EXERCISE 1: Streaming Median (Midterm Spring '18)

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<Item> > {
      public StreamingMedian() {...}
      public void insert(Item 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`.

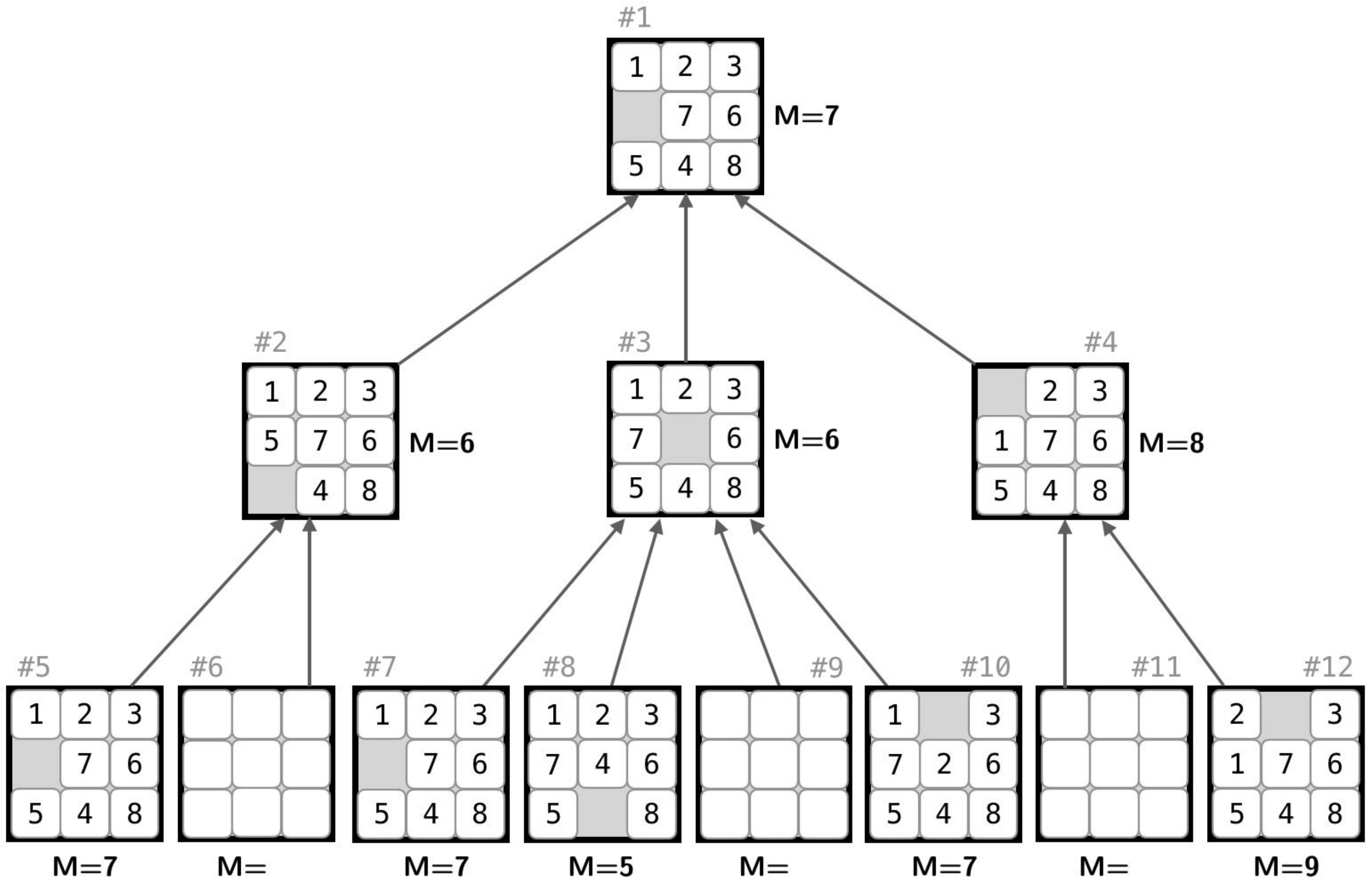
(a) Describe an implementation that can support the `insert()` and `median()` operations in $O(n)$ time, where n is the number of elements seen so far. Mention whether your analysis of the running time is *worst-case*, *amortized* or *probabilistic*.

(b) Describe an implementation that can support the `insert()` and `median()` operations in $O(\log n)$ time, where n is the number of elements seen so far. An amortized bound is fine.

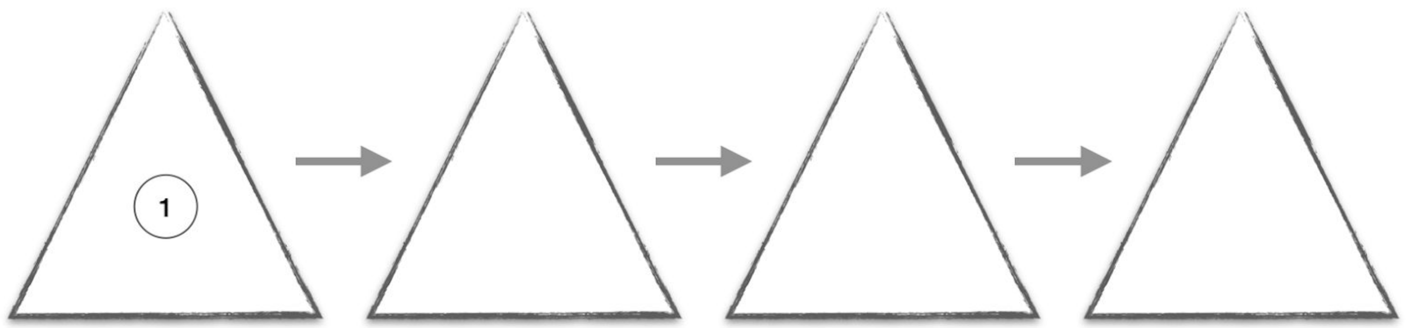
EXERCISE 2: 8-Puzzle

(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 of the first 4 iterations of removing a board and inserting its neighbors using the A* algorithm.



Assignment Tips (from the Checklist).

- **Can I use the expression a == b to check whether two arrays `a[]` and `b[]` are equal?**
  No. That expression checks the two arrays for reference equality (and not whether the two arrays contain the same sequence of values).

- **Can I call `Arrays.equals(a, b)` to check whether two arrays `a[]` and `b[]` are equal? It depends.**
  If `a[]` and `b[]` are of type `int[]`, then `Arrays.equals()` works as expected. If `a[]` and `b[]` are of type `int[][]`, then use `Arrays.deepEquals()`.

- **How do I implement equals()?**
  Java has some arcane rules for implementing `equals()`, discussed on p. 103 of *Algorithms, 4th edition*. Note that the argument to `equals()` is required to be of type Object. For online examples, see Date.java or Transaction.java.

- **How do I return an `Iterable<Board>`?**
  Add the items you want to a `Stack<Board>` or `Queue<Board>` and return that. For example, see the `keys()` method in ArrayST.java. The client should not depend on whether the iterable returned is a stack or queue (because it could be any iterable).

- **I'm getting the right number of moves for puzzle04.txt but the wrong number of moves for some other puzzles. How might I identify what is going wrong?**
  Check the detailed trace for `puzzle04.txt` in the testing section of the Checklist. Even if your program returns the *correct number of moves* for this puzzle, the trace might reveal that you are adding/removing boards in the *wrong order*.

- **I run out of memory when running some of the large sample puzzles. What should I do?**
  Be sure to ask Java for additional memory, e.g.:

  ```
  java-algs4 -Xmx1600m PuzzleChecker puzzle36.txt
  ```

  If your program is unable to solve certain instances, document that in your `readme.txt` file. You should expect to run out of memory when using the Hamming priority function. Be sure not to put the JVM option in the wrong spot or it will be treated as a command-line argument, e.g.:

  ```
  java-algs4 PuzzleChecker -Xmx1600m puzzle36.txt.
  ```

- **My program is too slow to solve some of the large sample puzzles, even if given a huge amount of memory. Is this okay?**
  You should not expect to solve many of the larger puzzles with the Hamming priority function. However, you should be able to solve most (but not all) of the larger puzzles with the Manhattan priority function.