Mergesort

- Our next sorting algorithm is the mergesort
- Consider the selection sort we have already studied:
  - It is simple, but inefficient, especially for large arrays
  - Imagine searching through a pile of 1000 index cards for the lowest item, and when finding it, searching through the remaining 999 cards for the lowest again
    - each card ends up being examined about 500 times
- Mergesort is a "divide-and-conquer" approach
  - Divide the 1000 cards two piles of 500
  - Sort each pile of 500 and then merge the two into a single ordered pile
  - To further simplify, each subpile could be divided and sorted, so on
  - Best implemented **recursively**

Mergesort Pseudocode:

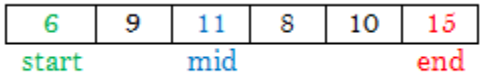
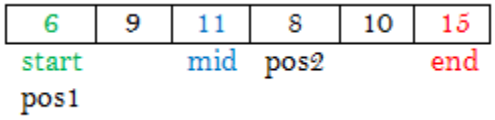
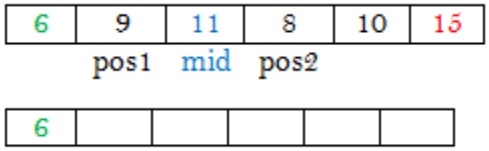
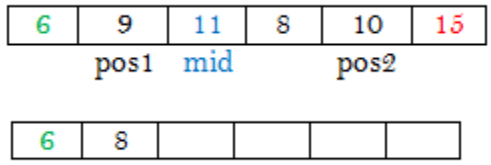
```
if there are items remaining {
      mergesort the left half of the items
      mergesort the right half of the items
      merge the two halves into a completely sorted list
}
```

Mergesort Implementation (calls a merge() method we will look at later)

```
/**
 * Sorts items[start..end]
 * pre: start > 0, end > 0
 * post: items[start..end] is sorted low to high
 */
public static void mergesort(int[] items, int start, int end) {
      if (start < end) {
            int mid = (start + end) / 2;
            mergesort(items, start, mid);
            mergesort(items, mid + 1, end);
            merge(items, start, mid, end);
      }
}
```

Mergesort Illustration

- The merge() method uses a temporary array to store items moved from two sorted portions of the items array
- The elements are moved so that the temporary array is sorted

	Suppose that this is the array at the entry to merge()
	The array is sorted from `start` to `mid` and from `mid`+1 to `end`. The merge() method starts by examining the first element of each sorted portion, `start` and `mid`+1, as indicated by `pos1` and `pos2`.
	Since `items[pos1] < items[pos2]`, the element `items[pos1]` is moved to the new array, and `pos1` is incremented
	In this case, `items[pos1] > items[pos2]`, so the element `items[pos2]` is moved to the new array and `pos2` incremented
	The process is repeated until all items have been moved Finally merge() copies the merged items in the temporary array to the original array.

Full mergesort implementation

merge() method is private because it is a helper method.

```
/**
 * Merges two sorted portion of items array
 * pre: items[start..mid] is sorted. items[mid+1..end] sorted.
 * start <= mid <= end
 * post: items[start..end] is sorted.
 */
private static void merge(int[] items, int start, int mid, int end) {
      int[] temp = new int[items.length];
      int pos1 = start;
      int pos2 = mid + 1;
      int spot = start;

      while (!(pos1 > mid && pos2 > end)) {
            if ((pos1 > mid) || ((pos2 <= end) && (items[pos2] < items[pos1]))) {
                  temp[spot] = items[pos2];
                  pos2 += 1;
            } else {
                  temp[spot] = items[pos1];
                  pos1  += 1;
            }
            spot += 1;
      }
      /* copy values from temp back to items */
      for (int i =  start; i <= end; i++) {
            items[i] = temp[i];
      }

}


/**
 * Sorts items[start..end]
 * pre: start > 0, end > 0
 * post: items[start..end] is sorted low to high
 */
public static void mergesort(int[] items, int start, int end) {
      if (start < end) {
            int mid = (start + end) / 2;
            mergesort(items, start, mid);
            mergesort(items, mid + 1, end);
            merge(items, start, mid, end);
      }
}
```

ICS4U Module 6: Note + Exercise 1d

Client code: TestSorts application has been modified to use the mergesort() method to sort an array of integers

```
import java.util.Scanner;

public class TestSorts {

	public static void displayArray(int[] array){
		for (int i = 0; i< array.length; i++){
			System.out.print(array[i] + " " );

		}
		System.out.println("\n");
	}

	public static void main(String[] args) {
		Scanner input = new Scanner(System.in);
		int numItems;
		int[] test;

		System.out.print("Enter number of elements: ");
		numItems = input.nextInt();

		/* populate array with random integers */
		test = new int[numItems];
		for (int i = 0; i < test.length; i++){
			test[i] = (int)(100 * Math.random());
		}
		System.out.println("Unsorted:");
		displayArray(test);

		Sorts.mergesort(test,0,test.length-1);

		System.out.println("Sorted:");
		displayArray(test);

	}

}
```

Output:

```
Enter number of elements: 10
Unsorted:
47 65 82 0 51 56 72 16 86 85

Sorted:
0 16 47 51 56 65 72 82 85 86
```

Programming Exercise:

Create an ObjectsMergesort application that implements a mergesort on an array of objects. Test the sort on an array of String objects.

Submit your source code to the Google Doc "ICS4U – Activity Submission Form"