Merge Sort

Outline

Merging

Merge sort is based on a simple operation known as merging: combining two ordered arrays to make one larger ordered array

To sort an array, divide it into two halves, sort the two halves (recursively), and then merge the results

Merge sort overview

input	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
sort left half	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
sort right half	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge results	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Merging

The merge algorithm

```
package edu.princeton.cs.algs4;

import java.util.Comparator;

public class Merge {
    private static void merge(Comparable[] a, Comparable[] aux,
                              int lo, int mid, int hi) {
        int i = lo, j = mid + 1;
        for (int k = lo; k <= hi; k++) {
            aux[k] = a[k];
        }
        for (int k = lo; k <= hi; k++) {
            if (i > mid)                    { a[k] = aux[j++]; }
            else if (j > hi)                { a[k] = aux[i++]; }
            else if (less(aux[j], aux[i])) { a[k] = aux[j++]; }
            else                            { a[k] = aux[i++]; }
        }
    }
}
```

Merging

Trace of merge operation

		a[]											aux[]								
	k	0	1	2	3	4	5	6	7	8	9	i	j	0	1	2	3	4	5	6	7	8	9
input		E	E	G	M	R	A	C	E	R	T												
copy														E	E	G	M	R	A	C	E	R	T
												0	5										
	0	A										0	6	E	E	G	M	R	A	C	E	R	T
	1	A	C									0	7	E	E	G	M	R		C	E	R	T
	2	A	C	E								1	7	E	E	G	M	R			E	R	T
	3	A	C	E	E							2	7		E	G	M	R			E	R	T
	4	A	C	E	E	E						2	8			G	M	R			E	R	T
	5	A	C	E	E	E	G					3	8			G	M	R				R	T
	6	A	C	E	E	E	G	M				4	8				M	R				R	T
	7	A	C	E	E	E	G	M	R			5	8					R				R	T
	8	A	C	E	E	E	G	M	R	R		5	9									R	T
	9	A	C	E	E	E	G	M	R	R	T	6	10										T
merged result		A	C	E	E	E	G	M	R	R	T												

Merge Sort

Top-down merge sort

```
package edu.princeton.cs.algs4;

import java.util.Comparator;

public class Merge {
    public static void sort(Comparable[] a) {
        Comparable[] aux = new Comparable[a.length];
        sort(a, aux, 0, a.length - 1);
    }

    private static void sort(Comparable[] a, Comparable[] aux,
                             int lo, int hi) {
        if (hi <= lo) { return; }
        int mid = lo + (hi - lo) / 2;
        sort(a, aux, lo, mid);
        sort(a, aux, mid + 1, hi);
        merge(a, aux, lo, mid, hi);
    }
}
```

Merge Sort

Trace of merge sort

	a[]														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 0, 1, 3)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 4, 4, 5)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 6, 6, 7)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 4, 5, 7)	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E
merge(a, aux, 0, 3, 7)	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
merge(a, aux, 8, 8, 9)	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E
merge(a, aux, 10, 10, 11)	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E
merge(a, aux, 8, 9, 11)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a, aux, 12, 12, 13)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a, aux, 14, 14, 15)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
merge(a, aux, 12, 13, 15)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, aux, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge(a, aux, 0, 7, 15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Merge Sort

Top-down merge sort uses between $1/2N \lg N$ and $N \lg N$ comparisons and at most $6N \lg N$ array accesses to sort an array of length N

Merge sort guarantees to sort an array of N items in time proportional to $N \lg N$, no matter what the input

The prime disadvantage of merge sort is that it uses extra space proportional to N

The `MergeX` variant from the text implements the following improvements

- Handles tiny subarrays using insertion sort
- Reduces the running time to be linear for arrays that are already in order by adding a test to skip the call to `merge()` if `a[mid]` is less than or equal to `a[mid + 1]`
- Eliminates the time (but not the space) taken to copy to the auxiliary array used for merging by using two invocations of the sort method: one that takes its input from the given array and puts the sorted output in the auxiliary array; the other that takes its input from the auxiliary array and puts the sorted output in the given array

Complexity of Sorting

No comparison-based sorting algorithm can guarantee to sort N items with fewer than $\lg N! \sim N \lg N$ comparisons (see p. 280 of the text for a nice proof of this proposition)

Merge sort is an asymptotically optimal comparison-based sorting algorithm, since the number of comparisons used by merge sort in the worst case is $\sim N \lg N$

Merge Sort and Other Sorts

A comparison of merge sort and selection sort

```
$ java SortCompare Merge Selection 10000 100
For 10000 random Doubles
    Merge is 44.8 times faster than Selection
```

A comparison of merge sort and insertion sort

```
$ java SortCompare Merge Insertion 10000 100
For 10000 random Doubles
    Merge is 39.1 times faster than Insertion
```

A comparison of merge sort and shell sort

```
$ java SortCompare Merge Shell 10000 100
For 10000 random Doubles
    Merge is 1.1 times faster than Shell
```

A comparison of merge sort and system sort

```
$ java SortCompare Merge System 10000 100
For 10000 random Doubles
    Merge is 2.0 times faster than System
```