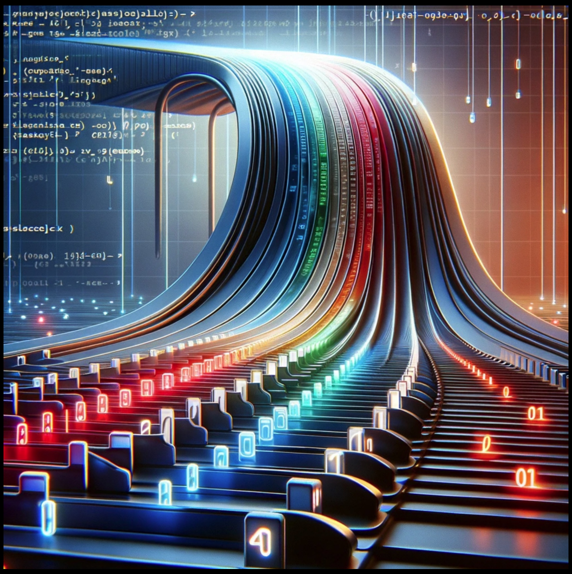
Priority Queue :



Heap by heap, we're building a mountain of problem-solving success in this heap-based world!

Hello Everyone

very Good Evening

We will Start from 9:06 pm

🙏 Heaps 🙏

~~~~~~~~~~

Agenda :

~~~~~~~~~~

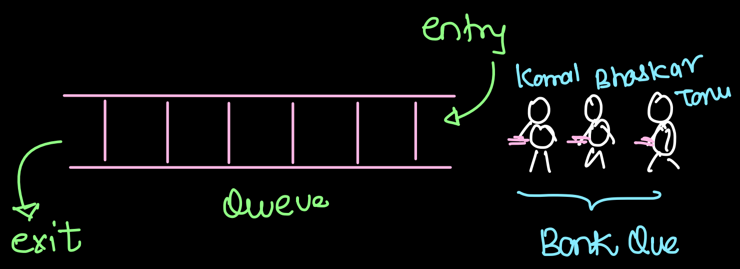
Intro. {
1. Introduction to Heaps / Priority Queue
2. Priority Queue of an object

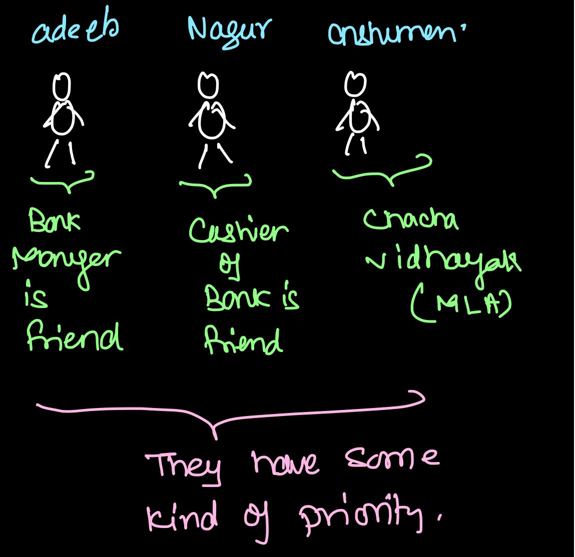
easy problem {
3. Kth Smallest Element

4. Running Median } Hard problem.

Introduction of Priority Queue:

Queue: FIFO → First In First Out





Priority Queue

→ Min Priority Queue (minPQ)

→ max. Priority Que. (max PQ)

Integers array:

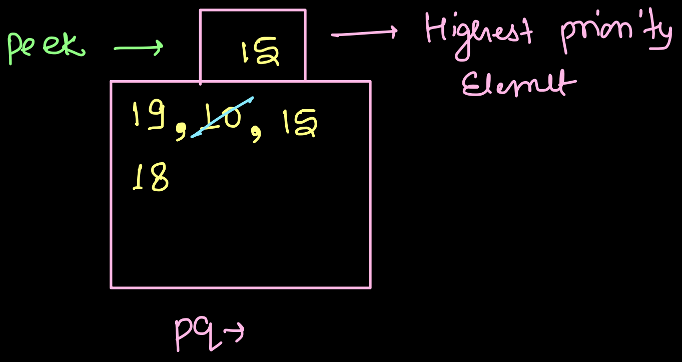
Higest priority element : min elemet of array.

Higest priority element : Max element of array.

Syntax:-

```
✓PriorityQueue<Integer> pq = new PriorityQueue<>();
                    ↳Default pq is minPQ in JAVA.

✓pq.add(19);
✓pq.add(10);
✓pq.add(15);
✓pq.add(18);
✓SOPln(pq.peek()); → 10
✓pq.remove(); → 10 will removed
✓SOPln(pq.peek()); → 15
```



NOTE: By Default Java have min PQ.

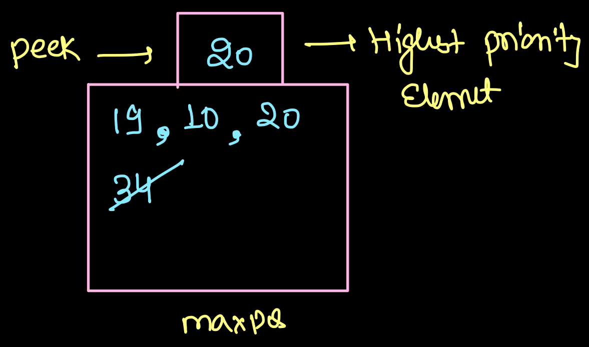
functions	working	time complexity
① pq.add(x) →	add x element in PQ and arrange that element according to priority.	→ O(logn)
② pq. peek() →	Return Highest priority Element	→ O(1)
③ pq. remove() →	It will remove highest priority element and arrange all element according to their priority.	→ O(logn)
④ pq. size() →	No. of element in PQ.	→ O(1)

Syntax of max Priority Queue:

```
PriorityQueue<Integer> pq = new PriorityQueue<>( );
                                                ↑
                                   Collections.reverseOrder()
```

pq → It will work as maxPQ

```
pq.add(19);
pq.add(10);
pq.add(20);
pq.add(34);
sopln(pq.peek()); → 34
pq.remove(); → 34
sopln(pq.peek()); → 20
```



Kth Smallest Element

Given an Array A[] and a value K. Find Kth smallest element from it.

8	4	10	5	11	9	7	6	14	1
0	1	2	3	4	5	6	7	8	9

K	ans
2	→ 4 (2nd smallest)
5	→ 7 (5th smallest)

Idea 1: Sort the array and return arr[k-1].

arr →

8	4	10	5	11	9	7	6	14	1
0	1	2	3	4	5	6	7	8	9

After Sorting:

arr:

1	4	5	6	7	8	9	10	11	14
0	1	2	3	4	5	6	7	8	9

K=2, ans = arr[K-1] ⇒ arr[2-1] = arr[1] = $\boxed{4}$ A

K=5, ans= arr[K-1] ⇒ arr[5-1] = arr[4] = $\boxed{7}$ m

T.C: O(nlogn)

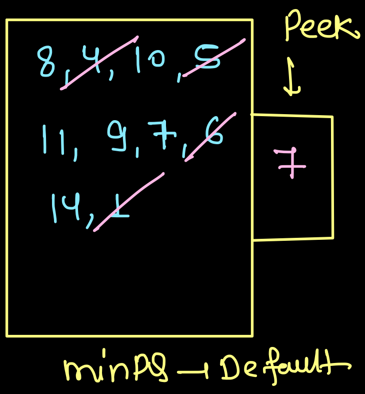
S.C: O(1)

Idea2: Using PQ.

arr →

8	4	10	5	11	9	7	6	14	1
0	1	2	3	4	5	6	7	8	9

, K=5



Removed

i=1 → 1

i=2 → 4

i=3 → 5

i=4 → 6

i=5 loop stops.

peek = 7 → Kth Smallest elmt

Steps:

* Add all elemnt of array in PQ.

* Remove K-1 elemnt from PQ.

* peek elemnt is Kth smallest elemnt in arry.

Time Complexity = $\underbrace{n\log n}_{\text{Insertion}} + \underbrace{k\log n}_{\text{Removal}}$

it is (k-1) logn

= O(nlogn)

Space Complexity : O(n)

Improvised Version:

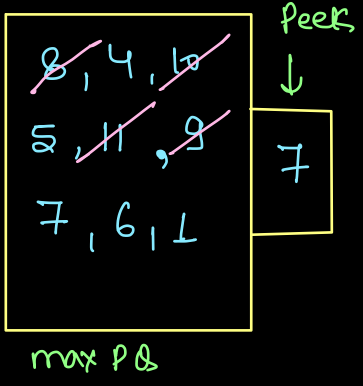
Idea3: Solve It using maxPQ.[do not add more than K elemnt in PQ]

arr →

8	4	10	5	11	9	7	6	14	1
0	1	2	3	4	5	6	7	8	9

K=5

i



Step:

* Add first K element in PQ.

```
i=k to i<n
* if( pq.peek() > arr[i]){
      pq.remove();
      pq.add(arr[i]);
  }
```

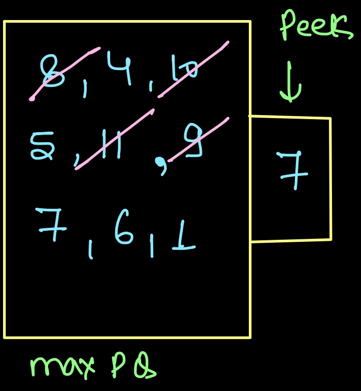
* In end, peek() is Kth Smallest elem

arr →

8	4	10	5	11	9	7	6	14	1
0	1	2	3	4	5	6	7	8	9

K=5

i



Step:

* Add first K element in PQ.

```
i=K to      * if( pq.peek() > arr[i]){
i<n                 pq.remove();
                    pq.add(arr[i]);
              }
```

* In end, peek() is kth Smallest ele

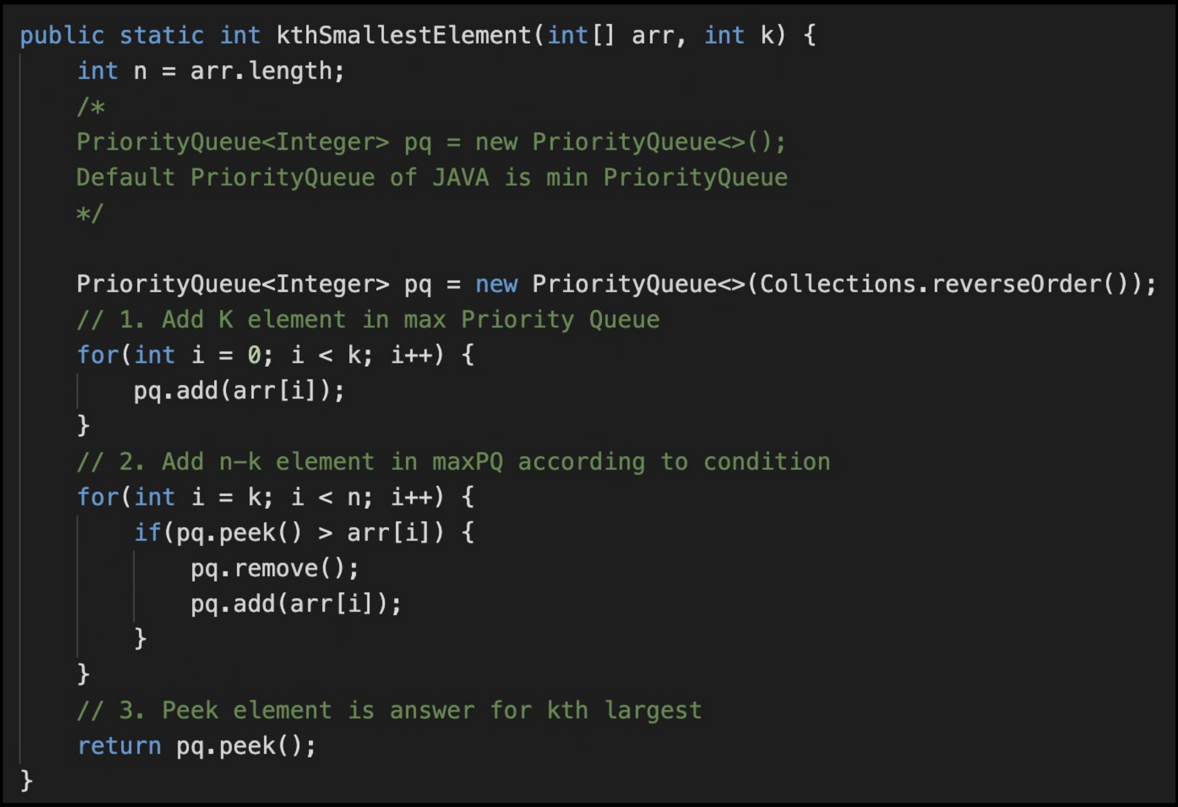
$$T.C: = K\log K + (n-K)\log K + O(1)$$

$$= \cancel{K\log K} + n\log K - \cancel{K\log K} + \text{Const.}$$

$$= n\log K + \text{Const.}$$

T.C. $O(n\log K)$

S.C: $O(K)$



```java
public static int kthSmallestElement(int[] arr, int k) {
    int n = arr.length;
    /*
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    Default PriorityQueue of JAVA is min PriorityQueue
    */

    PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
    // 1. Add K element in max Priority Queue
    for(int i = 0; i < k; i++) {
        pq.add(arr[i]);
    }
    // 2. Add n-k element in maxPQ according to condition
    for(int i = k; i < n; i++) {
        if(pq.peek() > arr[i]) {
            pq.remove();
            pq.add(arr[i]);
        }
    }
    // 3. Peek element is answer for kth largest
    return pq.peek();
}
```

What is medium ?

A:

9	3	7	5	1
0	1	2	3	4

Sort → 1 3 5 7 9 → Middle element → ⑤ is medium

A:

9	3	7	15	1
0	1	2	3	4

Sort → 1 3 7 9 15 → middle elmt → ⑦

A:

9	3	7	5	1	10
0	1	2	3	4	5

Sort → $1 \quad 3 \quad \underbrace{5 \quad 7} \quad 9 \quad 10$

two middle [even number of elmt in array]

It depends on problem statement:

✱ first mid is medium → ⑤

✱ second mid is medium → ⑦

✱ Avg of both mid is medium → $\frac{5+7}{2}$

$= \frac{12}{2} =$ ⑥ [illegible]

$\underbrace{\text{10:13 pm – 10:25 pm}}_{\text{Break time.}}$

Running Median

Median of each prefix subarray. for even: Avg.

Given an array A[], at every index find out the median till now.

A:

9	6	3	10	4
0	1	2	3	4
↓	↓	↓	↓	↓
9	7.5	6	7.5	6

ans:

median

9 → 9

9,6 → 6,9 → 7.5

9,6,3 → 3,6,9 → 6

9,6,3,10 → 3,6,9,10 → 7.5

9,6,3,10,4 → 3,4,6,9,10 → 6

Idea: Every time at every index, catch that subarray from 0 to i and sort that subarray, middle element is median element.

Required complexity to sort subarray. no. of subarray.

$$T.C = n\log n * n$$
$$= O(n^2 \log n)$$

Allowed T.C : O(nlogn)

→ that means in every iteration on element of array, we need to calculate median in logn T.C.

division: * all the value in left PQ < all the element in Right PQ.

* We will try to balance their size

array:

5	3	23	11	20	25	17	7
0	1	2	3	4	5	6	7
5	4	5	8	11	15	17	14

~~5~~, 3, 5
~~11~~, 11, ~~17~~
~~25~~, 7

left → max

~~5~~, 23, ~~11~~
20, 25, ~~17~~
17

Right → min

Scenerio: Even number of elemt:

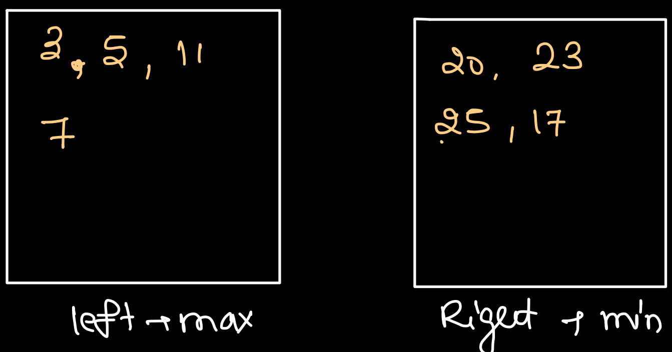
- * add elemt in left PQ.
- * remove highest priority elemt from left & add it in Right

* medium = $\frac{left.peek() + right.peek()}{2}$

Odd number of elmt:

- * Add element in right PQ.
- * Remove highest priority from right & add it in the left
- * peek of left is medion.

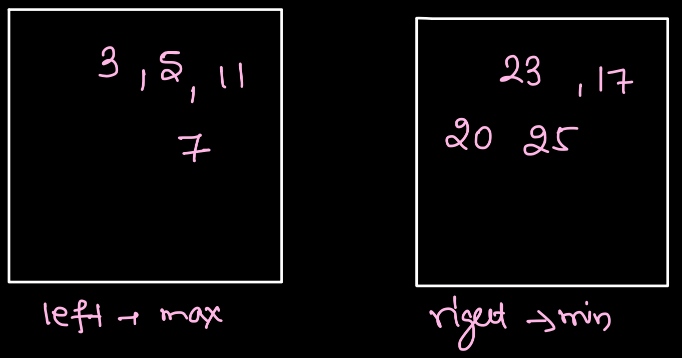
	5	4	5	8	11	15	17	14
array:	5	3	23	11	20	25	17	7
	0	1	2	3	4	5	6	7



```
if( left.size() == right.size()){
    // odd number is comig.
    right.add(arr[i]);
    left.add(right.remove());
    SOPln( left.peek());
}
else {
    // even number is comig
    left.add (arr[i]);
    right.add (left.remove());
    SOPln( (left.peek() + right()) / 2 )
}
```

	5	4	5	8	11	15	17	14
array:	5	3	23	11	20	25	17	7
	0	1	2	3	4	5	6	7

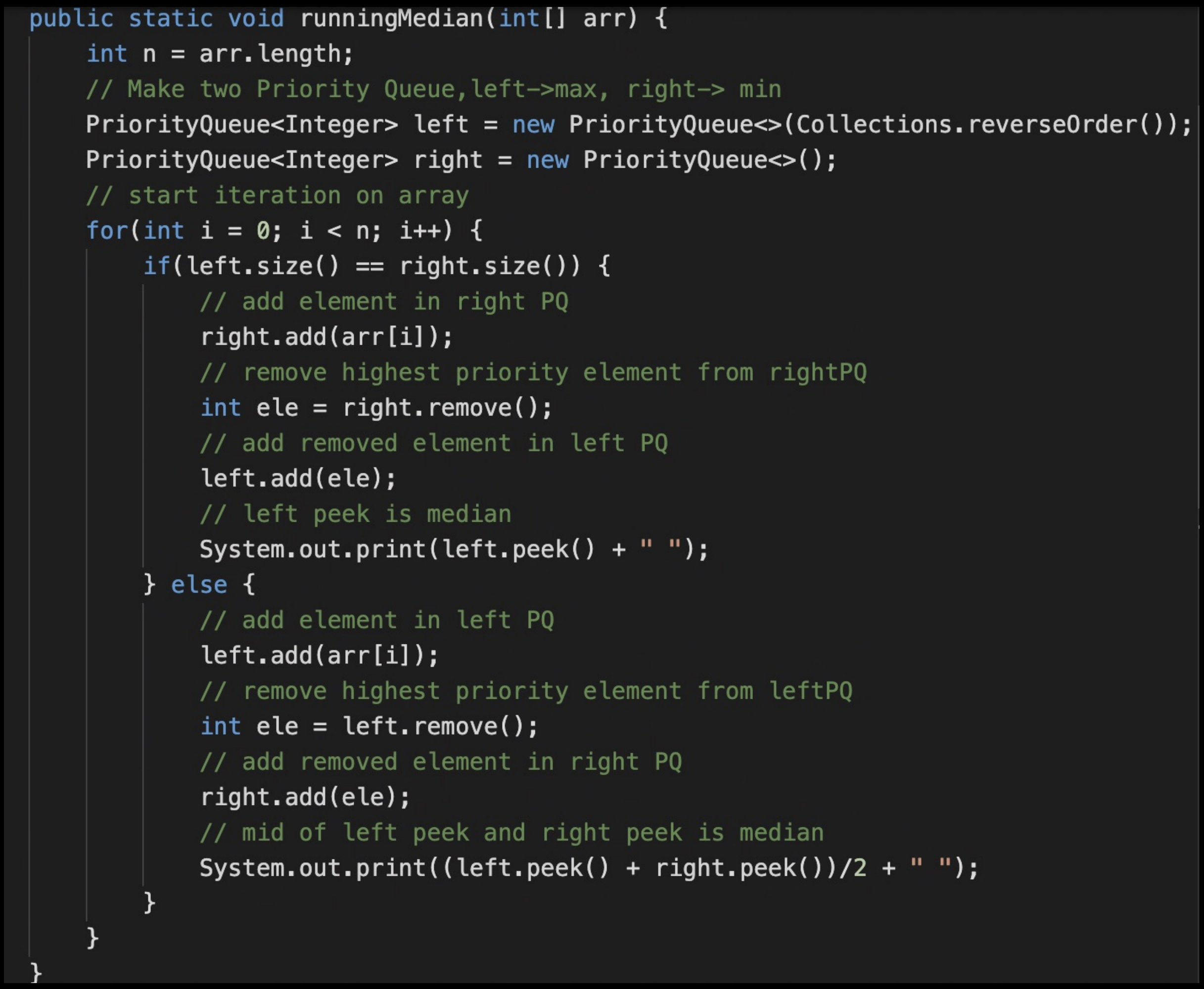
↑
i



```
if(left.size()== right.size()){
    // odd number is comig.
    right.add(arr[i]);
    left.add(right.remove());
    SOPln(left.peek());
}
else {
    // even number is comig
    left.add(arr[i]);
    right.add(left.remove());
    SOPln( (left.peek()+right.peek())/2)
}
```

T.C: O(nlogn)

S.C: O(n)

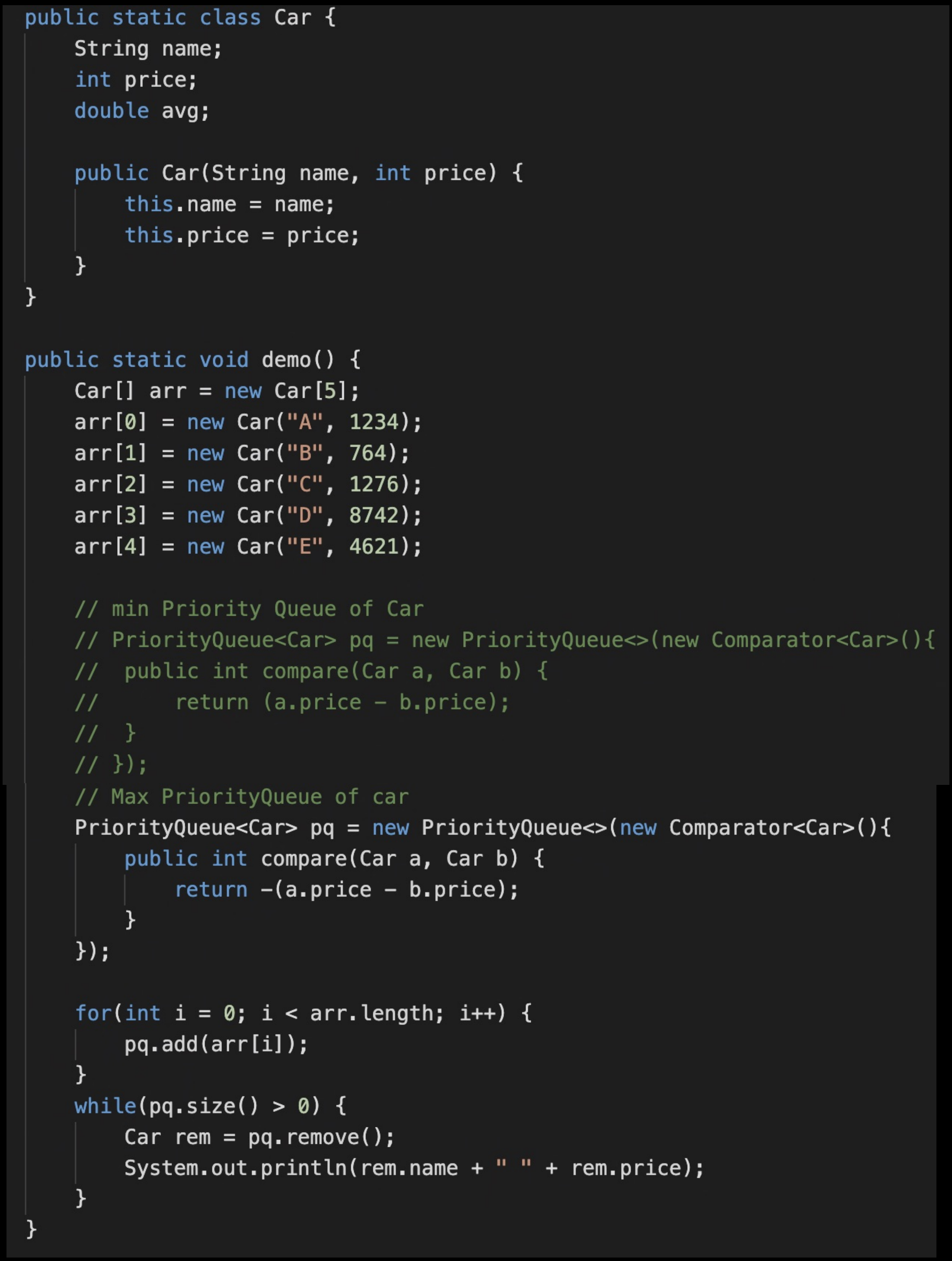


```
public static void runningMedian(int[] arr) {
    int n = arr.length;
    // Make two Priority Queue,left->max, right-> min
    PriorityQueue<Integer> left = new PriorityQueue<>(Collections.reverseOrder());
    PriorityQueue<Integer> right = new PriorityQueue<>();
    // start iteration on array
    for(int i = 0; i < n; i++) {
        if(left.size() == right.size()) {
            // add element in right PQ
            right.add(arr[i]);
            // remove highest priority element from rightPQ
            int ele = right.remove();
            // add removed element in left PQ
            left.add(ele);
            // left peek is median
            System.out.print(left.peek() + " ");
        } else {
            // add element in left PQ
            left.add(arr[i]);
            // remove highest priority element from leftPQ
            int ele = left.remove();
            // add removed element in right PQ
            right.add(ele);
            // mid of left peek and right peek is median
            System.out.print((left.peek() + right.peek())/2 + " ");
        }
    }
}
```

Contest on 29th December

→ Contest 5

Syllabus → Stack, Queues and Tres.



```java
public static class Car {
    String name;
    int price;
    double avg;

    public Car(String name, int price) {
        this.name = name;
        this.price = price;
    }
}

public static void demo() {
    Car[] arr = new Car[5];
    arr[0] = new Car("A", 1234);
    arr[1] = new Car("B", 764);
    arr[2] = new Car("C", 1276);
    arr[3] = new Car("D", 8742);
    arr[4] = new Car("E", 4621);

    // min Priority Queue of Car
    // PriorityQueue<Car> pq = new PriorityQueue<>(new Comparator<Car>(){
    //  public int compare(Car a, Car b) {
    //      return (a.price - b.price);
    //  }
    // });
    // Max PriorityQueue of car
    PriorityQueue<Car> pq = new PriorityQueue<>(new Comparator<Car>(){
        public int compare(Car a, Car b) {
            return -(a.price - b.price);
        }
    });

    for(int i = 0; i < arr.length; i++) {
        pq.add(arr[i]);
    }
    while(pq.size() > 0) {
        Car rem = pq.remove();
        System.out.println(rem.name + " " + rem.price);
    }
}
```