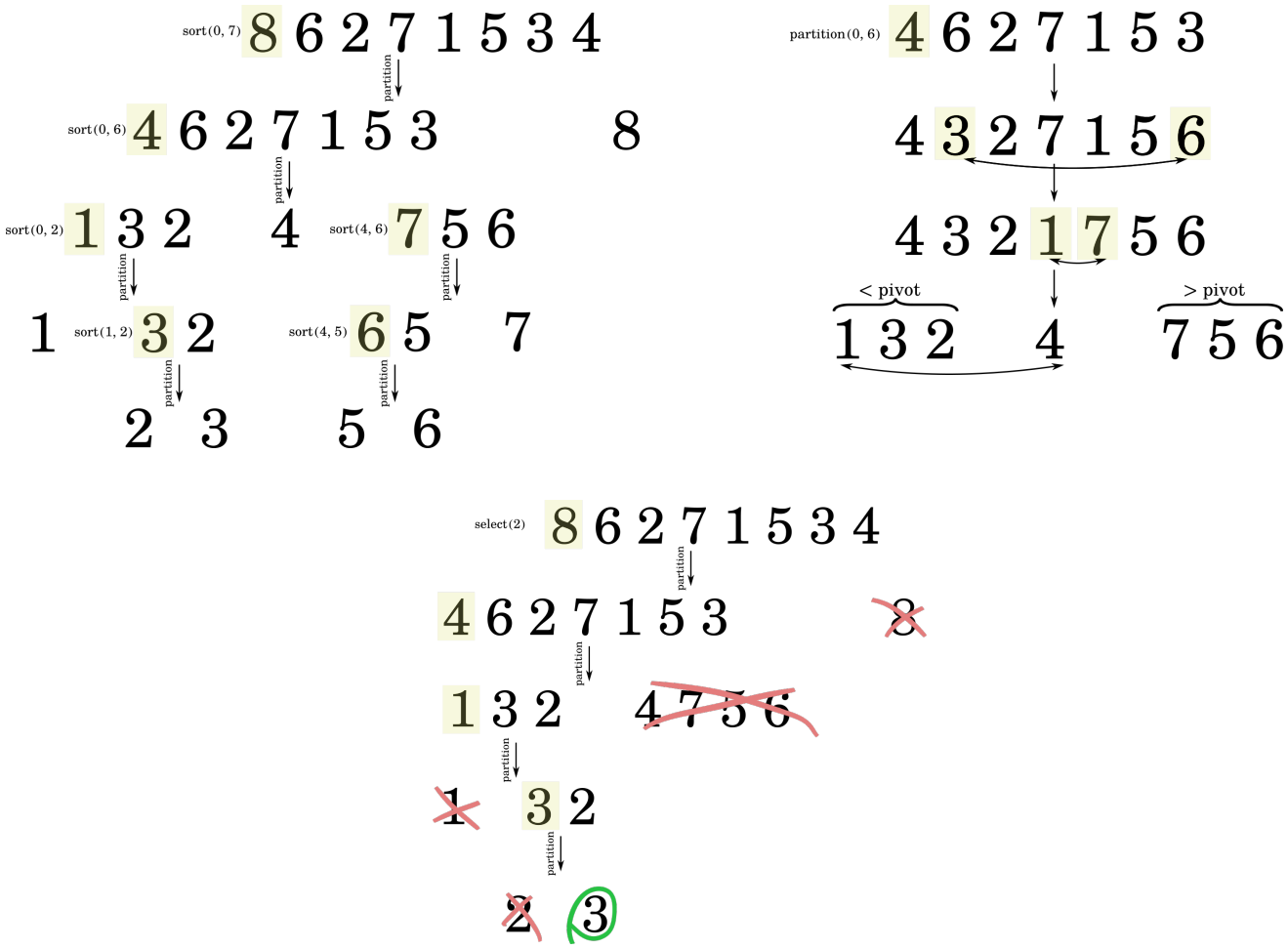
Precept Outline
- Review of Lectures 7 and 8:
  - Quicksort
  - Heaps and Priority Queues

Relevant Book Sections
- Book chapters: 2.3, 2.4 and 2.5

A. Review: Quicksort + Heaps

Your preceptor will briefly review key points of this week's lectures. Here are some images representing examples they will show you, for partition, quicksort and quickselect.



B. Runtime of Priority Queue

Consider the following code which uses a binary-heap based **minimum priority queue** (MinPQ). Assume that $n \geq k$, and that `a[]` is an array containing arbitrary 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 }
```

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

What is the order of growth of the running time of the code as a function of both n and k?

Suppose we were to remove line 7. What would the code's output and order of growth be?

C. Designing a Data Type Using a Priority Queue

This problem was taken and slightly adapted from the Fall 2019 Midterm exam

Design a data type to implement a *double-ended priority queue*. The data type must support inserting a key, deleting a smallest key, and deleting a largest key. (If there are ties for the smallest or largest key, you may choose among them arbitrarily.)

To do so, create a `MinMaxPQ` data type that implements the following API:

```
public class MinMaxPQ<Key extends Comparable<Key>> {
    MinMaxPQ()                                  // create an empty priority queue
    void insert(Key x)                          // add x to the priority queue
    Key min()                                   // return a smallest key
    Key max()                                   // return a largest key
    Key delMin()                                // return and remove a smallest key
    Key delMax()                                // return and remove a largest key
}
```

Here are the performance requirements:

- The `insert()`, `delMin()`, and `delMax()` must take time proportional to $\log n$ or better in the worst case, where n is the number of keys in the priority queue.
- The `min()` and `max()` methods must take constant time in the worst case.

In your answer mention: the instance variables you'll use, your implementation of `min()`/`max()`, your implementation of `insert(x)` and your implementation of `delMin()`/`delMax()`.

Notes: To describe your solution, use either English prose or Java code (or a combination of the two). If your solution uses an algorithm or data structure from the course, do not reinvent it; simply describe how you are applying it.