Today

• Introduction to Sorting
  – Bubble sort
  – Selection sort
  – Merge sort

– You should understand the idea behind bubble sort & selection sort
– You should be able to understand the code given in slides (and know how to use the code in similar problems by making slight modification).
Bubble sort

- Compare each element (except the last one) with its neighbor to the right
  - If they are out of order, swap them
  - This puts the largest element at the very end
  - The last element is now in the correct and final place

- Compare each element (except the last two) with its neighbor to the right
  - If they are out of order, swap them
  - This puts the second largest element next to last
  - The last two elements are now in their correct and final places

- Compare each element (except the last three) with its neighbor to the right
  - Continue as above until you have no unsorted elements on the left
Bubble Sort

- **Step-by-step example of "5 1 4 2 8"**

  - **First Pass:**
    (5 1 4 2 8) → (1 5 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
    (1 5 4 2 8) → (1 4 5 2 8), Swap since 5 > 4
    (1 4 5 2 8) → (1 4 2 5 8), Swap since 5 > 2
    (1 4 2 5 8) → (1 4 2 5 8), Since these elements are already in order (8 > 5), algorithm does not swap.

  - **Second Pass:**
    (1 4 2 5 8) → (1 4 2 5 8)
    (1 4 2 5 8) → (1 2 4 5 8), Swap since 4 > 2
    (1 2 4 5 8) → (1 2 4 5 8)
    (1 2 4 5 8) → (1 2 4 5 8)
    Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without any swap to know it is sorted.

  - **Third Pass:**
    (1 2 4 5 8) → (1 2 4 5 8)
    (1 2 4 5 8) → (1 2 4 5 8)
    (1 2 4 5 8) → (1 2 4 5 8)
    (1 2 4 5 8) → (1 2 4 5 8)
public static void bubbleSort(int[] data)
{
    for (int k = 0; k < data.length - 1; k++)
    {
        for (int i = 0; i < data.length - 1 - k; i++)
        {
            if (data[i] > data[i + 1])
            {
                int temp = data[i]; // swap data[i] and data[i+1]
                data[i] = data[i + 1];
                data[i + 1] = temp;
            }
        }
    }
}
Bubble Sort

• Animation from Wikipedia:

\[
\begin{align*}
6 & \quad 5 & \quad 3 & \quad 1 & \quad 8 & \quad 7 & \quad 2 & \quad 4
\end{align*}
\]

Selection sort

- Given an array of length $n$,
  - Search elements 0 through $n-1$ and select the smallest
    - Swap it with the element in location 0
  - Search elements 1 through $n-1$ and select the smallest
    - Swap it with the element in location 1
  - Search elements 2 through $n-1$ and select the smallest
    - Swap it with the element in location 2
  - Search elements 3 through $n-1$ and select the smallest
    - Swap it with the element in location 3
  - Continue in this fashion until there’s nothing left to search
Example and analysis of selection sort

- Step by step example: 7 2 8 5 4
- Iteration 1: found 2, swap it with 7
- Iteration 2: found 4, swap it with 7
- Iteration 3: found 5, swap it with 8
- Iteration 4: found 7, swap it with 7
- The selection sort might swap an array element with itself--this is harmless, and not worth checking for
public static void selectionSort(int[] anArray) // textbook version
{
    for(int i = 0; i < anArray.length - 1; i++)
    {
        int iSmallest = getIndexOfSmallest(i, anArray);
        swap(i, iSmallest, anArray);
    }
}

private static int getIndexOfSmallest(int startIndex, int[] a)
{
    int min = a[startIndex];
    int indexOfMin = startIndex;
    for (int index = startIndex + 1; index < a.length; index++)
    {
        if (a[index] < min)
        {
            min = a[index];
            indexOfMin = index;
            //min is smallest of a[startIndex] through a[index]
        }
    }
    return indexOfMin;
}
Selection Sort

• Write all in one method:
Selection Sort

- Animation from Wikipedia:

```
8
5
2
6
9
3
1
4
0
7```

• Bubble Sort and Selection Sort:
  – Intuitive and easy to implement
  – Help build basic abstract sorting concepts
  
  – Requires $\sim n^2 \times c$ operations in worst case
    • $n$: number of items to sort
    • $c$: some constant factor
  
  – Not really used in practice

• Two commonly used sorting algorithms in practice:
  – Quick Sort & Merge Sort
• **Strategy:** Recursively split the list in half and merge the two returned segments.

• Java’s built-in sort function is a variant of merge sort
  ```java
  Collections.sort( .. );
  ```

• \( \sim n \log(n) \times c \) operations in worst case
  - Check the difference between \( n \log(n) \) and \( n^2 \) when \( n \) is large.

![Graph comparing \( n^n \) and \( n \log(n) \) vs. \( n \) value](image-url)
Split the array into two or more parts

Sort each part individually

Merge
Selection Sort

• Animation from Wikipedia:

   6  5  3  1  8  7  2  4

• http://www.cs.pitt.edu/~kirk/cs1501/animations/SortApplets.html
Merge Sort

• Not easy to implement Merge sort correctly.

• No Java code here (beyond the level of COMP110)

• Just understand the big idea behind