Finding the Mode of a list

- Mean = total/count
- Median = middle value, separating top 50% from bottom 50%
- Mode = most frequent number.

23,22,49,25,43,23,5,31,43,27,21,45,43,16,5,21,18,27,39,18,21,7,42,28,21,19

Algorithm:

- set *mode* to the first number and *modeCount* to 1
- **for** each value in the list:
  - step through the list to count how many times value occurs in the list
  - **if** *count* > *modeCount* **then** set *mode* and *modeCount* to *value* and *count*

What's the cost if there are *n* numbers?

Mode: the bad way

```
public int mode(List<Integer>numbers){
    int mode = numbers.get(0);
    int modeCount = 1;
    for (int value : numbers){
        int count = 0;
        for (int other : numbers){
            if (other == value) {
                count++;
            }
        }
        if (count > modeCount) {
            mode = value;
            modeCount = count;
        }
    }
    return mode;
}
```

Analysis

O(1)	int mode = numbers.get(0);	1 x O(1)
O(1)	int modeCount = 1;	1 time
	for (int value : numbers){	
O(1)	int count = 0;	n times
	for (int other : numbers){	
O(1)	if (other == value) {	nxn times
O(1)	count++;	n … nxn times
	}	
	}	
O(1)	if (count > modeCount) {	n times
O(1)	mode = value;	1 … n times
O(1)	modeCount = count;	1 … n times
	}	
	}	
O(1)	return mode;	1 time
	}	

n*n iterations

n iterations

Finding the Mode of a list faster

- Much easier to see if the list is sorted in order:

  23,22,49,25,43,23,5,31,43,27,21,45,43,16,5,21,18,27,39,18,21,7,42,28,21,19

  5,5,7,16,18,18,19,21,21,21,21,22,23,23,25,27,27,28,31,39,42,43,43,43,45,49

- Algorithm
  - sort the list
  - set *mode* to first number and *modeCount* to 1
  - set *count* to 1
  - **step** through the list from index 1
    - **if** the number is the same as the previous number, **then** increment *count*
    - **else**
      - **if** *count* > *modeCount*, **then** set *mode* and *modeCount* to previous value and *count*
      - reset *count* to 1
  - **if** *count* > *modeCount*, **then** set *mode* and *modeCount* to previous value and *count*

What's the cost if there are *n* numbers?

Finding the Mode of a list faster

Analysis

- Algorithm
  - sort the list **O(n log(n))** — **1 time**
  - set *mode* to first number and *modeCount* to 1 **O(1)** — **1 time**
  - set *count* to 1 **O(1)** — **1 time**
  - **step** through the list from index 1 — **n iterations**
    - **if** number is same as previous number, **then** **O(1)** — **n times**
      - increment *count* **O(1)** — **1 … n times**
    - **else**
      - **if** *count* > *modeCount*, **then** — **n … 1 times**
        - **O(1)** set *mode* and *modeCount* to previous number and *count* — **n … 1 times**
        - reset *count* to 1 **O(1)** — **n … 1 times**
  - **if** *count* > *modeCount*, **then** **O(1)** — **1 time**
    - **O(1)** set *mode* and *modeCount* to previous value and *count*

Total: O(n log(n))

Finding the Mode of a list even faster

- Count using a map to count without sorting:

23,22,49,25,43,23,5,31,43,27,21,45,43,16,5,21,18,27,39,18,21,7,42,28,21,19

5-2 7-1 16-1 18-2 19-1 21-4 22-1 23-2 25-1
27-2 28-1 31-1 39-1 42-1 43-3 45-1 49-1

What's the cost if there are *n* numbers?

- Algorithm
  - **for** each value in the list
    - **if** the value is in the map, **then** increment the associated *count*
    - **else** add the value to the map with an associated count of 1.
  - **for** each key in map,
    - **if** associated count > *modeCount*, **then** set *mode* and *modeCount* to key and count

Finding the Mode of a list even faster

- Algorithm
  - **for** each value in the list
    - **if** the value is in map, **then**
      - increment the associated *count*
    - **else**
      - add value to map with associated count =1.

n times

  - **for** each key in map,
    - **if** associated count > *modeCount*, **then**
      - set *mode* and *modeCount* to key and count

n times

Analysis

n x **O(1)**	containskey(key)
1…n x **O(1)**	get(..) & put(..)
n…1 x **O(1)**	put(key, 1)
O(1)	get all keys
n x **O(1)**	get(key)
1…n x **O(1)**	

Total: O(n)