EXERCISE 1: Streaming Median

Assume that your application receives a stream of data and that it should at any point be able to report what the *median* of the data received so far is. Design a data type that can support such median queries on data streams efficiently.

```
public class StreamingMedian<ItemType extends Comparable<ItemType> >
{
      public StreamingMedian() {...}
      public void insert(ItemType key) {...}
      public ItemType median() {...}
}
```

Note. The median of a set of elements is the middle element when the elements are considered in sorted order. If there is an even number of elements, the median is the smaller of the two middle elements. For example, the median of `{1, 2, 3}` is `2` and of `{1, 2, 3, 4}` is `2`.

(a) Describe an implementation that can support the `insert()` and `median()` operations in time that is ***at most linear*** (per operation) in the number of elements seen so far (n).

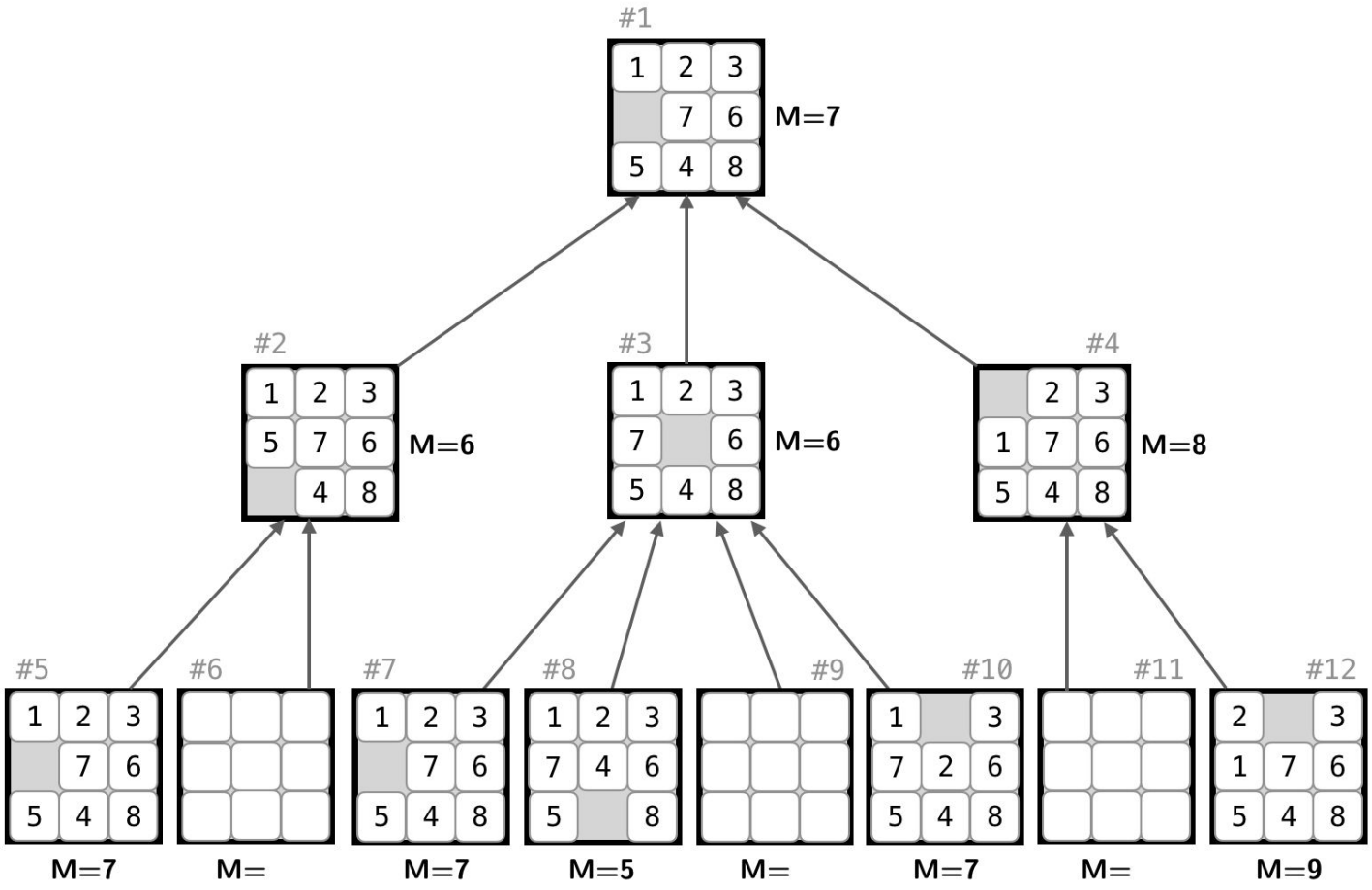
(b) Describe an implementation that can support the `insert()` and `median()` operations in time that is ***at most logarithmic*** (per operation) in the number of elements seen so far (n). An amortized bound is fine.

(*HINT: Use two priority queues!*)

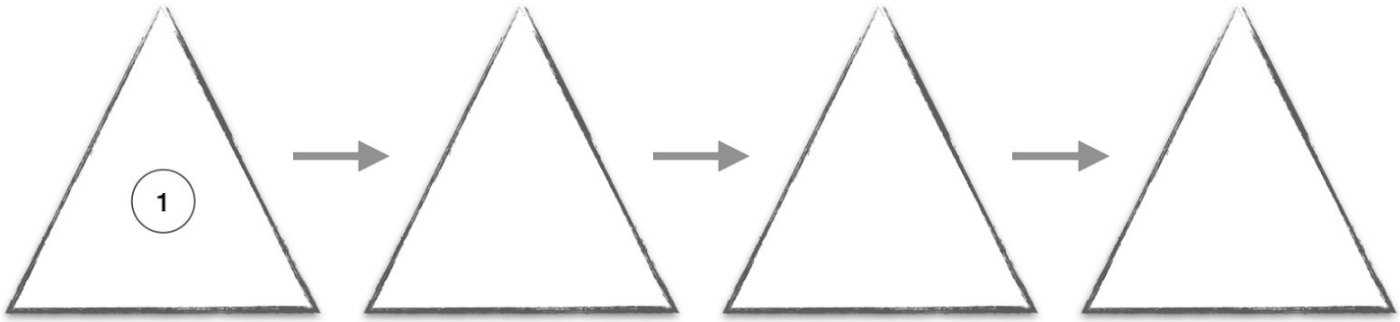
EXERCISE 2: 8-Puzzle

(a) Complete the following diagram for the neighbors of an 8-puzzle board.

Reminder: The *Manhattan distance* between any board and the goal board is the sum of the vertical and horizontal distances of each tile to its goal position. (**M**=Manhattan Distance)



(b) Assuming that you start at board #1, show the contents of the priority queue after each iteration of removing a board and inserting its neighbors.



EXERCISE 3: Running Time Analysis

Suppose we have the following code which uses a binary-heap based minimum priority queue (MinPQ). Assume that $N > k$, and that `a[]` is an array containing random integers.

```
1  void foo(int k, int[] a) {
2        MinPQ<Integer> pq = new MinPQ<Integer>();
3        int N = a.length;
4
5        for (int i = 0; i < N; i++) {
6              pq.insert(a[i]);
7              if (pq.size() > k) pq.delMin();
8        }
9
10       for (int i = 0; i < k; i++)
11             System.out.println(pq.delMin());
12 }
```

(a) Describe what the code outputs in terms of the array `a[]` and the parameter `k`.

(b) What is the order of growth of the running time of the code as a function of both N and k?

(c) Suppose we were to remove line `7`, what would the code output? What would the order of growth be?