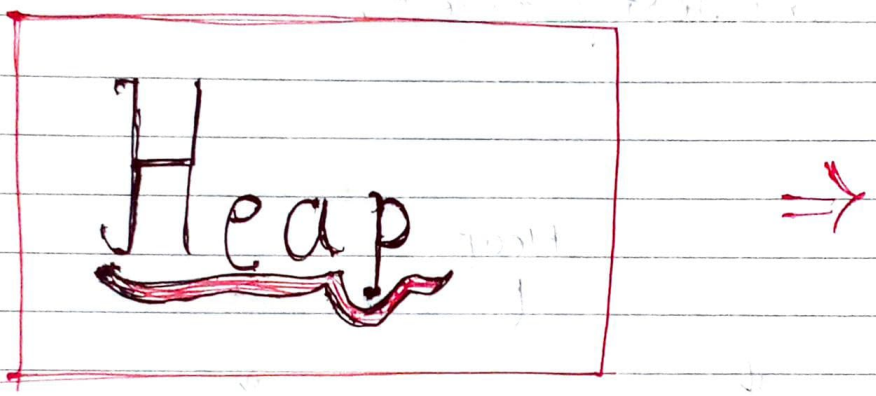
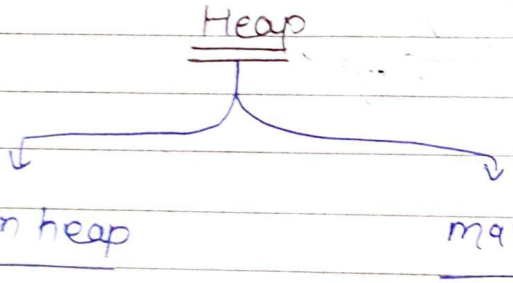

Heap

1.) Heap Introduction and Identification

Identification :-

(1) K .
(2) Smallest / largest .



Imp

K + Smallest ⟶ max.

K + largest ⟶ min.

T.C :- n log k

n log n ⟶ n log k.

CODE

(1.) max heap =>

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

(2.) Min heap =>

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

If we want to store pairs then,

```
...
1, 12
0, 11.
```

At global:

```
#define pair<int, pair<int, int>> ppi;
```

↳ We can replace int with ppi in heap.

2.] **k^{th} Smallest Element.**

Given,

arr[] = 7, 10, 4, 3, 20, 15

k = 3

O/P = 7

Approach 1 ⇒ Sort the array and return arr[k-1]

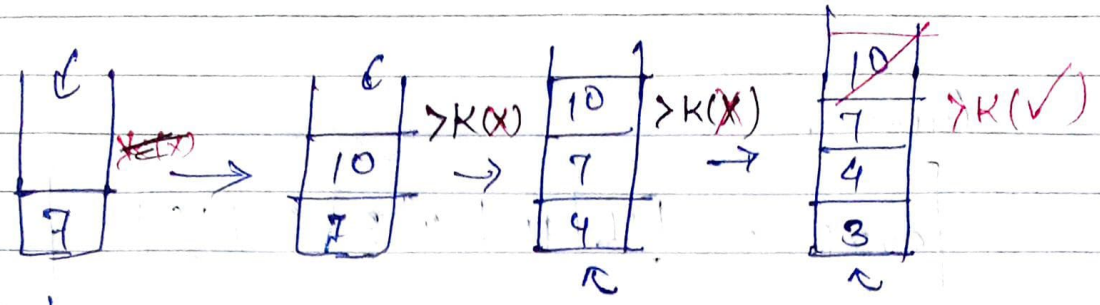
But the T.C → n log n

we can reduce it by using heap

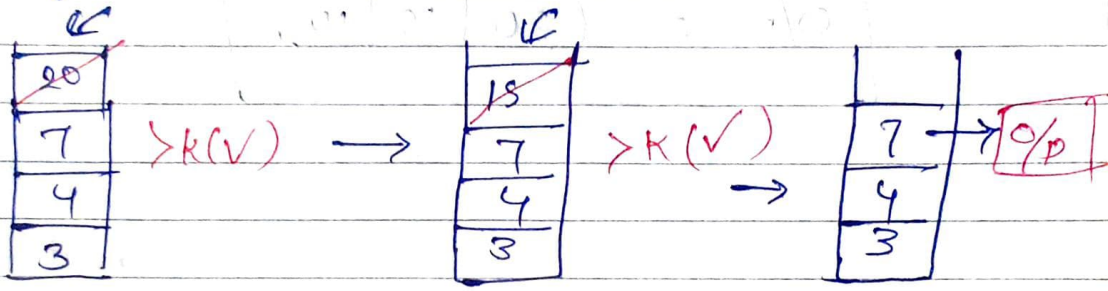
Approach 2 ⇒ n log k

3rd smallest element → max heap

arr →
0 ⟶ n-1



max-heap



CODE

```
int maxHeap(int arr[], int k, int n)
{

  priority_queue<int> maxh;

for(int i=0; i<n; i++)
 {
     maxh.push(arr[i]);
   if(maxh.size()>k)
    {   maxh.pop();
    }
 }

  return maxh.top();
}
```

Q. Return K largest Elements in Array

Given,

arr(.) =	7	10	4	3	20	15

K = 3

O/p =	20	15	10

Note: K && largest ⟶ min-heap.

CODE

```
vector<int> minHeap(int arr[], int k, int n)
{
    priority_queue<int, vector<int>, greater<int>> minh;
    for(int i=0; i<n; i++)
    {
        minh.push(arr[i]);
        if(minh.size() > k)
            minh.pop();
    }
```

```
vector<int> res;
while(!minh().size())
{
    res.push_back(minh.top());
    minh.pop()
}

return res;
}
```

4.

Sort a K sorted Array
or
Sort Nearly Sorted Array

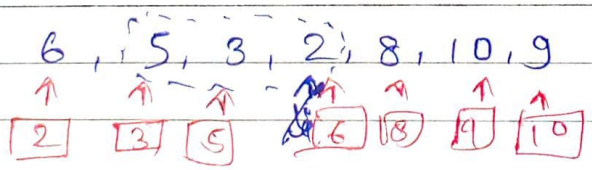
Given,

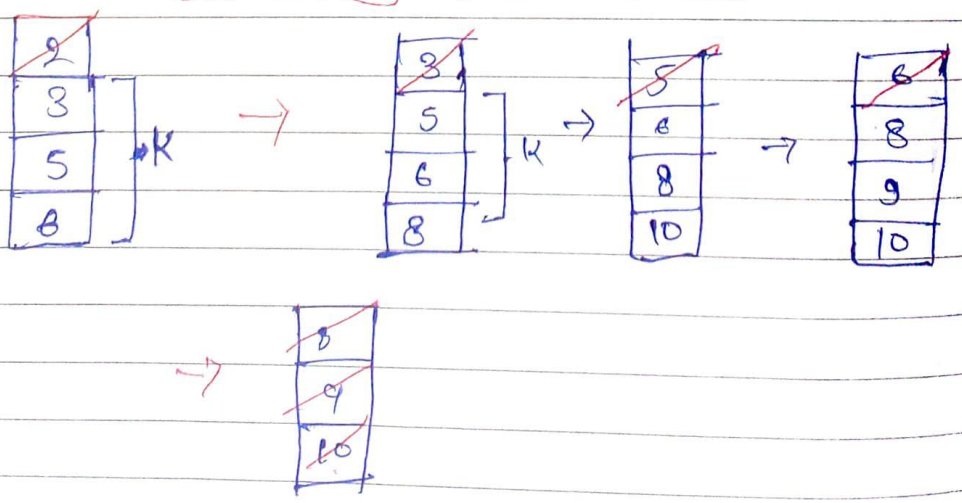
arr : | 6 | 5 | 3 | 2 | 8 | 10 | 9 |

K = 3

O/P ⇒ | 2 | 3 | 5 | 6 | 8 | 9 | 10 |

means, the element at index i can be found in range [i-k, i+k]





CODE

```
vector<int> sortk(vector<int>arr; int k, int n)
 {

priority_queue<int, vector<int>,greater<int>> minh;
vector<int> res;
for(int j=0; j<n; j++)
{
    minh.push(arr[j]);
   if(minh.size() > k)
    {
       res.push_back(minh.top());
        minh.pop();
    }
{
while(!minh.empty())
 {
   res.push_back(minh.top());
       minh.pop();
 }
return res;
}
```

5. **K - closest Number**

Given,

arr[] : | 5 | 6 | 7 | 8 | 9 |

K = 3

x = 7

o/p : | 6 | 7 | 8 | → order doesn't matter

Here we have to use key, value pair where key will be the absolute d/f b/w arr[i] − x.

and min-heap will be used

key	value
pairs of abs(arr[i]-x)	arr[i]

key ← 0 0 → value

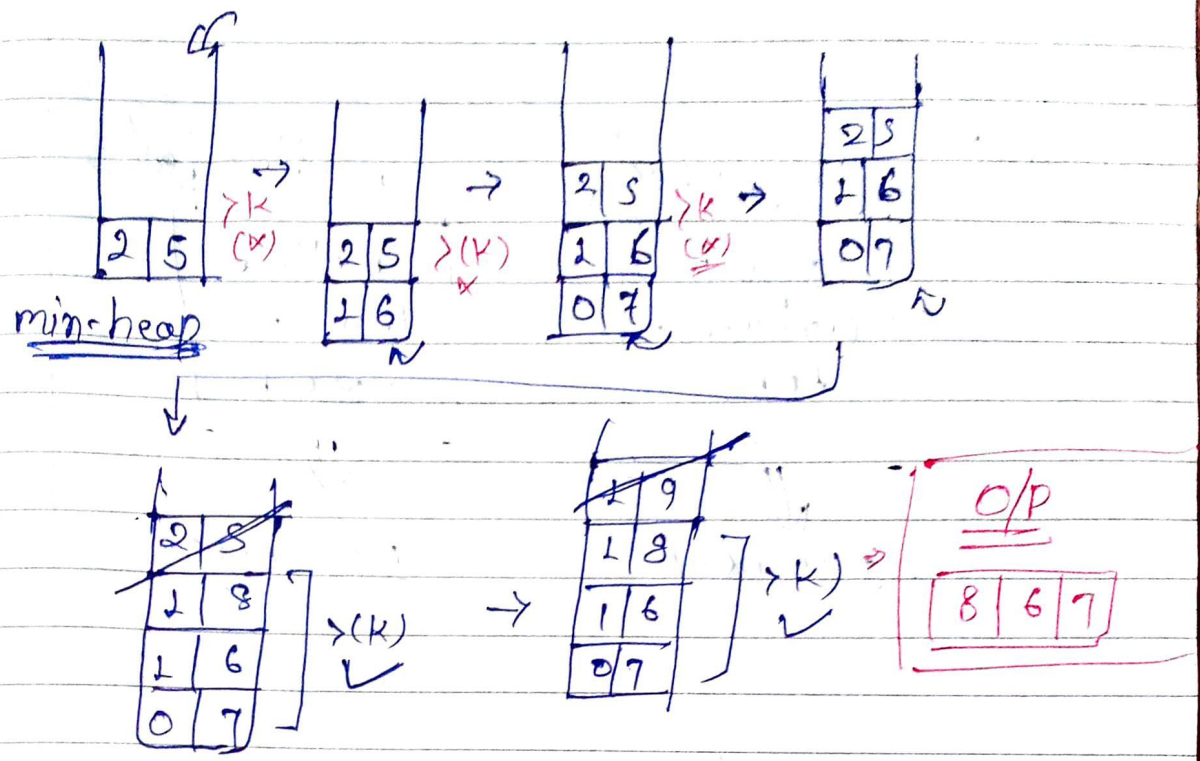
min-heap

Eg :

5, 6, 7, 8, 9

− 7 7 7 7 7

2 | 1 0 1 | 2



CODE #define minh maxh

```
vector<int>kClose(vector<int>arr, int k, int x, int n)
{

priority_queue<pair<int,int>> minh;

for(int i=0; i<n; i++)
{
    minh.push_back({abs(arr[i]-x), arr[i]});
    if(minh.size()>k)
        minh.pop();
}

  vector<int> res;
while(!minh.empty())
{
   res.push_back(minh.top());
    minh.pop();
}
```

return res;
}

6. Top-k Frequent Numbers

Note:

largest feel:
- largest
- greatest
- top
- ...

⇓ min heap

smallest feel:
- smallest
- lowest
- closest
- ...

⇓ max heap

Problem statement:

Given,

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

~~Frequency~~ k = 2

o/p: [1 | 2]

---

1 → 3

3 → 1

2 → 2

4 → 1.

Approach 2)

By using Unordered_map.
and
min-heap.

```
unordered_map<int,int> mp;
for(int i=0; i<n; i++)
{
    mp[arr[i]]++;
}
```

freq. of element

mp

1	3
3	1
2	2
4	1

min heap creation

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

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

minh

```
vector<int> res;
while (!minh.empty())
{
    res.push_back(minh.top().second);
    minh.pop()
}
return res;
}
```

storing minh to res. vector

7.

Frequency Sort

arr = | 1 | 1 | 1 | 3 | 2 | 2 | 4 |

↓ we have to sort it via max frequency.

O/P | 2 | 1 | 1 | 2 | 2 | 3 | 4 |

Note :- This Q. is same as previous (6).

arr

↓

unordered_map (freq, store).

↓

max-heap (we won't check the size of k).

↓

store in vector

↓

return

CODE is super-simple

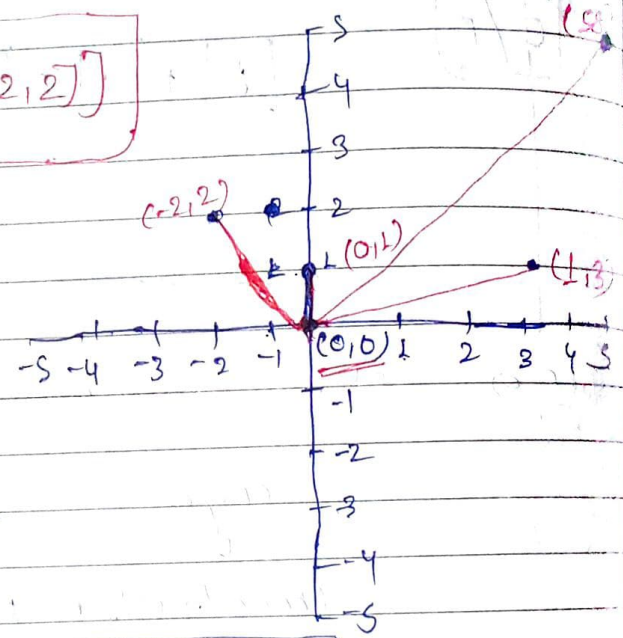
8. K-Closest Points to Origin

Given,

$$arr = \begin{bmatrix} x_1 & y_1 \\ 1 & 3 \\ -2 & 2 \\ 5 & 8 \\ 0 & 1 \end{bmatrix} \qquad K = 2$$

Note: order doesn't matter

o/p ⇒ [[0,1], [-2,2]]



We can use distance formula.

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

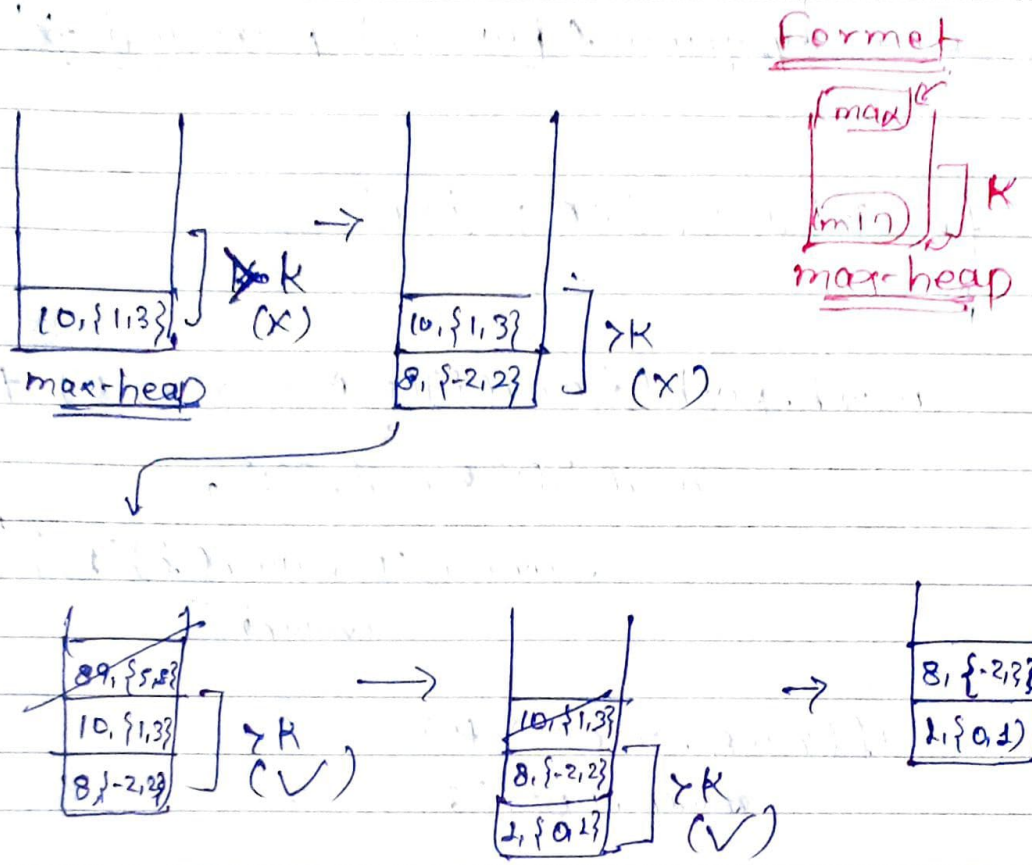
As we have from origin, $(x_2, y_2) = (0, 0)$.

$$dist = \sqrt{x_1^2 + y^2}$$

As we have to compare,

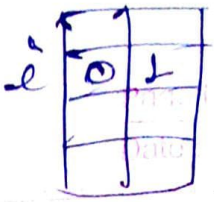
$$\left(\sqrt{x^2+y^2}\right)^2 \Rightarrow \boxed{x^2+y^2}$$

max-heap will be applied ✓



any ✓ [[-2,2], [0,1]]

[[0,1], [-2,2]] is also accepted.



CODE

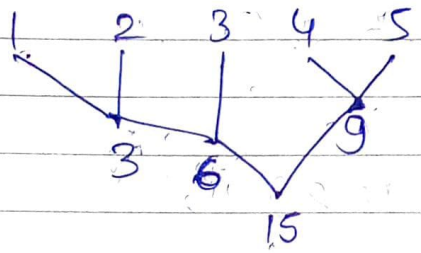
```
vector<vector<int>> res;

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

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] } });
                  (key)                    (value)

   if(maxh.size() > k)
      maxh.pop();
}

while (!maxh.empty())
{
   pair<int,int> p = maxh.top().second;
   res.push_back(p);
   maxh.pop();
}

return res;
```

Q. Connect Ropes to minimize the cost.

Given

arr = [1, 2, 3, 4, 5]

O/P = 33



=> cost (33)

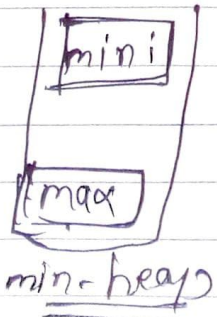
cost
↓
3 + 6
+ 9 + 15
↓
33

optimal O/P

Note =>

To get optimal output we have to add two smallest numbers each time

min-heap is required =>



min-heap

CODE

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

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

while(minh.size() >= 2)
{
    int first = minh.top();
    minh.pop();
    int second = minh.top();
    minh.pop();
    cost = cost + first + second;
    minh.push(first + second);
}

return cost;
```

16. **Sum of Element**

Given

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

K1 = 3
K2 = 6

O/P => 23

Approach 1) sort and traverse.

| 1 | 3 | 5 | 11 | 12 | 15 |

3rd 6th

(11+12) => 23

Approach 2)

Use maxHeap.

and

traverse.

CODE

```
int sumElement(int arr[], int k, int n)
{

    priority_queue<int> maxh;

    for(int i=0; i<n; i++)
    {
        maxh.push(arr[i]);
        if(maxh.size() > k)
            maxh.pop();
    }

    return maxh.top();
}


int main() {
    int first = sumElement(arr, 3, arr.size());
    int second = sumElement(arr, 6, arr.size());
    int sum = 0;
    for(int i=0; i<n; i++)
    {
        if(arr[i] > first && arr[i] < second)
            sum += arr[i];
    }
    return sum;
}
```