# More Sorting

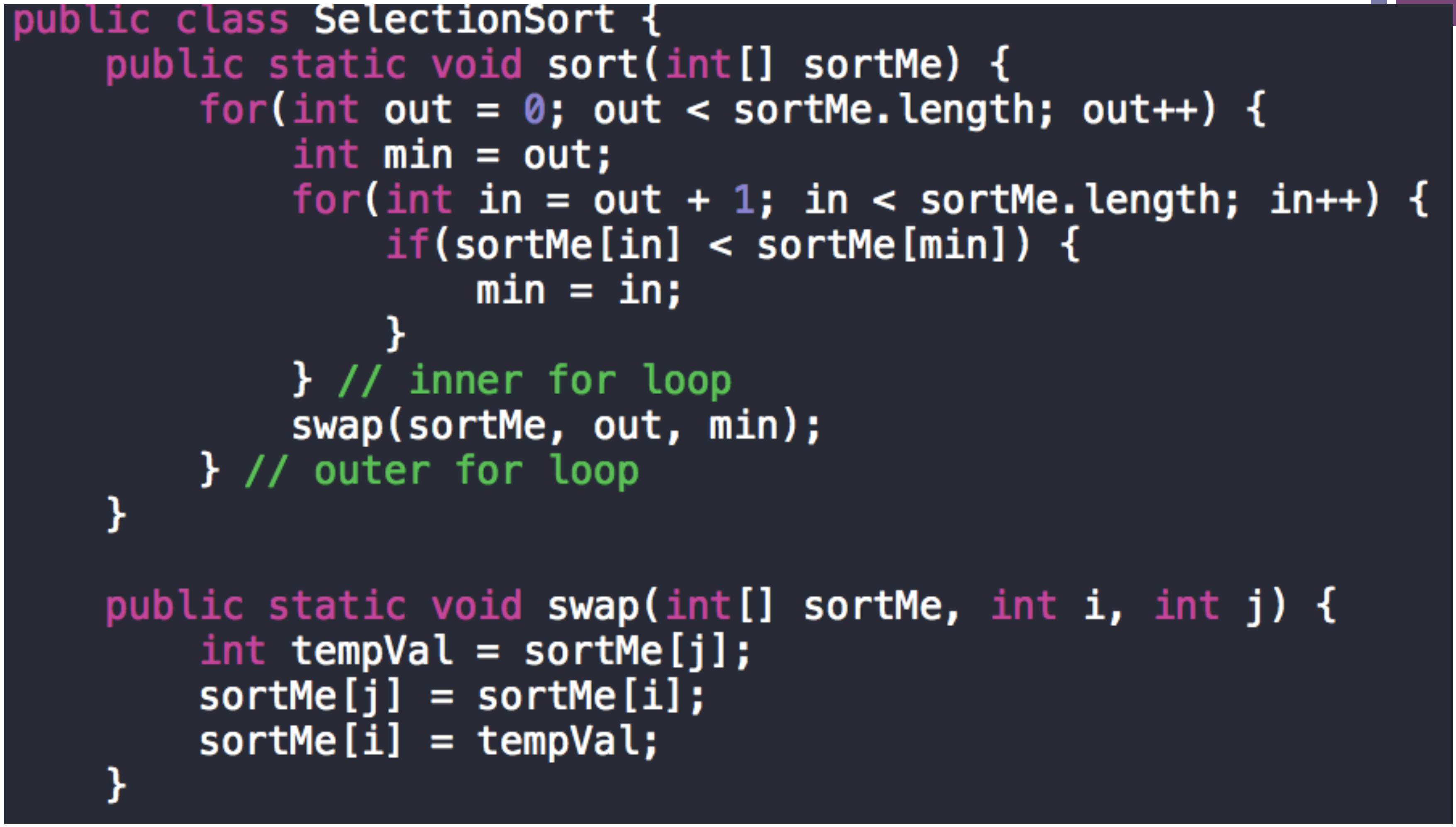
+ 
## Sorting…

- Why sort?
  - To make searching faster!
    - How?
      - Binary Search gives log(n) performance.
- There are many algorithms for sorting: bubble sort, selection sort, insertion sort, quick sort, heap sort, …
- Why so many?

## Selection Sort

- Basic idea:
  - step forward on each item of the array starting with the first item, if there is a smaller item in front of the item being stepped on, then swap the two items. Repeat until you've stepped on every item.
- Implementation:
  - nested loop
    - first loop marks the current item
      - inner loop finds the smallest item between the current item and the last item inclusively, then swaps the items
- Time Complexity?

+ Selection Sort

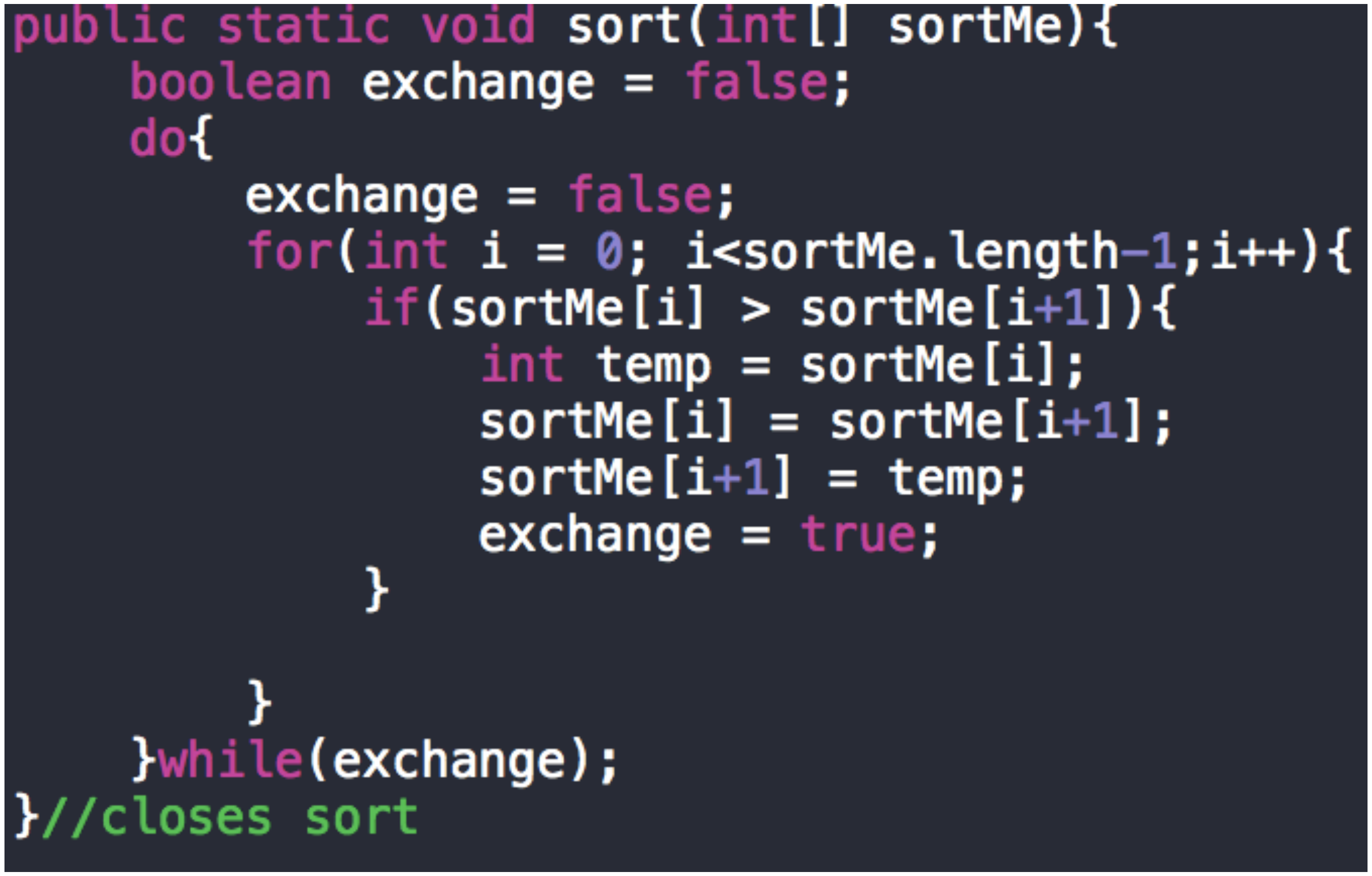

```
public class SelectionSort {
    public static void sort(int[] sortMe) {
        for(int out = 0; out < sortMe.length; out++) {
            int min = out;
            for(int in = out + 1; in < sortMe.length; in++) {
                if(sortMe[in] < sortMe[min]) {
                    min = in;
                }
            } // inner for loop
            swap(sortMe, out, min);
        } // outer for loop
    }

    public static void swap(int[] sortMe, int i, int j) {
        int tempVal = sortMe[j];
        sortMe[j] = sortMe[i];
        sortMe[i] = tempVal;
    }
```

## Bubble sort

- Basic idea:
  - start with the first item in the array compare adjacent items if they are not sorted, swap them, go to the next item and repeat until you get to the end.
  - repeat the above process until sorted
- Implementation:
  - nested loop
    - first loop checks if the array is sorted
      - inner compares and swaps
- Time Complexity?

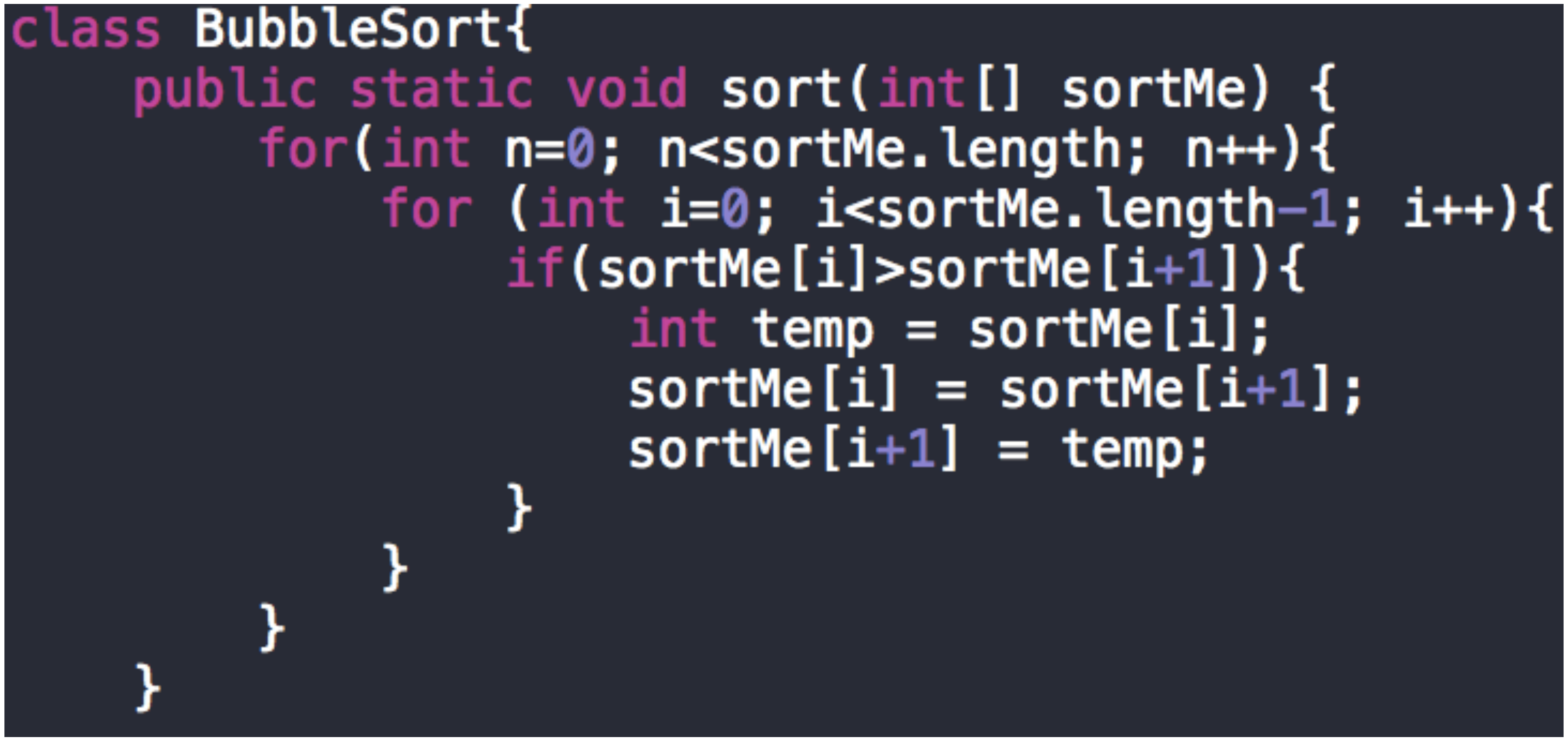
## Bubble Sort



```
public static void sort(int[] sortMe){
    boolean exchange = false;
    do{
        exchange = false;
        for(int i = 0; i<sortMe.length-1;i++){
            if(sortMe[i] > sortMe[i+1]){
                int temp = sortMe[i];
                sortMe[i] = sortMe[i+1];
                sortMe[i+1] = temp;
                exchange = true;
            }

        }
    }while(exchange);
}//closes sort
```

## Bubble Sort



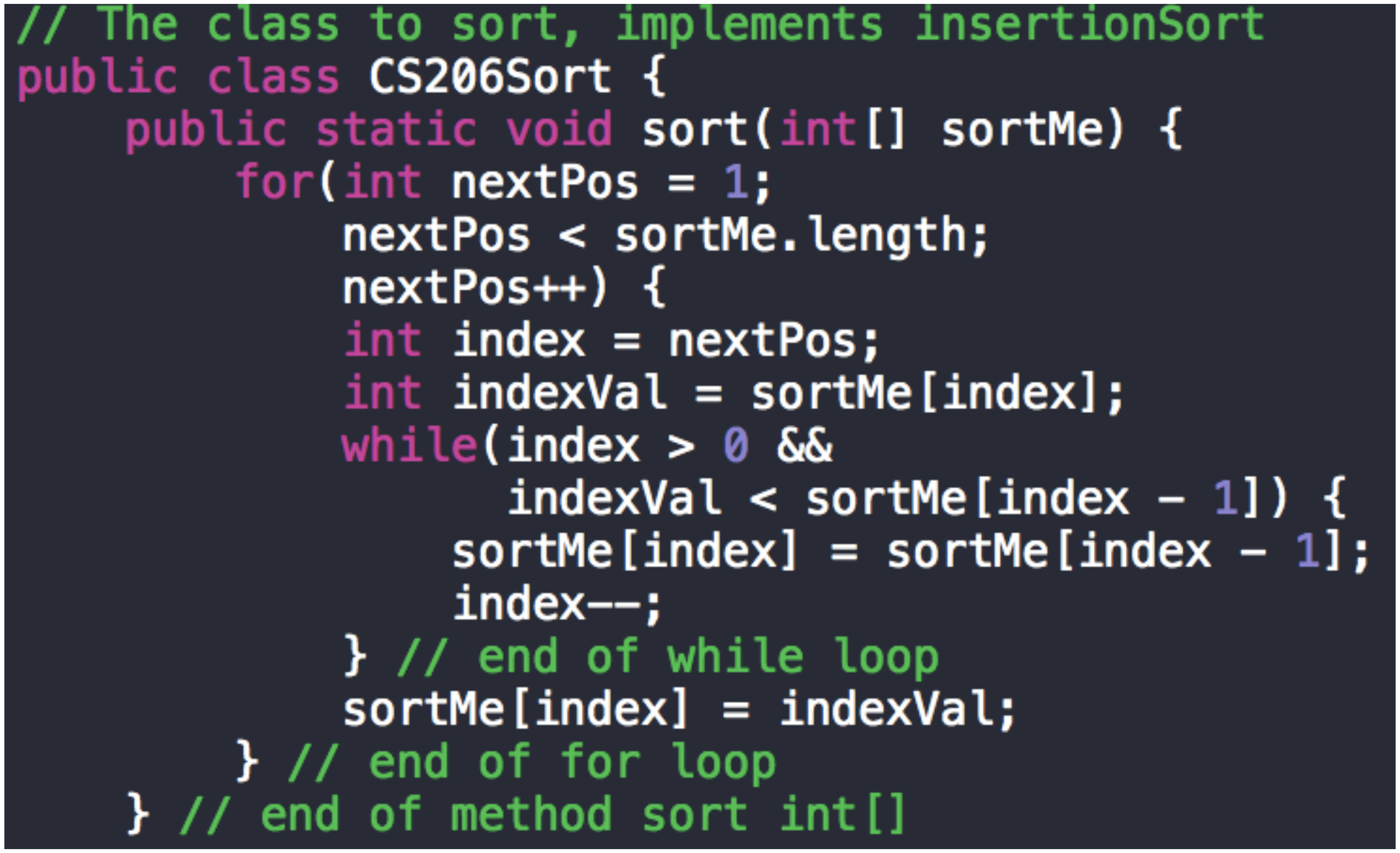
```
class BubbleSort{
    public static void sort(int[] sortMe) {
        for(int n=0; n<sortMe.length; n++){
            for (int i=0; i<sortMe.length-1; i++){
                if(sortMe[i]>sortMe[i+1]){
                    int temp = sortMe[i];
                    sortMe[i] = sortMe[i+1];
                    sortMe[i+1] = temp;
                }
            }
        }
    }
```

## Insertion Sort

- Basic idea:
  - start with a sorted subarray, insert the next item from your unsorted list into the right position of the sorted list.
  - When you get to the end of the unsorted list, you are done
- Implementation:
  - nested loop
    - first loop gets next item to insert
      - inner compares, copies and makes space
      - inserts into space
- Time Complexity?

+

## Insertion Sort



```
// The class to sort, implements insertionSort
public class CS206Sort {
    public static void sort(int[] sortMe) {
        for(int nextPos = 1;
            nextPos < sortMe.length;
            nextPos++) {
            int index = nextPos;
            int indexVal = sortMe[index];
            while(index > 0 &&
                  indexVal < sortMe[index - 1]) {
                sortMe[index] = sortMe[index - 1];
                index--;
            } // end of while loop
            sortMe[index] = indexVal;
        } // end of for loop
    } // end of method sort int[]
```

## Heapsort

- General idea
  - remove items from array one at a time and put them into a heap
  - remove items from heap one at a time and put them back into the array.
- Overall time complexity?
- Overall space needs?

+ 
## Heapsort

- General idea
  - remove items from array one at a time and put them into a heap
    - O(n log n) time
    - n extra space
  - remove items from heap one at a time and put them back into the array.
    - O(n) time
    - (same) n extra space
- Overall time complexity?
- Overall space needs?
  - can we do it with no added space?

# Heapsort (no added space)

- General idea
  - max-heapify the array
  - remove items from heap one at a time and swap them into the end of the heap.
- Overall time complexity?
- Overall space needs?

## Merge Sort

- General idea
  - Split array into 2 parts, left array and right array
    - Merge sort left array
    - Merge sort right array
    - Merge sorted left and  sorted right
  - Divide and Conquer
- Time Complexity?
- Extra Space?

+
## Quicksort

- General idea
  - Recursively partition around a pivot point.
    - Choose a pivot value (typically the first element of the array)
    - partition all elements less than pivot value to the left
    - partition all elements greater than pivot value to the right
    - quicksort sub array to left of pivot point
    - quicksort sub array to right of pivot point
- Time complexity
  - best case O(n log n)
  - worst case $O(n^2)$ – when array is sorted.

## Quicksort

- How do we reduce our time complexity for the worst case?
  - select a better pivot
    - Add step 0: find the median of the start, mid, and last values and swap it with the start value
    - do standard quicksort.