Todays Content:

a) Heap Intro

b) k^{th} smallest elements

c) Running median

Note: Session starts at 9:07PM sharp.

PriorityQueue : Arranges data based on their priority values

Functions:

1. add( ) : add element in PriorityQueue : log(N) : N = no of elements
2. peek() : return peek element : O(1) : peek : highest priority
3. remove() : removes peek element : logN : N = no of elements
4. size() : returns size : O(1)

minPQ: Priority based on minvalues

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
pq.add(19);
pq.add(10);
pq.add(15);
pq.add(18);
```

pq:

19 ~~10~~ 15 18

How is it implented?

: link recording

output

1. Print(pq.peek()); 10
2. pq.remove(); remove 10
3. Print(pq.peek()); 15
4. Print(pq.peek()); 15

Note: If want own ordering, overide comperator

18

Given ar[N] distinct elements, return k^{th} smallest element in ar[]

	0	1	2	3	4	5	6	7	8	9
ar[10] :	8	-3	10	5	11	2	7	9	14	-1

k = 4 : ans = 5

	0	1	2	3	4	5	6	7
ar[9] :	-3	6	2	0	8	7	10	4

k = 3 : ans = 2

Ideas:

1. Sort arr[] return k^{th} ele : return arr[k-1]

   TC: $O(N \log N + 1)$

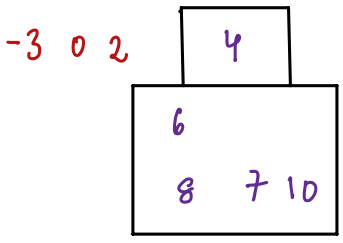
2. Add all arr[] elements in Priority Queue: {Priority ⇒ Min}

   a. Remove k-1 peek elements
   b. Return peek element.

   } Note: At max size = N

	0	1	2	3	4	5	6	7
ar[9] :	-3	6	2	0	8	7	10	4

k = 4



Steps: 1. Remove 3 peek elements

2. return peek ele = 4

TC: $O(N \log N + k \log N + 1)$ SC: $O(N)$

3. k^{th} smallest ele ⇄ Using Priority Queue {Order: Max}

	0	1	2	3	4	5	6	7	8	9
ar[10] :	8	-3	10	5	11	2	7	9	14	-1

k = 4 :

Nok:

Priority: Max

Peek: 10 6 7

5
-3 2 -1

Final ans: 5

Steps: 1. Create Priority Queue: Priority ⇒ Max Priority

2. Insert first k elements in pq: {i : [0 k-1]} Size = k

3. i = k; i < N; i++) {

```
    // When can ar[i] be an ans?
    if ( pq.peek() > ar[i]) {
        pq.remove() // peek is removed
        pq.add(ar[i])
    }
}
```

4. return pq.peek();

TC: $O(k \log k + (N-k) * (\log k))$ SC:

$\approx O(\cancel{k \log k} + N \log k - \cancel{k \log k})$

TC: $\approx O(N \log k)$ SC: $O(k)$

It's slighty better than sorting

10:20pm

Median: Point can divide into 2 parts { 1st half all <= 2nd half all }

ar[5] = { 2 9 6 4 5 }

{ 2 4 **5** 6 9 } median = 5

ar[9] = { 2 -1 10 4 15 3 -2 }

{ -2 -1 2 **3** 4 10 15 } median = 3

ar[6] = { -1 10 3 | 9 6 2 }

4.5

{ -1 2 **3 6** 9 10 } median

↳ median avg = [3+6]/2 = 4.5

ar[8] = { 2 4 -3 12 | 6 24 20 10 } median

8

{ -3 2 4 **6 10** 12 20 24 }

↳ median avg = [6+10]/2 = 8

2Q) Given an array print median of all prefix Subarrays:

{Subarray Starting index = 0}

Note: N Prefix Subarrays

	0	1	2	3	4
ar[5] =	9	6	3	10	4

Subarray:		median:
[0 0]	{9}	9
[0 1]	{9 6} → {6 9}	= {9+6}/2 = 7.5
[0 2]	{9 6 3} → {3 6 9}	= 6
[0 3]	{9 6 3 10} → {3 6 9 10}	= {9+6}/2 = 7.5
[0 4]	{9 6 3 10 4} → {3 4 6 9 10}	= 6

Ideal: Sorting all prefix subarrays using Insertion Sort.

TC: O(N²) SC: O(1)

	0	1	2	3	4	median
	9	6	3	10	4	
[0 - 0] =	9					{9}
[0 - 1] =	9	6				
	6	9				{7.5}
[0 - 2]	6	9	3			
	3	6	9			{6}
[0 - 3]	3	6	9	10		
	3	6	9	10		{7.5}
[0 - 4]	3	6	9	10	4	
	3	4	6	9	10	{6}

Code: TODO

```
i = 0; i < N; i++){
}
```

Optimization:

	0	1	2	3	4	5	6	7	8
ar[9] =	3	1	6	10	14	2	17	12	9
	1	2	3	6	$\boxed{9}$	10	12	14	17
	6	2	3	1	$\boxed{9}$	12	17	14	10

1st half & 2nd half

obs: All ele left & All ele on right

1. max ele left & min ele on right
2. 1st half size − 2nd half size = 1 } Include median in 1st half
3. Max ele of 1st half

	0	1	2	3	4	5	6	7	8	9
ar[10] =	3	4	16	12	10	14	8	9	2	1
	1	2	3	4	$\boxed{8}$	$\boxed{9}$	10	12	14	16
	3	4	2	1	$\boxed{8}$	$\boxed{9}$	16	12	10	14

1st half & 2nd half

obs: All ele 1st h & All ele on 2nd h

1. max ele 1st h & min ele on 2nd h
2. 1st half size − 2nd half size = 0 } = 1st ele on 1st half 2nd ele on 2nd half
3. Median = (max 1st half + min 2nd half)/2

Combine obs:

1. max ele 1st h & min ele on 2nd h
2. 1st half size − 2nd half size = 1 or 0

Odd:

Median = Max ele of 1st half

even:

Median = (max 1st half + min 2nd half)/2

	0	1	2	3	4	5	6	7	8	9
ar[10] = {	4	6	9	2	1	10	14	7	3	5 }

operations

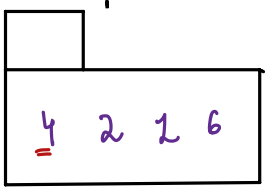
1: max ele

2: Insert

3: rem max

4: size

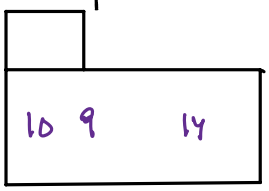
1st half Bon



max 1st half = min 2nd half ele

1st half size − 2nd half size = 0 1

2st half Bon



operations

1: min ele

2: Insert

3: rem min

4: size

1st half Bon = PQ : { Priority: MAX }

2st half Bon = PQ : { Priority: MIN }

ele: Insert	bon1 − bon2 = 0,1	Transfer	Median
4 : Insert in 1st h:	1 − 0 = 1	* no need	Odd: max 1st = 4
6 > max 1st 4 : Insert 2nd h	1 − 1 = 0	* no need	even: (max 1st + min 2st)/2 = 5
9 > max 1st 4 : Insert 2nd h	1 − 2 = −1 ?	✓: min 2nd h → 1st h	
	2 − 1 = 1		odd: max 1st = 6
2 > max 1st 6 : Insert 1st h	3 − 1 = 2	✓: max 1st → 2nd h	even: (max 1st + min 2st)/2 = 5
	2 − 2 = 0		
1 > max 1st 4 : Insert 1st h	3 − 2 = 1	* no need	Odd: max 1st = 4
10 > max 1st 4 : Insert 2nd h	3 − 3 = 0	* no need	even: (max 1st + min 2st)/2 = 5
14 > max 1st 4 : Insert 2nd h	3 − 4 = −1	✓: min 2nd h → 1st h	
	4 − 3 = 1		Odd: max 1st = 6

```
void medians (int ar[]){   TC: O(NlogN) SC: O(N)
    int N = ar.length;
    PriorityQueue manpq; // order based on max → 1st half
    PriorityQueue minpq; // order based on min → 2nd half
    print(ar[0]) // median of sub[0 0]
    manpq.add(ar[0])
    for(int i=1; i<N; i++){
        int n = ar[i];
        if( n <= manpq.peek()) { manpq.add(n)} // n <= max 1st h : Insert 1st h
        else { minpq.add(n)}
            // 1st h size - 2nd h size > 1 : Transfer max of 1st → 2nd h
        if( manpq.size() - minpq.size() > 1 ){
            int max = manpq.peek()
            manpq.remove(); minpq.add(max)   } logN
        }
        else if(manpq.size() - minpq.size() < 0)
            // 1st h size - 2nd h size < 0 : Transfer min of 2nd → 1st h
            int min = minpq.peek()
            minpq.remove(); manpq.add(min)   } logN
        }

        int total = minpq.size() + manpq.size();
        if( total % 2 == 1){ // odd elements → max of 1st half
            print( manpq.peek())
        }
        else { // even ele → ( max of 1st h + min of 2nd h)/2
            print( [manpq.peek() + minpq.peek()]/2)
        }
    }
}
```

Note: Each priority size at max = N/2 = O(N)

TC: O(N * {1 + logN + 2 + logN + 1 + }) = O(NlogN) SC: O(N)