# Homework 1 Writeup

In this homework we will look at several ways to implement a function, kth smallest. Given an array and an integer k, the function should return the kth smallest element in the array.

You will implement this function using 3 different methods:

1) mergesort
2) quickselect
3) min heaps

Your goal will be to correctly implement the methods mergesort() and quickselect(), as well as a Heap class.

Note: Make sure you fully understand how each algorithm works before you implement it! It is highly recommended to rewatch some of the example animations of each algorithm, and that you read through and use the pseudocode from the notes in your implementations to get the most out of this assignment!

Note 2: In general, we won't be testing for illegal inputs and actions throughout your homeworks (null input, empty array, extracting from an empty heap, etc). While this is important from a programming perspective, the main focus of these homeworks will be on the algorithms themselves.

**Part 1: Understanding**

Your first goal is to read and understand the code that we've given you. There are 3 files in total:

1) KthSmallest. This is the main file of your homework. It contains 3 static methods, which all return the kth smallest element of an array, using the 3 different implementations that you will be writing. Read these functions carefully- we've written for you how the functions that you'll be writing will be used to implement KthSmallest!

2) Sorting. This file contains two static methods, mergeSort(CompareInt[] arr) and quickSelect(int k, CompareInt[] arr). You will be implementing both of these functions, which are used by KthSmallest. Using helper functions is highly encouraged!

3) MinHeap. This file is used by your third implementation of the kthSmallest() function. You will have to implement the minHeap's add() and extractMin() functions.

**Part 2: Mergesort**

The first implementation of kthSmallest() uses mergesort. In the Sorting class, implement the mergeSort(CompareInt[] arr) method. Don't forget to refer to the slides for helpful pseudocode!

NOTE: Rather than finding the kth smallest element on an array of primitive ints, you will be using the provided CompareInt class. The stubs in the homework that we've provided for you all use this class: for example, you will see CompareInt[] as the input array, rather than the usual int[].

You can treat CompareInts more or less as normal integers. The only major change in your code will be how you compare two CompareInts. For example, rather than writing a statement as follows:

```
int i;
int j;

if (i < j) {
        ...
} else if (i > j) {
        ...
} else {
        //i == j
}
```

You will write the following:

```
CompareInt i;
CompareInt j;

if (i.compareTo(j) < 0) {
        ...
} else if (i.compareTo(j) > 0) {
        ...
} else {
        //i and j have the same value
}
```

If you haven't seen the compareTo method before, you can find the javadocs for the comparable interface here:

https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html

In general, i.compareTo(j) will return a negative number if i < j, a positive number if i > j, and 0 if the two are equal.

**Part 3: Quickselect**

The second implementation uses quickselect. Also in the Sorting class, implement the quickSelect(int k, CompareInt[] arr) method. A few tips:

-Remember to look at the pseudocode from the slides!
-Don't forget that arrays are 0-indexed. However, the int k is not (selecting k=1 means select the smallest element in the array).

**Part 4: Heaps**

The final implementation of kthSmallest() uses a minHeap. First, go back to KthSmallest and reread the function that uses the min heap. Next, go to the MinHeap class and take a look at the parts that we've already given to you:

constructor:

minHeap(int n) - In this heap implementation, we've chosen to pass in the maximum number of elements that our heap can contain. This will simplify your implementation, as you don't have to worry about resizing the internal array that we're storing the data in.

field variables:

CompareInt[] heap - We will be storing our heap internal in this array. There are a few things to remember. To simplify the way we do indexing for our heap, we will 1-index our heap. This means that we'll use an array of size n+1, and ignore the 0th element of the array (heap[0]). I.e. the first element to be inserted into the heap should go into heap[1].

int size - We need to keep track of the number of elements in our heap. Notice that this is completely different from heap.size (the "capacity", or maximum number of elements that we can store in the heap).

Finally, you need to implement the minHeap's two main functions:

add(int val) - adds "val" to the heap. We've asked you to throw an IllegalArgumentException() if the heap is already at maximum capacity (it already has heap.size - 1 elements in it).

extractMin() - removes the smallest element from the heap. Some tips for both functions:

-What should happen to size when we add/extract an element?
-don't forget to use helper functions here! swim() and sink() are used in the lecture slides

NOTE: This implementation of building the heap inserts each of the n elements one by one into a heap, requiring O(nlgn) time. There is actually a better implementation that can build a heap directly from an array in linear time (O(n)), but it was not covered in the lecture recordings.