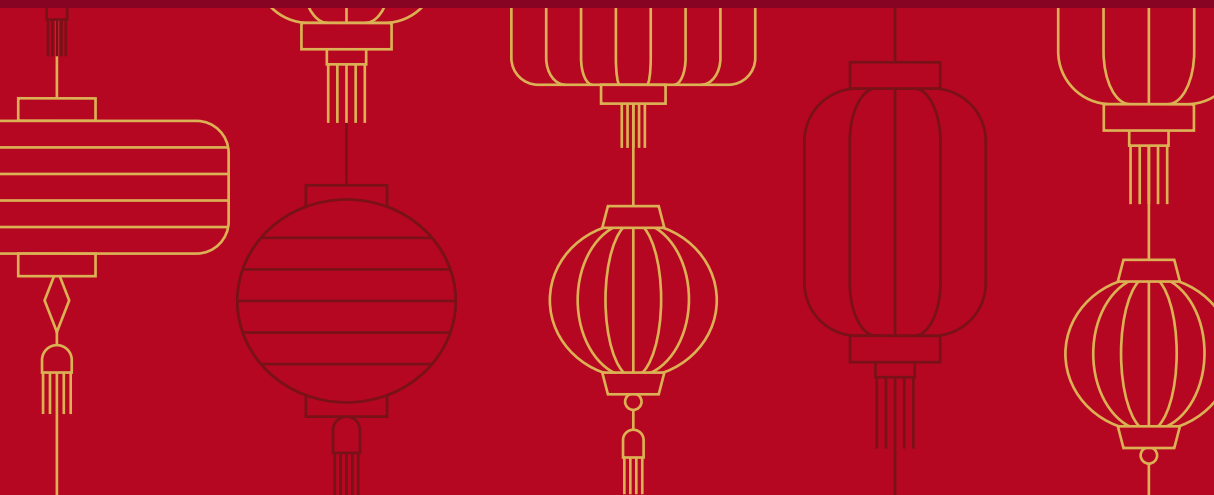
Thanks

~ Arbind



Heap

Identification

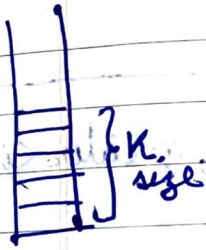
1) K
2) Smallest / Largest

⇒ Heap

Heap

- MinHeap → K+ Largest
- MaxHeap → K+ smallest

Sorting



nlogn ⟶ Merge sort

Using heap

n log K

arr[]: | 7 | 10 | 4 | 3 | 20 | 15 |

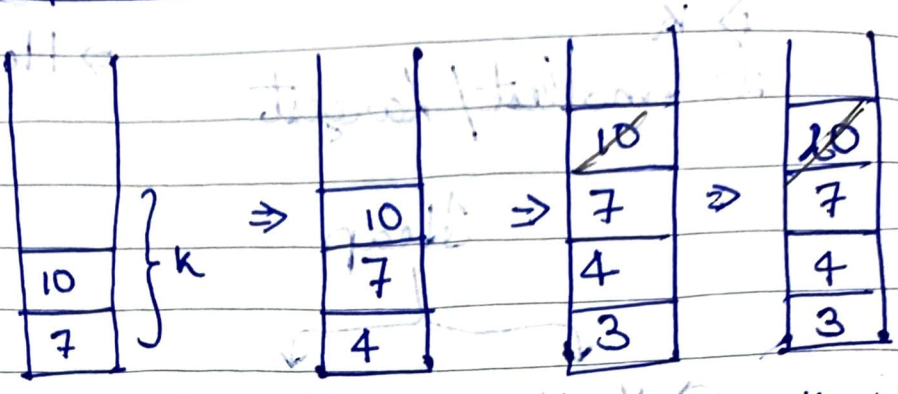
K = 3

If we sort,

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

7 → extra work done to sort this element which has no use

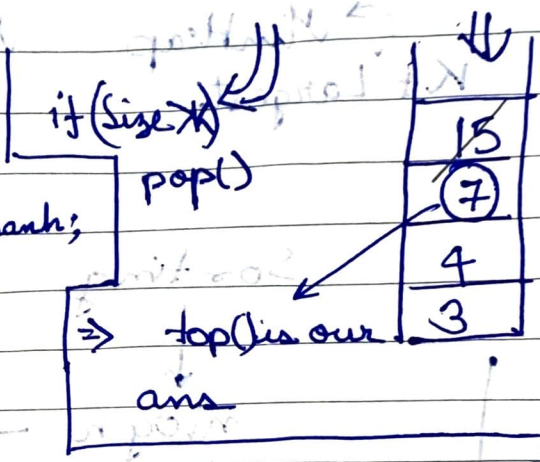
7	10	4	3	20	15



Code

Maxheap

```
priority_queue <int> maxh;
```



Minheap

```
priority_queue <int, vector <int>, greater <int>>
    minh;
```

Using type def

```
type def pair <int, pair <int, int>> ppi;
```

Kth Smallest Element

arr[] : 7 10 4 3 20 15

k = 3

Code

```
priority_queue<int> maxh;
for(int i=0; i<size; i++){
{
    maxh.push(arr[i]);
    if(maxh.size()>K)
        maxh.pop();
}
return maxh.top();
```

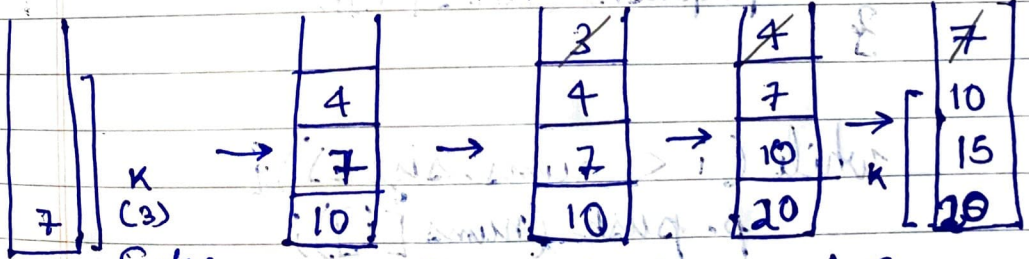
K largest Element

arr[]:

7	10	4	3	20	15

K=3

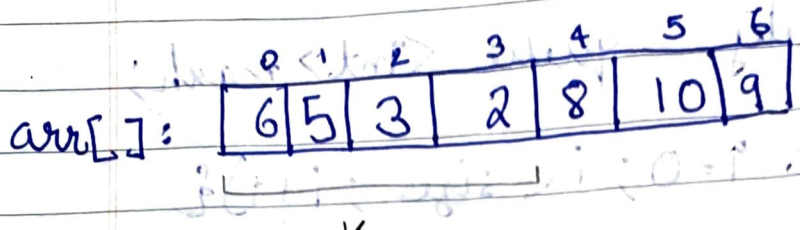
Output: 20 15 10

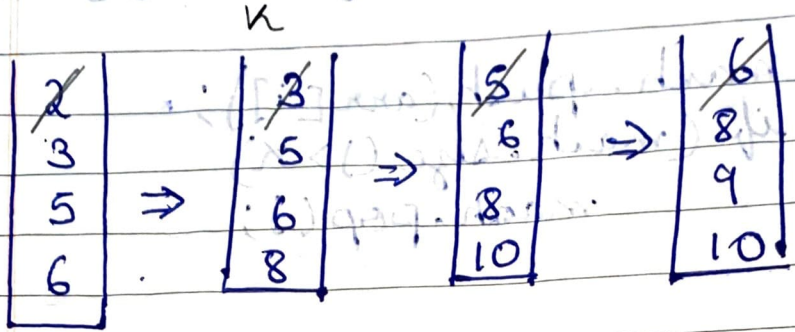


Code:

```
while (!minh.empty()) {
    cout << minh.top() << " ";
    minh.pop();
}
```

Sort a K sorted Array





2 → 3 → 5 → 6 → 8 → 9 → 10

Code:

```
priority_queue<int, vector<int>, greater
                                <int>> p;

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

while(i < nums.size()){
    p.push(nums[i]);
    cout << p.top() << " ";
    p.pop();
}
```

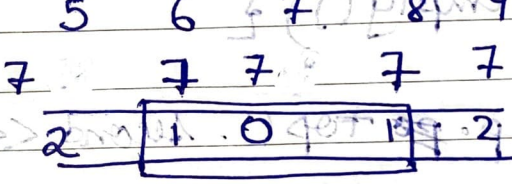
```
while (!p.empty()) {
    cout << p.top() << " ";
    p.pop();
}
```

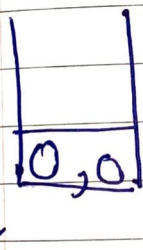
K Closet Number

arr[] : 5 6 7 8 9

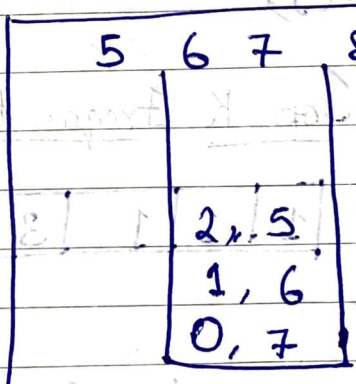
K = 3

x = 7





we will use pair one will b abs(arr[i] - x) other is arr[i]



x = 7

K = 3

elements

```
priority_queue < pair < int, int >> pq;
```

abs (arr[i] - x)

arr [i]

Code:

```
priority_queue <pair<int, int>> p;
for(int i=0; i<size; i++){
        p.push({abs(arr[i]-x), arr[i]});
    if (p.size()>k)
        p.pop();
}

while (!s.empty()){
{
    cout << p.top().second << " ";
    p.pop();
}
```

Top K Frequent Numbers

arr[]:

1	1	1	3	2	2	4

k = 2

1 → 3

3 → 1

2 → 2

4 → 1

Largest
highest
Top

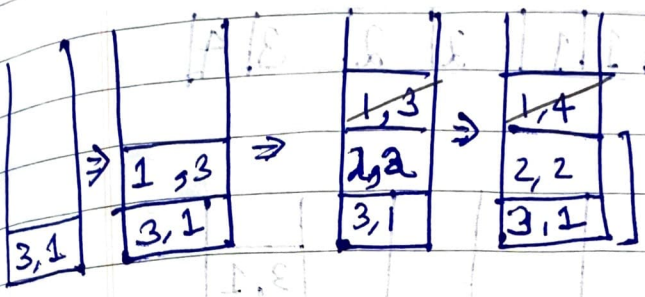
Smallest
Lowest
closest

↓ minheap

↓ maxheap

k=2 | 1 | 1 | 1 | 3 | 2 | 2 | 4 | ⇒

1	3
3	1
2	2
4	1




```
unordered_map <int, int> mp;
priority_queue <int, vector<int>, greater<int>> minh;
for(int i=0 ; i<n ; i++)
    mp [arr[i]] ++
```

```
for( auto i = mp.begin() ; i != mp.end() ;i++){
        minh.push({i->second, i->first});
        if( minh.size() > k){
            minh.pop();
        }
```

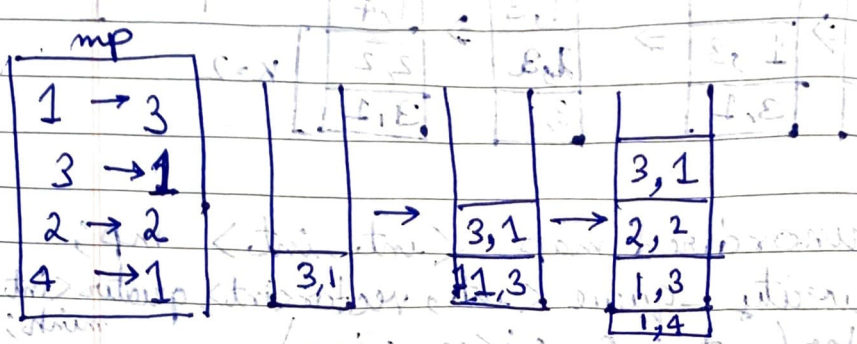
```
while (! minh.empty()){
    cout<<minh.top().second<< endl;
    minh.pop();
}
```

Note:
In priority_queue int is replaced with pair

Frequency Sort (Max, heap)

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

Output | 1 | 1 | 1 | 2 | 2 | 3 | 4 |

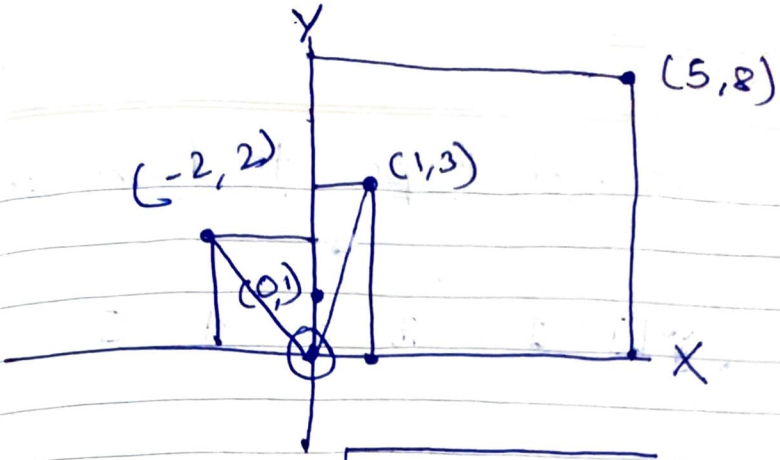


Code:

```
while (mxh.size() > 0) {
    int freq = mxh.top().first;
    int ele = mxh.top().second;
    for(int i=0; i< freq; i++){
        cout << ele << " ";
    }
    mxh.pop()
}
```

K-closest points to Origin

arr[]: =
$$\begin{bmatrix} 1 & 3 \\ -2 & 2 \\ 5 & 8 \\ 0 & 1 \end{bmatrix} \quad k=2$$
x y



$$\text{distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$= \sqrt{x^2 + y^2}$$

⇓

if we sort on basis of x^2+y^2 & remove root the answer will be same

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

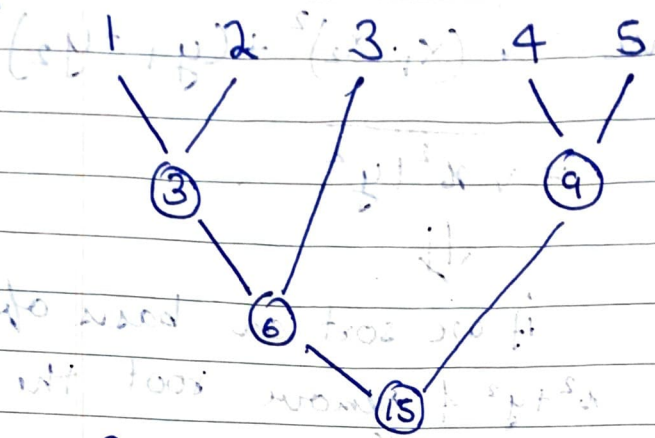
Code:

priority_queue<pair<int, pair<int, int>>> ;

```
for(int i = 0; i < n ; i++){
    maxh.push({arr[i][0] * arr[i][0] +
        arr[i][1] * arr[i][1],
        {arr[i][0], arr[i][1]}});
if(maxh.size() > k){
    maxh.pop();
}
Print the whole queue
}
```

Connect Ropes to Minimise the cost

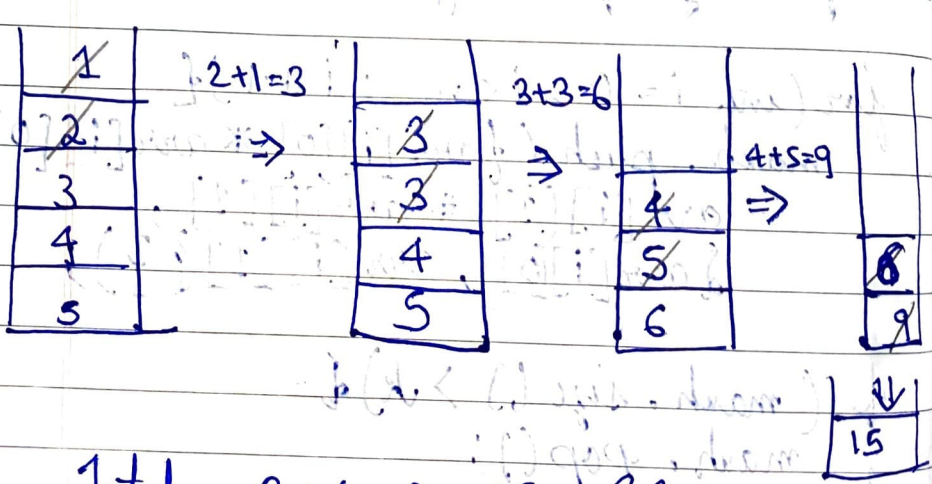
arr[] : 1 2 3 4 5



3 + 6 + 9 + 15
= 33

Concept:

⇒ Minimal number will come when we take 2 ropes with min value



Total = 3 + 6 + 9 + 15 = 33

Code:

```
priority_queue<int, vector<int>, greater<int>> q;

for(int i=0; i<stones.size(); i++){
    q.push(stones[i];
}

int temp=0, ans=0;
while(q.size != 1){
    for(int i=0; i<n; i++){
        temp += q.top();
        q.pop();
    }
    q.push(temp);
    ans += temp;
    temp = 0;
}
return ans;
```

Sum of Elements

arr[]:

1	3	12	5	15	11

K1 = 3

K 2 = 6

1	3	5	11	12	15

↑ 3rd ↑ 6th

Code

```
int n = v.size();
if (n<k1 || n <k2)
        return -1;
if (k2 > k1)
        swap(k1,k2);
priority_queue <int> p;
for(int i=0 ; i<n ; i++){
        p.push(v[i]);
        if(n < k1)
            p.pop();
}
int first = p.top();
for (int i=0; i<k1-k2 ; i++)
        p.pop();
return first + p.top();
```