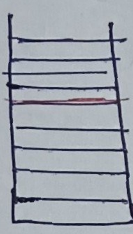
Heap Introduction and Identification.

Heap
- min
- max

→ K
→ Smallest / Largest

K th smallest → max heap
K th largest → min heap

If ask Kth largest.



Heap-top पर answer आ जाता.

use of K

arr[] [| | | |]
say sort arr[]
so use of merge sort

Time complexity O(nlogn)
but using heap ↓
O(nlogk)

Q find Kth smallest element

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

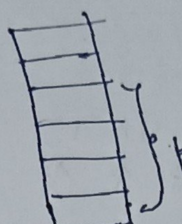
K=3

0	1	2			
3	4	7	10	15	20

sort

nlogn.

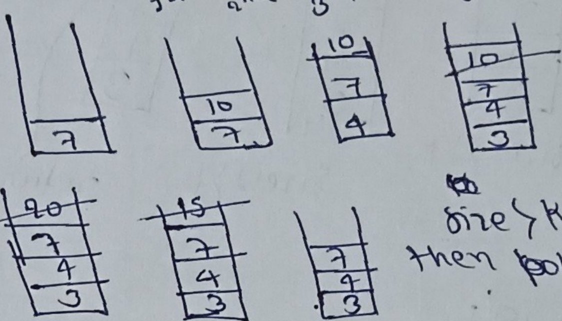
we need only Kth but use to all sort. so, now only sort Kth smaller.



यहाँ K के ऊपर sorting नहीं होंगे तो nlogk आ जायेगा.

K=3

arr[] | 7 | 10 | 4 | 3 | 20 | 15 |
1st 2nd 3rd 4th 5th



pop element greater than Kth smaller element.

define Heap STL.

max Heap

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

min heap

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

→ when need of pair then

```
pair<int, pair<int>>
```

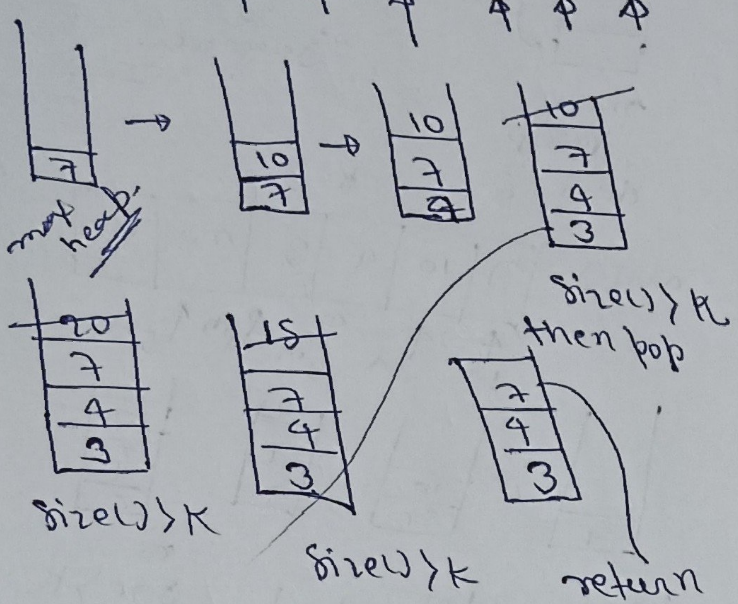
and replace every int.

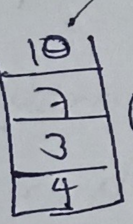
but define

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

Kth smallest element

arr[]: 7 10 4 3 20 15





heap not sure this sort. sorted / unsorted भी हो सकता है।

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();
```

Return Kth largest element

arr[]: 7 10 4 3 20 15

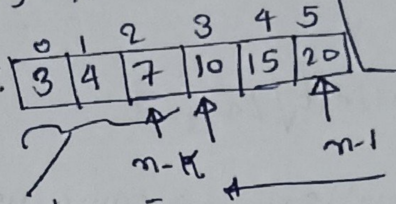
Key = (K) = 3

F2F interview
sort → nlogn
Heap → nlogK

Problem statement

op: 20 15 10

sorting:



ये extra work here sorting K like. = loop from end to n-K+1 = 20 15 10

nlogn

ये हम reduce कर सकते हैं heap से।

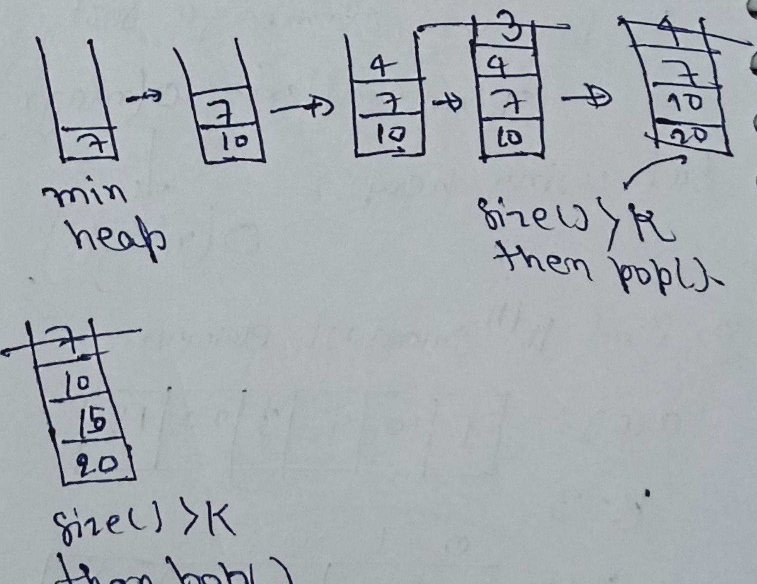
Identification of heap

→ given K & largest

Approach heap.

arr[]: 7 10 4 3 20 15

K = 3



```
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> v;
while(minh.size() > 0){
    v.push(minh.top());
}
return v;
```

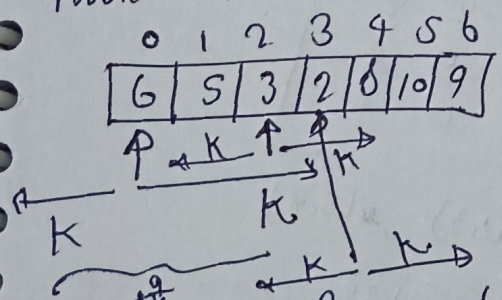
(nearly sorted).

#sort a k sortest array.

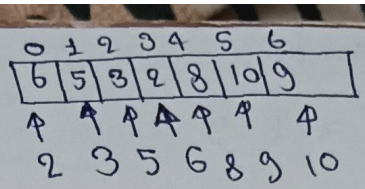
arr[]: | 6 | 5 | 3 | 2 | 8 | 10 | 9 |

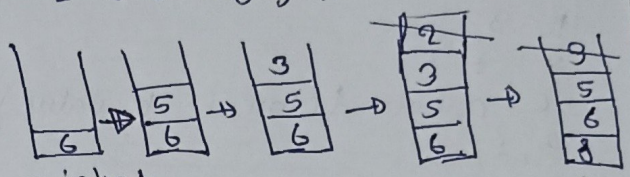
k = 3

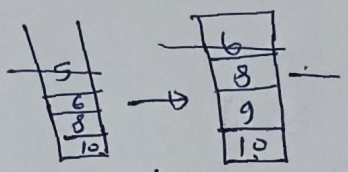
Problem Statement

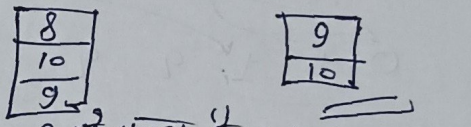


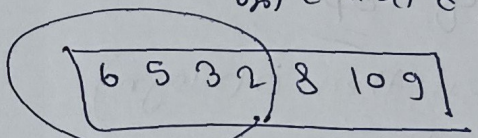
i. index or left element
h. left or right of arr[i]
of k element hoga.











only यहाँ तक
sort होगे।

n log k

Code

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

K Closest Number

arr[]: 5 6 7 8 9
K = 3
X = 7
↳ around (closest K value) find out.

op: 6 7 8

Approach.

```
  5 6 7 8 9
- 7 7 7 7 7
-----------
abs: 2 1 0 1 2
```

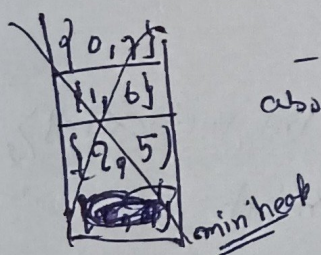
0 1 1 2 2

7 6 8 op

If we do sort then O(nlogn) हो जाएगा so,

heap से करते O(nlogK) में;

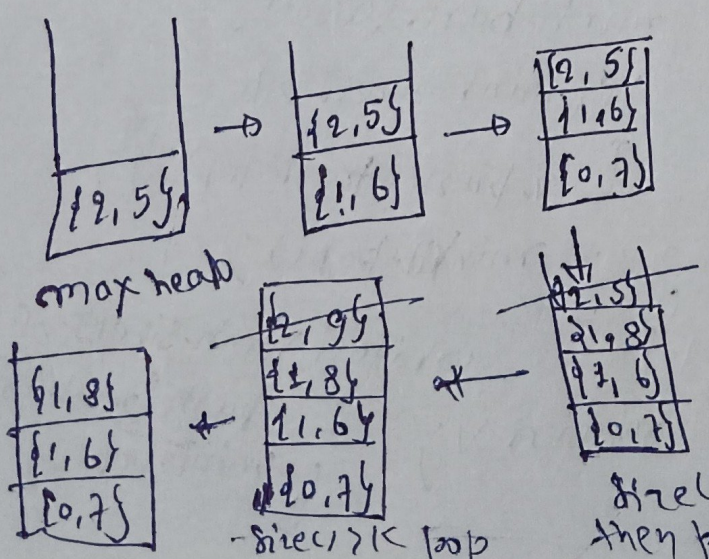
```
  5 6 7 8 9
- 7 7 7 7 7
-----------
abs 2 1 0 1 2
```



size < K

{diff, arr[i]}

abs(arr[i] - x)



ans: 8 7 6

code

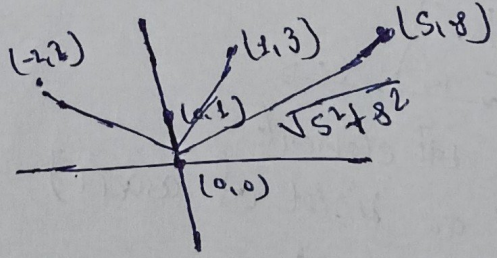
```
priority_queue < pair<int,int> > maxh;
for(int i=0; i<n; i++)
{
  maxh.push(abs(arr[i]-x), arr[i]);
  if(maxh.size() > k)
  { maxh.pop();
  }
}
vector<int> v;
while(maxh.size() ≥ 0)
{ v.push(maxh.top()); second
  maxh.pop();
}
return v;
```

frequency sort

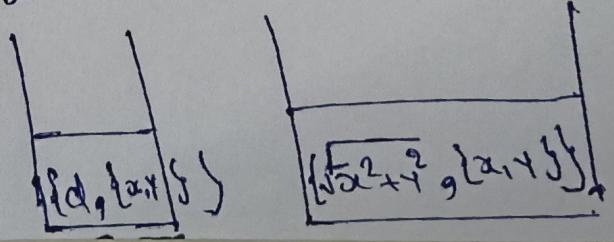
arr: 1 1 1 3 2 2 4
op 1 1 1 2 2 3 4
freq: 3 3 3 2 2 1 1 ← maxheap frequency base

K closest point to origin.

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

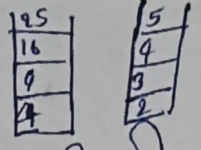


d: $\sqrt{x^2+y^2}$ → minimise करना है।

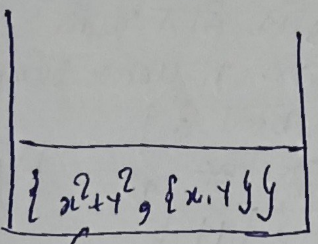


4 9 16 25 →

2 3 4 5 →



$\sqrt{x^2+y^2}$ से अच्छा x^2+y^2 है सही।



arr[n][2]

(pair<int, pair<int,int>>

distance minimize करना है तो
यहां max heap लगेगा।

Code

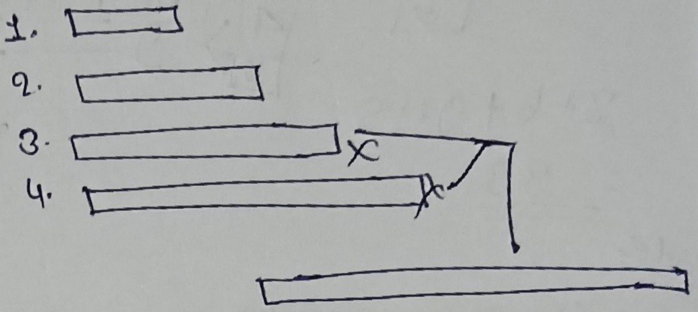
~~priority_queue<pair<pair<int,int>>~~

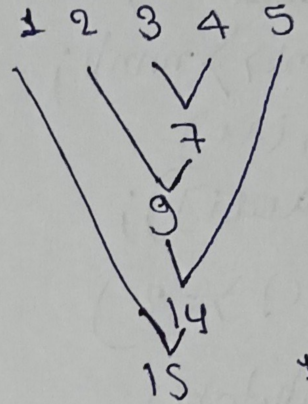
```
priority_queue<pair<int, pair<int,int>>>
    maxh;
for(int i=0; i<n; i++) {
  maxh.push(
  {arr[i][0]*arr[i][0] + arr[i][1]*
     arr[i][1], {arr[i][0], arr[i][1]}});
  if (maxh.size() > K)
    { maxh.pop();
    }
}

while(maxh.size() > 0)
{
  pair<int,int> p = maxh.top().second;
  Cout << p.first << " " << p.second << "\n";
  maxh.pop();
}
```

~~return~~

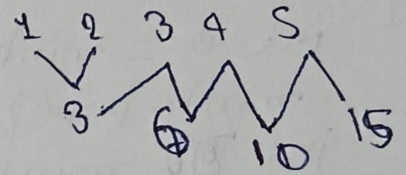
Connect Ropes to ~~minise~~ minimize the cost.

arr[]: 1 2 3 4 5

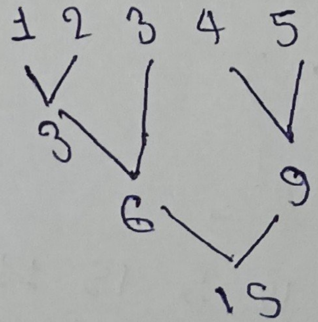




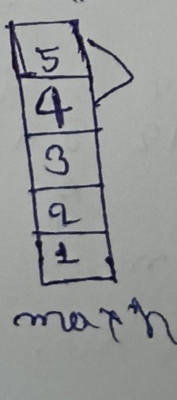
7+9+14+15
= 45



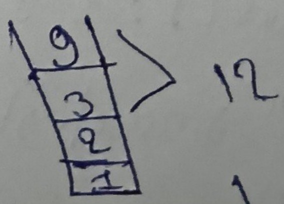
~~3+7+11~~
3+6+10+15
= 34

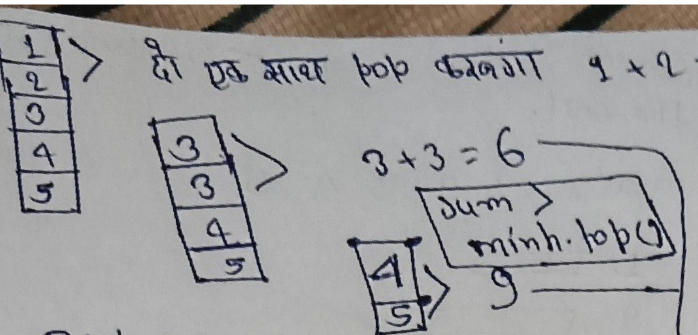


approach 3+6+15+9 = 33



5+4 = 9





3+6+9+15
= 33

sort nlogn
1 3 5 11 12 15
3rd smallest — 6th smallest
इसके बीच जितने no h uska sum करना है।

अब इसको nlogk में करना है।

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 a = minh.top();
    minh.pop();
    int b = minh.top();
    minh.pop();
    int sum += a+b;
    minh.push(a+b);
}
return sum;
```

Sum of Element

arr[] : 1 3 12 5 15 11
$k_1 = 3$
$k_2 = 6$

PS:

0	1	2	3	4	5
1	3	12	5	15	11

k_1th smallest no.
k_2th smallest no.

1	3	12	5	15	11

kth smallest no.

```
a = k1th smallest(arr[], k1);
b = k2th smallest(arr[], k2);
for(int i=0; i<n; i++)
{   if(arr[i] > a && arr[i] < b)
    {   sum += arr[i]
    }
}
return sum;
```