Revised: 5/9/2024

Sorting II

Michael C. Hackett

Assistant Professor, Computer Science

COMMUNITY COLLEGE OF PHILADELPHIA

Lecture Topics

- Recursive Sorting
  - Bubble Sort
  - Merge Sort
  - Quicksort

Using Recursion to perform a Bubble Sort

- You’ve seen the bubble sort (and its variants) implemented using an iterative algorithm.

- Since any problem that can be solved iteratively can be solved recursively, we can design a recursive replacement for the iterative bubble sort.

Using Recursion to perform a Bubble Sort

- For an ascending sort, the first pass will move the largest value to the end of the array (length - 1).
  - The next pass will move the second largest value to index length - 2.
  - The next pass will move the third largest value to index length - 3.
  - And so on…

- The last pass of the iterative algorithm sorts for index 0.
  - At this point, the smallest value is guaranteed to already be in index 0.

Using Recursion to perform a Bubble Sort

- The base case is when the algorithm sorts for index 0.
  - An array with a length of 1
  - An array of length 1 implies the value at index 0 is already in the correct position (because it is the only value).

- The recursive case is for sorting an array with a length > 1.

Bubble Sort (Recursive Algorithm)

```
public void bubbleSort(int[] a, int length) {
    if(length == 1) {
        return;
    }

    for(int i = 0; i < length-1; i++) {
        if(a[i] > a[i+1]) {
            int temp = a[i+1];
            a[i+1] = a[i];
            a[i] = temp;
        }
    }

    bubbleSort(a, length-1);
}
```

Calls the method again, but one index less than the index we just sorted for (Similar to what we did for Bubble Sort Improved 2)

Using Recursion to perform a Bubble Sort

- This recursive bubble sort algorithm, like the iterative version, also performs in polynomial time.

- Let’s see if there is any difference in the number of comparisons made between a recursive bubble sort and the iterative bubble sorts.

Using Recursion to perform a Bubble Sort

- First, how many times will the function call itself?
    - The base case is reached when length = 1
    - Each recursive call subtracts one from the length
    - This means the method will call itself *length* times

- The number of repetitions in the for loop decreases with each recursive call
    - length causes length-1 repetitions in the for loop
    - length-1 causes length-2 repetitions
    - length-2 causes length-3 repetitions
    - and so on…

Using Recursion to perform a Bubble Sort

- Let’s say we have an array with a length of five
  - First call (length = 5), for loop repeats length-1 (4) times
  - Second call (length = 4), for loop repeats length-1 (3) times
  - Third call (length = 3), for loop repeats length-1 (2) times
  - Fourth call (length = 2), for loop repeats length-1 (1) time
  - Fifth call (length = 1), method returns (base case)

- Each iteration of the for loop performs one comparison
  - Total Comparisons **=** $\boldsymbol{\sum_{i=1}^{n-1} i}$
  - **Same as Bubble Sort Improved 2**

Merge Sort and Quicksort

- We’ll now see a pair of comparative sorting algorithms, Merge Sort and Quicksort, that are typically implemented using recursion.
  - They can be done iteratively, but the code is much easier to read when implemented recursively.

- Both algorithms take a “divide and conquer” approach to sorting, much like the how the binary search algorithm performs its searches.

Merge Sort Algorithm

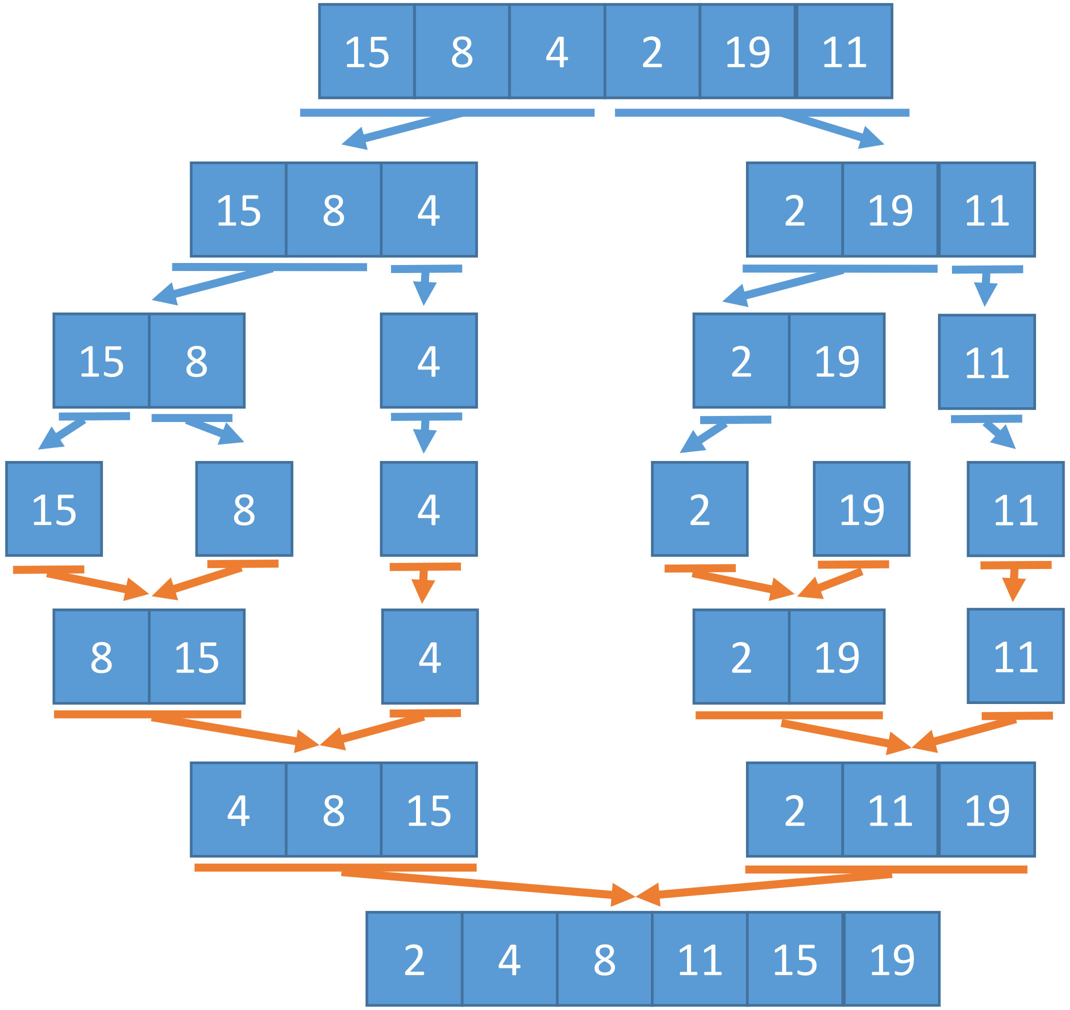
- In the Merge Sort algorithm, the array is repeatedly (recursively) split in half until it reaches halves that only contain one element.
  - At this point, the lowest depth has been reached.

- Then, working backwards, it sorts/merges the smaller arrays back together

Merge Sort Algorithm

- The image on the right gives the basic idea of how it works.
  - It divides up the array (Blue lines)
  - Then merges the array back together (Orange lines)

- Each merge involves two, sorted arrays.
  - Since the arrays to merge are in order, merging them is not computationally difficult.



Merge Sort Algorithm

- The Java functions are a bit too long to put here in their entirety.
  - See the Sample Code provided.

- Here is a link to an online tool that visualizes the Merge Sort sorting an array of numbers.

https://www.sortvisualizer.com/mergesort/

Quicksort

- In the Quicksort algorithm, the array is repeatedly (recursively) split into two smaller partitions, until the partitions only contain one element.
    - At this point, the lowest depth has been reached.

- The algorithm chooses a value in each partition, called the **pivot**.
    - One of the two partitions will contain any values less than the pivot.
    - The other partition will contain any values greater than the pivot.

- The process repeats recursively until partitions of length 1 are reached.
    - At which point, the array will have been sorted through the pivot processes.

Quicksort

- There are a few ways of selecting the pivot:
  - Always use the last element.
  - Always use the middle element.
  - Always use the first element.
  - Always use a randomly chosen element.
  - The Sample Code provided uses the middle element as the selected pivot value.
- Here is a link to an online tool that visualizes the Merge Sort sorting an array of numbers.

https://www.sortvisualizer.com/quicksort/