Challenge 1: Using only loops and the methods on PriorityQueue, implement a sorting algorithm that has O(n * lg(n)) performance.

```
void pqSort(int[] arr) {
    PriorityQueue<              > pq = new PriorityQueue<>(Integer::compare);









}
```

Challenge 2

- poll() removes and returns the max/min element from a PriorityQueue
- peek() returns (without removing) the max/min element from a PriorityQueue
- Using Integer::compare as the comparator for Java's default PQ makes a MIN heap

First, try describing what the code does in your own words. Consider adding a sequence of numbers and thinking about pq1, pq2!

```
class _______________Tracker {
    PriorityQueue<Integer> pq1 = new PriorityQueue<>(Collections.reverseOrder(Integer::compare));
    PriorityQueue<Integer> pq2 = new PriorityQueue<>(Integer::compare);
    void add(int n) {
        if(pq2.size() == 0 && pq1.size() == 0) {
            pq2.add(n);
            return;
        }
        int current = get();
        if(n >= current) {
            pq2.add(n);
        }
        else {
            pq1.add(n);
        }
        int sizeDifference = pq2.size()                    -   pq1.size();
        if(sizeDifference > 1) { pq1.add(pq2.poll()); }
        else if(sizeDifference <                - 1) { pq2.add(pq1.poll()); }
    }

    int get() {
        if(pq2.size() == pq1.size()) { return (pq2.peek() + pq1.peek()) / 2; }
        if(pq2.size() > pq1.size()) { return pq2.peek(); }
        else { return pq1.peek(); }
    }

    public String toString() {
        return "" + pq1 + " " + this.get() + " " + pq2;
    }
}
```

Consider calling add with values 1, 7, 5, 10, 3. After:

What will the **size** of pq1, pq2 be?

A:1, 4 B: 5, 5 C: 2, 3 D: 3, 2 E: 3, 3

What will be the **result** of get()?
A: 1 B: 3 C: 5 D: 10 E: 3