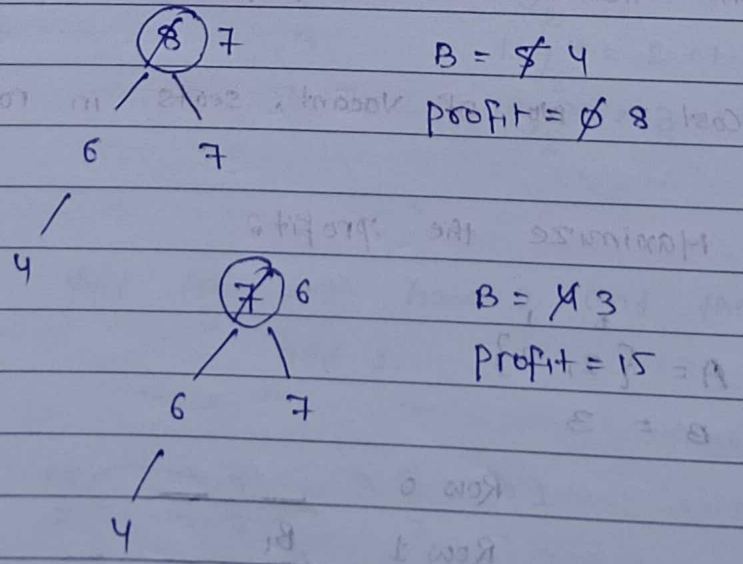
Array : [6, 8, 4, 7]

B = ~~8~~ 4

find Max

profit = 8

Solve by Max-heap :

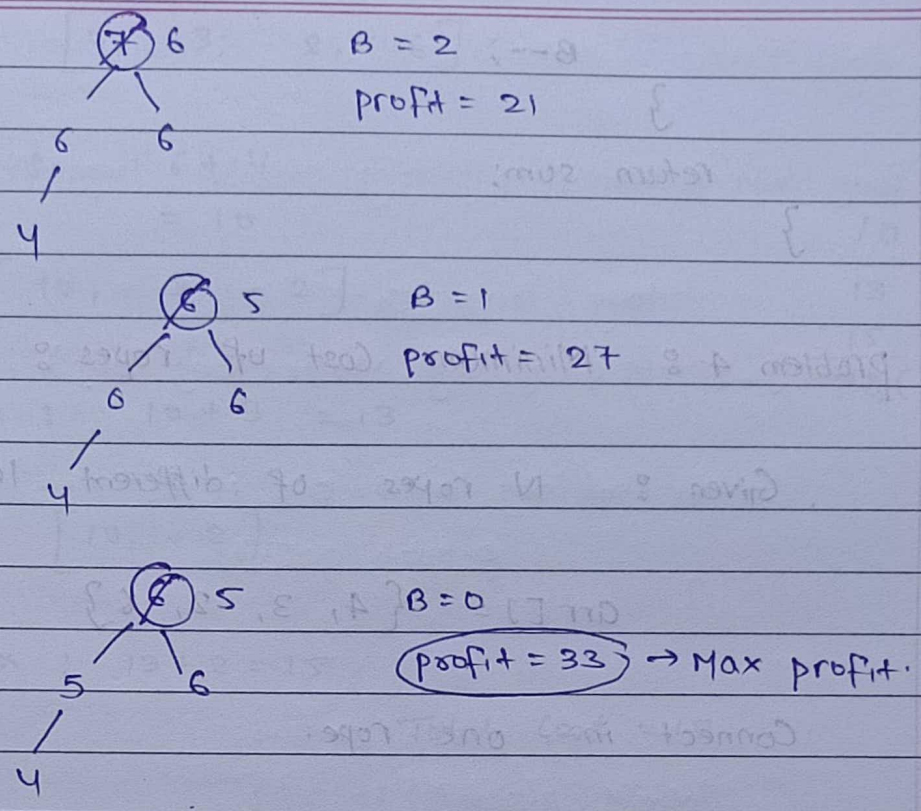


B = ~~8~~ 4

profit = ~~8~~ 8

B = ~~7~~ 3

profit = 15



Time Complexity : O(NlogN).

Code :

```
int solution::solve(vector<int> &A, int B){
    if(B==0)
        return 0;

    int sum = 0;
    priority_queue<int> p;
    for(int i=0; i<A.size(); i++){
        if(A[i])
        p.push(A[i]);
    }

    while(B!=0 && p.size()!=0){
        sum += p.top();
        if(p.top()>1)
        p.push(p.top()-1);
        p.pop();
```

```
        B--;
    }
    return sum;
}
```

* **problem 4 : Minimum Cost of ropes :**

Given : N ropes of different length

arr[] = {4, 3, 2, 6}

Connect in one rope.

Cost of Connect = sum of both ropes

find minimum Cost of Connect all ropes.

Example

[4, 3, 2, 6]

Choose Minimum :

3 & 2

5

[4, 5, 6]

Minimum :

4 + 5 = 9

[9, 6]

Minimum

9 + 6 = 15.

5
9
15
29 (Min)

Total Cost = 15

[4, 3, 2, 6]

Max : 6 + 4
= 10

[10, 3, 2]

Max : 10 + 3 = 13

[13, 2]

Max : 13 + 2 = 15

Total Cost = 15

10
13
15
38 (Max)

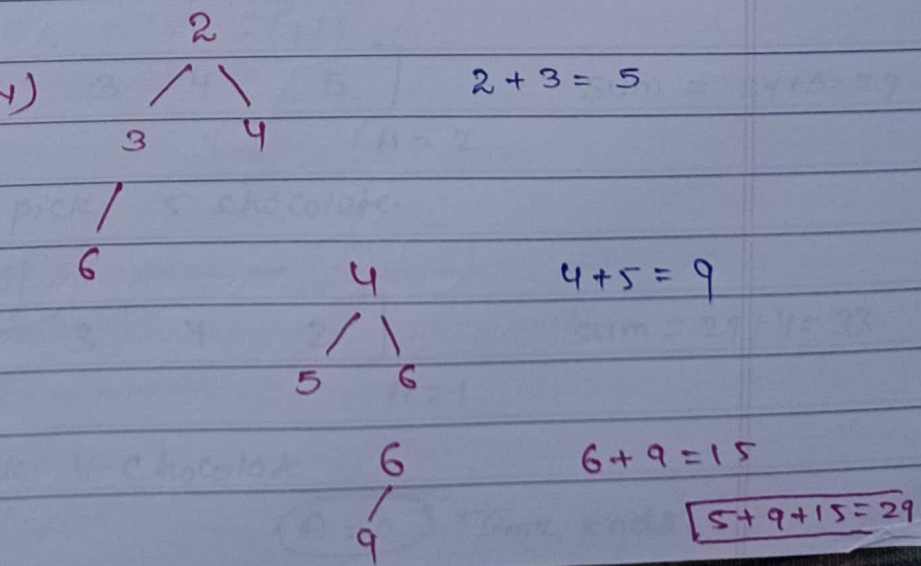
By finding from Minimum we get the Correct answer.

Every time we find 2 smallest size rope.

By array Every time it takes O(N)
Total time : $O(N^2)$

So, We use Min Heap.
Then we solve easily.

Time Complexity : O(NlogN)



Code :

```
long long minCost ( long long arr[], long long n ) {

        long long cost = 0;
priority_queue <long long, vector <long long>, greater <long long>> p;

    for (int long long i=0; i<n; i++) {
            p.push (arr[i]);
    }

    long long first, second;
    While ( p.size ( ) != 1 ) {
            first = p.top ( );
            p.pop ( );
            second = p.top ( )
            p.pop ( );
            first += second;
            Cost += first;
            p.push ( first );
        }
    return cost;
}
```

problem 5 : Magician and chocolate (interview BIT)

Given N Bags, Each Bags Contains some chocolate.

On one unit of time, kid choose a random Bag i, eats Bi chocolates, then magicians fills the bags with Bi/2 chocolates.

Find (maximum) number of chocolates that kid can eat in A unit of time.

May be Concept of Heap.

A = 5 (time)

[2, 4, 6, 8, 10] (Maximum chocolate)

sum = 10

Boy picks 10 chocalate

[2 4 6 8 5] sum = 10+8 = 18

A = 4

Boy pick bag of 8 chocolate

[2 4 6 4 5]

A = 3 sum = 18+6 = 24

He pick bag of 6 chocolate

[2 4 3 4 5] sum = 24+5 = 29

A = 2

He pick 5 chocolate.

[2 4 3 4 2] sum = 29+4 = 33

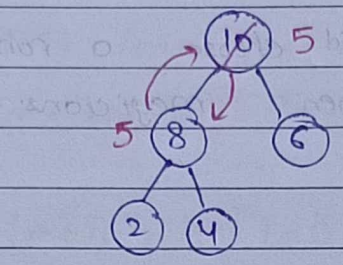
A = 1

He pick 4 chocolate

(A = 0) Time ends.

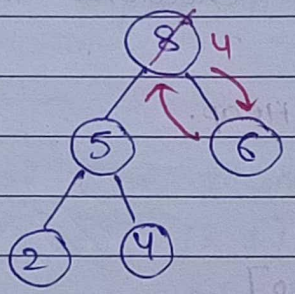
Boy picks maximum of 33 chocolate

we can solve this question by Max Heap.



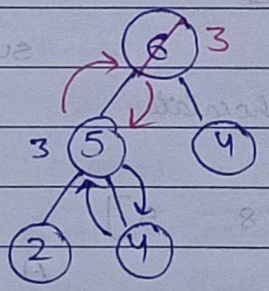
A = ~~4~~ 4

sum = ~~0~~ 10



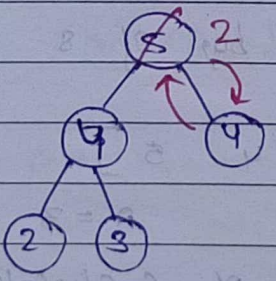
A = ~~4~~ 3

sum = ~~10~~ 18



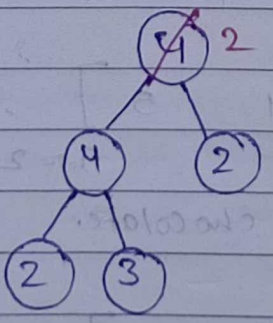
A = ~~3~~ 2

sum = ~~18~~ 24



A = ~~2~~ 1

sum = ~~24~~ 29



A = ~~1~~ 0

sum = ~~29~~ 33

final sum = 33

Code :

```
int solution :: nchoc (int A, vector <int> &B) {
    priority_queue <int> p;
    long long int total = 0;

    for (int i = 0; i < B.size(); i++)
        p.push(B[i]);

    while (A != 0 && p.size() != 0) {
        total = (total + p.top()) % 1000000007;
        if (p.top() / 2)
            p.push(p.top() / 2);
        p.pop();
        A--;
    }
    return total;
}
```

problem 6 : k largest Element :

* Arr[] = { 10, 20, 15, 18, 25, 17 }

① Sort → N log N

② k element को bahar Nikal do

Total time = K + N log N.
= N log N

*

Max Heap : creation : N log N => N

Then k element को निकाल दो
↳ K log N.

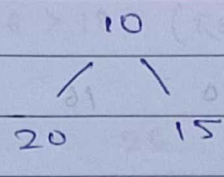
N + k log N.

* k - largest Element
↳ then create Min Heap

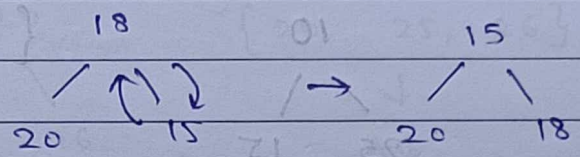
k - smallest Element
↳ then create Max Heap.

For this problem :
Create MIN Heap
of size k.
↳ k log k.

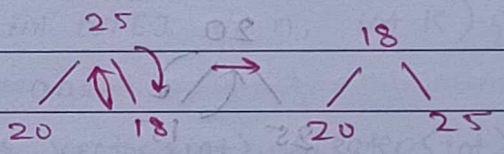
Min heap



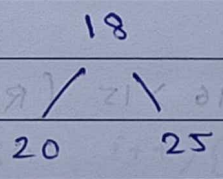
Now, 18 → 18 > 10



Now, 25 → 25 > 15



Now 17 → 17 < 18



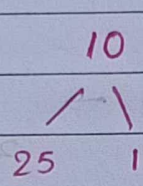
Largest Element : 18, 20, 25

Example 2 :

10 25 15 20 16 18 26

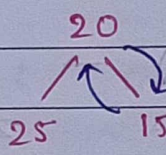
k = 3

Mean Heap of 3 elements

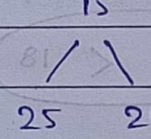


{10, 25, 15}

20 → 20 > 10 (Remove 10)

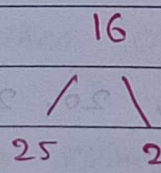


{20, 25, 15}



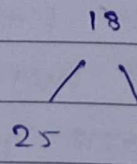
{15, 25, 20}

16 → 16 > 15 (Remove 15)



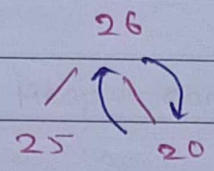
{16, 25, 20}

18 → 18 > 16 (Remove 16)

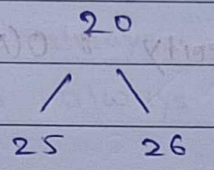


{18, 25, 20}

26 → 26 > 18 (Remove 18)



{25, 25, 20}



{20, 25, 26}

↓

Ans

Code :

```
vector<int> kLargest(int arr[], int n, int K){
        vector<int> ans;
    priority_queue<int, vector<int>, greater<int>> p;

    for(int i=0; i<k; i++){
            p.push(arr[i]);
        }

    for(int i=k; i<n; i++){
        if(p.top() < arr[i]){
                p.pop();
            p.push(arr[i]);
        }
    }
    while(!p.empty()){
        ans.push_back(p.top());
        p.pop();
    }
    reverse(ans.begin(), ans.end());
    return ans;
}
```