

# **Data Structures and Algorithm Analysis**

## **Lab 6, Merge sort.**

# Contents

- Using mergesort with algs4.
- Implement mergesort.

## Mergesort with algs4

Top-down mergesort:

<https://algs4.cs.princeton.edu/code/javadoc/edu/princeton/cs/algs4/Merge.html>

Bottom-up mergesort:

<https://algs4.cs.princeton.edu/code/javadoc/edu/princeton/cs/algs4/MergeBU.html>

Optimized mergesort:

<https://algs4.cs.princeton.edu/code/javadoc/edu/princeton/cs/algs4/MergeX.html>

## Using mergesort with algs4

Using the Merge class in algs4 library:

```
import edu.princeton.cs.algs4.Merge;
import edu.princeton.cs.algs4.StdOut;

public class TestMergeSort {
    public static void main( String[] args ) {

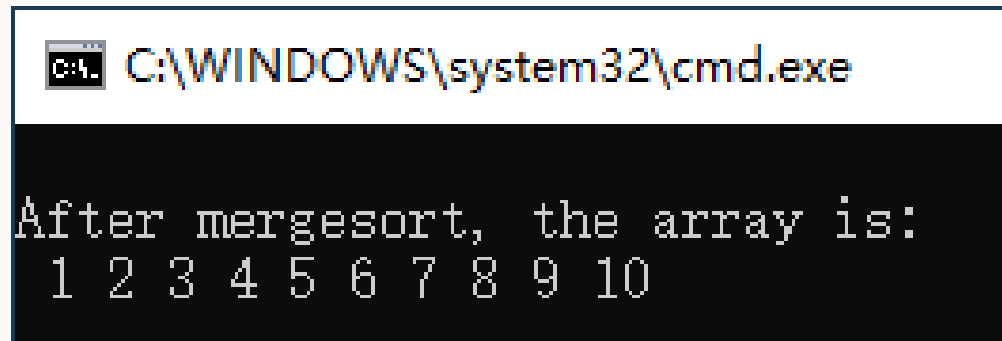
        Integer[] array = new Integer[] { 7, 6, 5, 4, 3, 2, 1,
            10, 9, 8 };

        Merge.sort(array);

        StdOut.println("\nAfter mergesort, the array is:");
        for( Integer i : array ) StdOut.print(" "+i);
        StdOut.println();
    }
}
```

## Using mergesort with algs4

The above code should produce the following result:



```
C:\WINDOWS\system32\cmd.exe

After mergesort, the array is:
1 2 3 4 5 6 7 8 9 10
```

If you want to use MergeX or MergeBU instead of Merge, just replace the class name.

## Using mergesort with algs4

The method we use to do the sorting in Merge is:

```
public static void sort(Comparable[] a)
```

This means we have to use "Integer" array instead of "int" array. One solution is to use a conversion. The other way is to implement your own mergesort.

## Implement mergesort

To implement your own mergesort, first define what you want to have in your class. Write "static" to save the time of initializing new objects.

```
public class MergeSort {  
    public static void sort( int[] array ) {  
    }  
    public static void sort( double[] array ) {  
    }  
}
```

## Implement mergesort

Since we want to implement a recursive mergesort, and we need to use external memory, we define another "sort" method, which specifies the range [low, high] in the signature. The "sort(int[] array)" simply calls that method and specify [0, length-1] as the range.

```
private static void sort( int[] array, int low, int high,
    int[] aux ) {
}
public static void sort( int[] array ) {
    sort(array, 0, array.length-1, new int[array.length]);
}
```



## Implement mergesort

- Divide the array into two parts.
- Sort each part.
- Merge two parts into one.

```
private static void sort( int[] array, int low, int high,
    int[] aux ) {
    int mid = low + (high-low)/2;
    sort( array, low, mid, aux );
    sort( array, mid+1, high, aux );
    merge(array, low, mid, high, aux);
}
```

However, this may cause infinite recursion.

## Implement mergesort

Add extra condition statement to stop recursion. We use insertion sort when the range [low, high] is too small.

```
private static void sort( int[] array, int low, int high,
    int[] aux ) {
    if( high - low < 10 ) {
        insertion(array, low, high);
        return;
    }
    int mid = low + (high-low)/2;
    sort( array, low, mid, aux );
    sort( array, mid+1, high, aux );
    merge(array, low, mid, high, aux);
}
```

## Implement mergesort

Let's implement "merge".

The purpose of "merge" method: given an integer "array", the interval [low, mid], and [mid+1, high] are already sorted, merge the two interval so that [low, high] is sorted.

```
private static void merge( int[] array, int low, int mid,
    int high, int [] aux ) {
}
```

# Implement mergesort

One possible solution.

```
private static void merge( int[] array, int low, int mid,
    int high, int [] aux ) {
    System.arraycopy(array, low, aux, low, high-low+1);
    int i, j1, j2;
    for( i = low, j1=low, j2 = mid+1; j1<=mid && j2 <= high; )
        {
            if( aux[j1] <= aux[j2] )
                array[i++] = aux[j1++];
            else
                array[i++] = aux[j2++];
        }
    while( j1 <= mid )
        array[i++] = aux[j1++];
    while( j2 <= high )
        array[i++] = aux[j2++];
}
```

## Implement mergesort

After that, you still need to write test code to make sure your implementation is correct. You can test your code on random generated data but don't forget to include some special cases:

- When array length is 0, algorithm should change nothing.
- When array is already sorted, algorithm should change nothing.

Try to include some more special cases if you are not confident about your algorithm.

## In class exercise 1: Natural mergesort

Finish exercise 2.2.16 in "Algorithms FOURTH EDITION":

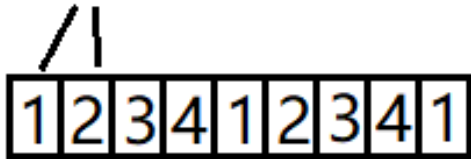
Write a version of bottom-up mergesort that takes advantage of order in the array by proceeding as follows each time it needs to find two arrays to merge:

Find a sorted subarray (by incrementing a pointer until finding an entry that is smaller than its predecessor in the array), then find the next, then merge them. Analyze the running time of this algorithm in terms of the array size and the number of maximal increasing sequences in the array.

## In class exercise 1: Natural mergesort

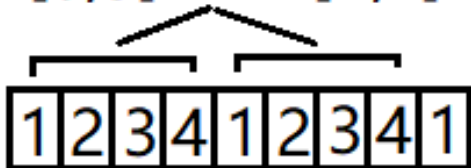
Old buttom up mergesort:

Merge  $[0,0]$  and  $[1,1]$  first, ignoring all array information



You are required to do:

Merge  $[0,3]$  and  $[4,7]$  first, since they are already sorted



## In class exercise 1: natural mergesort

You must test the correctness and performance of the natural merge you have written, using anything you have learned in the previous labs.



## In class exercise 2: Less Copy of Auxiliary Array

Finish exercise 2.2.16 in "Algorithms FOURTH EDITION":

Eliminate the copy to the auxiliary array. It is possible to eliminate the time (but not the space) taken to copy to the auxiliary array used for merging. To do so, we use two invocations of the sort method: one takes its input from the given array and puts the sorted output in the auxiliary array; the other takes its input from the auxiliary array and puts the sorted output in the given array. With this approach, in a bit of recursive trickery, we can arrange the recursive calls such that the computation switches the roles of the input array and the auxiliary array at each level.