Heaps
- insert → log N
- delete → log N
- getMin() / getMax() → O(1)

N → O(n log n)
⇓
O(N)

Q⇒ Given N distinct element, return first K small elements

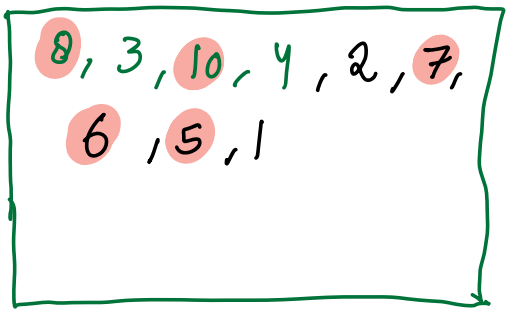
arr = { 8, 3, 10, 4, 11, 2, 7, 6, 5, 1 }
K = 4 ⇒ { 1, 2, 3, 4 }

Appl: (↗ min)
Insert all elements in heap & delete k elements,
O (N log N + k log N)

App2
Max Heap

K = 4

arr = { 8, 3, 10, 4, 11, 2, 7, 6, 5, 1 }



Obs 1: ele > get_max() that ele cannot be part of ans

Obs 2: ele < get_max()
a) insert in my max heap
b) delete max from heap

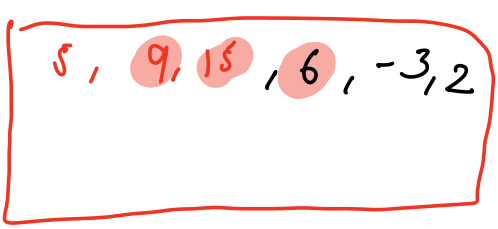
① inserted first k elem in heap ⇒ $K \log K$

② for remaining ele. ⇒ $[n-k]\ (\log_2 k)$

$O((n-k)\log k + K \log k)$

⇒ 5, 9, 15, 19, 6, -3, 2

k = 3



Median

arr [5] = { 2, 9, 6, 4, 5 }

↳ 2, 4, 5, 6, 9

arr [7] = { 2, -1, 10, 4, 15, 3, -2 }

↳ -2, -1, 2, 3, 4, 10, 15

arr[6] = {-1, 10, 3, 9, 6, 2}

↳ { -1, 2, 3, 6, 9, 10 }

$\frac{3+6}{2} = \frac{9}{2} = 4$

arr [8] = { 2, 4, -3, 12, 6, 24, 20, 10 }

↳ { -3, 2, 4, 6, 10, 12, 20, 24 }

$\frac{6+10}{2} = 8$

Q) Given an array find median of all prefix subarray

arr [5] = { 9, 6, 3, 10, 4 }

Prefix subarrays :

{9} → 9

{9, 6} → {6, 9} → 15/2 = 7

{9, 6, 3} → {3, 6, 9} → 6

{9, 6, 3, 10} → {3, 6, 9, 10} → 7

{9, 6, 3, 10, 4} → 6

arr[5] = [9], 6, 3, 10, 4

9

9, 6

6, 9

O(N²) ← Insertion sort

Optimization

Case 1

arr[9] = {3, 1, 6, 18, 14, 2, 17, 12, 9}

↳ {1, 2, 3, 6, 9, 10, 12, 14, 17}

{3, 1, 2, 6} — 1st half

{14, 10, 12, 17} — 2nd half

obs 1: Max ele of 1st half < Min ele of 2nd half

obs 2: | size of 1st half − size of 2nd half | = 1

median = Max value of 1st half

Case2 arr[10] = {3, 4, 16, 12, 10, 14, 8, 9, 2, 1}

↳ {1, 2, 3, 4, 8, 9, 10, 12, 14, 16}

1st half | 2nd half

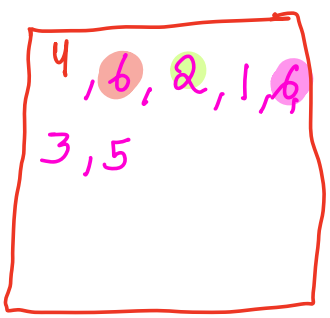
obs1: |size 1st half − size 2nd half| = 0

obs2: Max ele of 1st half < Min ele of 2nd half

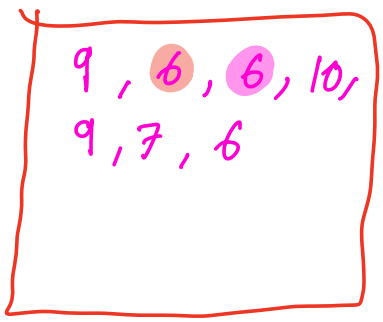
Median = $\frac{\text{Max of first half} + \text{Min of 2nd half}}{2}$

ex:

4, 9, 6, 2, 1, 10, 9, 7, 3, 5



Max Heap



Min heap

Break: 8:58

```
int [] Median (int arr[])
{
    int n = arr.length;
    int ans[n];
Max heap <int> maxh;
  maxh.insert (arr[0]);
Min heap <int> minh;
ans[0] = arr[0];
i= 1; i<n; i++ {
     if (arr[i] < maxh.get_max()
    }
          maxh.insert (arr[i]);

     }
     else
      {
          minh.insert (arr[i];
      }
  if( minh.size() > maxh.size())
    {
      int ele = minh.get_min();
       minh.delete_min();
       maxh.insert(ele);
    }
  else if( maxh.size() - minh.size() >1)
```

2N logN
O(N logN)

```
{
    int ele = maxh.get_max();
    maxh.delete_max();
    minh.insert(ele);
}
int s = (i+1)
if (s%2 == 0)
{
  ans[i] = [maxh.get_max() + minh.getmin()
                    2
}
else
{
  ans[i] =  maxh.getMax();
}
}
return ans;
}
```

// Priority Queue < >