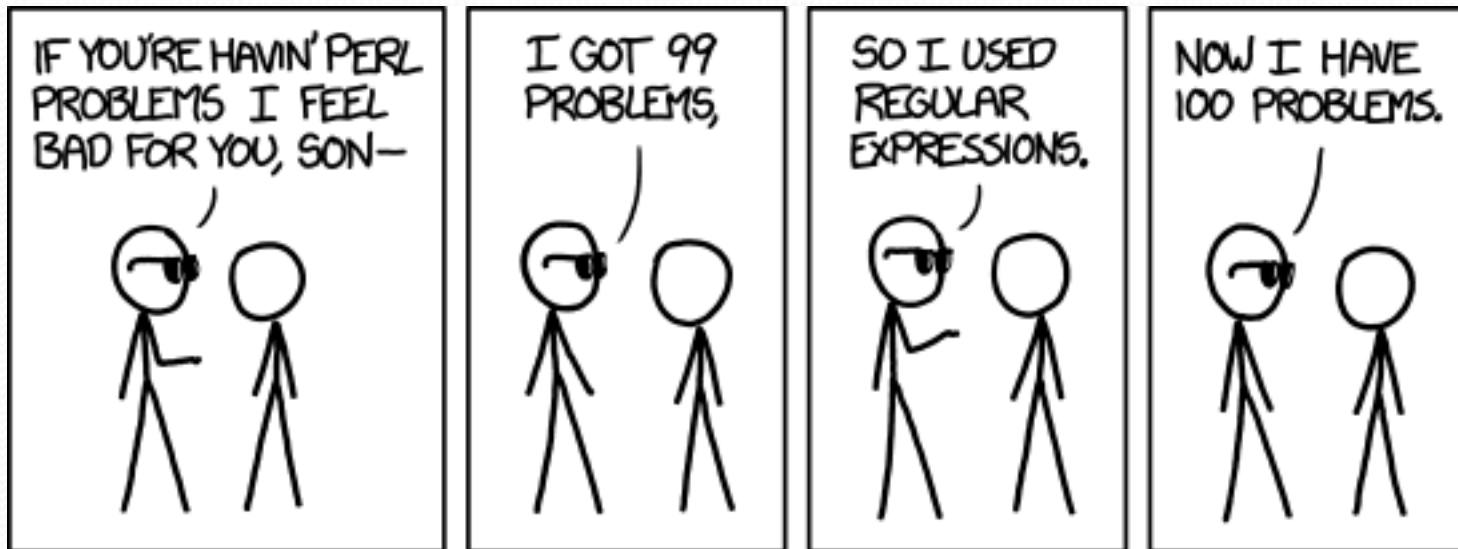


CSE 154

Sorting

reading: 13.3, 13.4



```
\b[A-Z0-9._%+~]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b
```

Collections class

Method name	Description
<code>binarySearch(list, value)</code>	returns the index of the given value in a sorted list (< 0 if not found)
<code>copy(listTo, listFrom)</code>	copies listFrom 's elements to listTo
<code>emptyList()</code> , <code>emptyMap()</code> , <code>emptySet()</code>	returns a read-only collection of the given type that has no elements
<code>fill(list, value)</code>	sets every element in the list to have the given value
<code>max(collection)</code> , <code>min(collection)</code>	returns largest/smallest element
<code>replaceAll(list, old, new)</code>	replaces an element value with another
<code>reverse(list)</code>	reverses the order of a list's elements
<code>shuffle(list)</code>	arranges elements into a random order
<code>sort(list)</code>	arranges elements into ascending order

Sorting

- **sorting**: Rearranging the values in an array or collection into a specific order (usually into their "natural ordering").
 - one of the fundamental problems in computer science
 - can be solved in many ways:
 - there are many sorting algorithms
 - some are faster/slower than others
 - some use more/less memory than others
 - some work better with specific kinds of data
 - some can utilize multiple computers / processors, ...
 - *comparison-based sorting* : determining order by comparing pairs of elements:
 - `<`, `>`, `compareTo`, ...

Sorting algorithms

- **bogo sort:** shuffle and pray
- **bubble sort:** swap adjacent pairs that are out of order
- **selection sort:** look for the smallest element, move to front
- **insertion sort:** build an increasingly large sorted front portion
- **merge sort:** recursively divide the array in half and sort it
- **heap sort:** place the values into a sorted tree structure
- **quick sort:** recursively partition array based on a middle value

other specialized sorting algorithms:

- **bucket sort:** cluster elements into smaller groups, sort them
- **radix sort:** sort integers by last digit, then 2nd to last, then ...
- ...

Bogo sort

- **bogo sort:** Orders a list of values by repetitively shuffling them and checking if they are sorted.
 - name comes from the word "bogus"

The algorithm:

- Scan the list, seeing if it is sorted. If so, stop.
 - Else, shuffle the values in the list and repeat.
- This sorting algorithm (obviously) has terrible performance!
 - What is its runtime?

Bogo sort code

```
// Places the elements of a into sorted order.
```

```
public static void bogoSort(int[] a) {  
    while (!isSorted(a)) {  
        shuffle(a);  
    }  
}
```

```
// Returns true if a's elements are in sorted order.
```

```
public static boolean isSorted(int[] a) {  
    for (int i = 0; i < a.length - 1; i++) {  
        if (a[i] > a[i + 1]) {  
            return false;  
        }  
    }  
    return true;  
}
```

Bogo sort code, cont'd.

```
// Shuffles an array of ints by randomly swapping each  
// element with an element ahead of it in the array.
```

```
public static void shuffle(int[] a) {  
    for (int i = 0; i < a.length - 1; i++) {  
        // pick a random index in [i+1, a.length-1]  
        int range = a.length - 1 - (i + 1) + 1;  
        int j = (int) (Math.random() * range + (i + 1));  
        swap(a, i, j);  
    }  
}
```

```
// Swaps a[i] with a[j].
```

```
public static void swap(int[] a, int i, int j) {  
    if (i != j) {  
        int temp = a[i];  
        a[i] = a[j];  
        a[j] = temp;  
    }  
}
```

Selection sort code

```
// Rearranges the elements of a into sorted order using
// the selection sort algorithm.
public static void selectionSort(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        // find index of smallest remaining value
        int min = i;
        for (int j = i + 1; j < a.length; j++) {
            if (a[j] < a[min]) {
                min = j;
            }
        }
        // swap smallest value its proper place, a[i]
        swap(a, i, min);
    }
}
```


Merge sort

- **merge sort:** Repeatedly divides the data in half, sorts each half, and combines the sorted halves into a sorted whole.

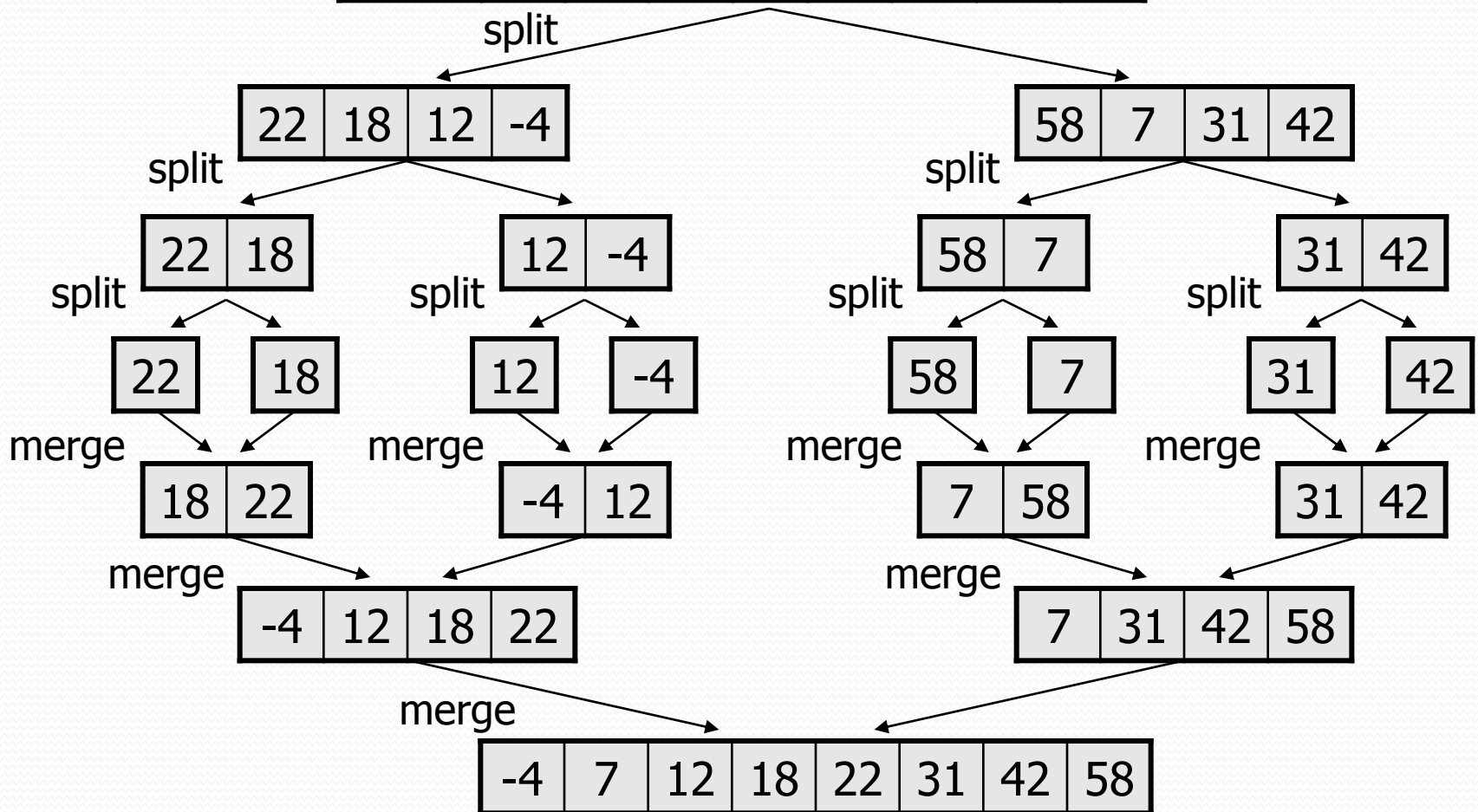
The algorithm:

- Divide the list into two roughly equal halves.
- Sort the left half.
- Sort the right half.
- Merge the two sorted halves into one sorted list.

- Often implemented recursively.
- An example of a "divide and conquer" algorithm.
 - Invented by John von Neumann in 1945

Merge sort example

index	0	1	2	3	4	5	6	7
value	22	18	12	-4	58	7	31	42



Merge halves code

```
// Merges the left/right elements into a sorted result.
// Precondition: left/right are sorted
public static void merge(int[] result, int[] left,
                        int[] right) {
    int i1 = 0;    // index into left array
    int i2 = 0;    // index into right array

    for (int i = 0; i < result.length; i++) {
        if (i2 >= right.length ||
            (i1 < left.length && left[i1] <= right[i2])) {
            result[i] = left[i1];    // take from left
            i1++;
        } else {
            result[i] = right[i2];    // take from right
            i2++;
        }
    }
}
```

Merge sort code

```
// Rearranges the elements of a into sorted order using
// the merge sort algorithm.
public static void mergeSort(int[] a) {
    // split array into two halves
    int[] left  = Arrays.copyOfRange(a, 0, a.length/2);
    int[] right = Arrays.copyOfRange(a, a.length/2,
a.length);

    // sort the two halves
    ...

    // merge the sorted halves into a sorted whole
    merge(a, left, right);
}
```

Merge sort code 2

```
// Rearranges the elements of a into sorted order using
// the merge sort algorithm (recursive).
public static void mergeSort(int[] a) {
    if (a.length >= 2) {
        // split array into two halves
        int[] left  = Arrays.copyOfRange(a, 0, a.length/2);
        int[] right = Arrays.copyOfRange(a, a.length/2,
a.length);

        // sort the two halves
        mergeSort(left);
        mergeSort(right);

        // merge the sorted halves into a sorted whole
        merge(a, left, right);
    }
}
```