CSSE 220

Merge Sort

Import *MergeSortSimple* project from repo

Today’s Plan

- Merge sort – a divide and conquer algorithm
- Big-O practice
  - Understanding Big-O helps move you from being *just a programmer* to being a *software engineer*
  - It’s on the Final

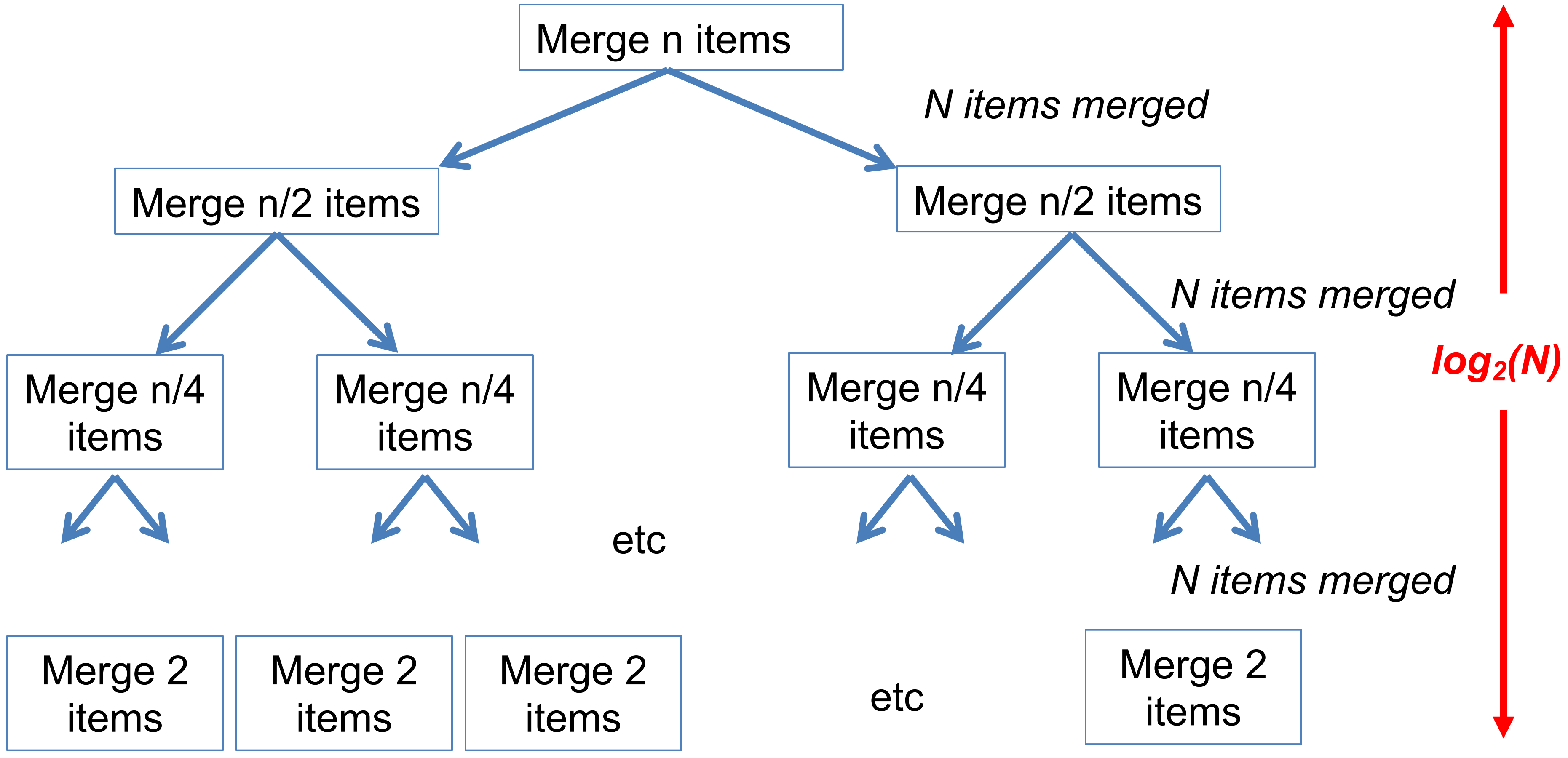
Merge Sort

- Basic recursive idea:
  - If list is length 0 or 1, then it's already sorted
  - Otherwise:
    - Make progress toward the base-case by dividing the incoming collection into two halves
      This gives us the $\log_2(N)$ part of the performance
    - Make 2 recursive calls to mergeSort the two halves
    - **Merge** the sorted halves back together
      This gives us the N part of the performance

Analyzing Merge Sort

If list is length 0 or 1,
then it's already sorted

- Otherwise:
  - Divide list into two halves
  - Make 2 recursive calls to sort the two halves
  - **Merge** the sorted halves back together



Next Week: Your Presentations

- Final submission of code due before class

- 8 minutes long
- ~3 minutes showing off your extra features
- ~5 minutes explaining JUST ONE technical decision in detail

Presentation Hints

- Know your time constraint 8 minutes long
  - Practice so it falls within 7.5 to 8.5 minutes
- ~3 minutes showing off your extra features
  - Give us a demo of your game, but …
  - Focus on what makes your game different
  - Everyone knows the game and its rules

Presentation Hints

- Explain one technical item in detail, ~5 minutes:
  - A bug you had to overcome
  - A cool way to do something
  - Etc.
- Possibly include:
  - A piece of code:
    - Put the code and put it into a pptx slide
    - Both poor design and good design
    - Do not zoom through a lots of classes
  - A UML diagram of the focused on part

Arcade Game Presentation

So the arcade game presentation is a way for you to show off your cool work and also show us (the professors) that you understand some of the hard issues we've been trying to teach you. In the presentation it's most important you do two things:

1. Showcase your special features. Remember that everyone in your class has built the same game, so there's no need to go through all the regular stuff in detail. Time: approximately 1/3 of your talk.
2. Talk about 1 interesting design challenge you had, and how you solved it. Time: approximately 2/3 of your talk.

Here's how the grading works (25 points total):

5 points: All team members in attendance. Each team member should also participate in the presentation in some way.

5 points: Team dressed up. You must dress nicely for the talk. Each team member should be in "business casual attire". Collared shirts, no jeans, etc. As long as I can tell you tried, you're probably OK.

5 points: Polished presentation, right length. Usually these talks are 10 minutes/team but depending on the class size it may be slightly different. It should be clear that you prepared for this talk and practiced. People should know their roles and the presentation should take the full time but not go over.

5 points: Clear design presentation. Explaining a design issue is difficult. Looking at a giant UML diagram with all your classes is usually not helpful, nor is just quickly moving through a whole bunch of code real quickly. What you'll want to explain is a UML diagram that highlights ONLY the issue you want to present, or slides with code that highlight the key section. One way or another, it should be clear that you've given thought to how to present your design issue in a good way.

5 points: Correct/Interesting design issue presented. You need to choose a topic that has a genuinely interesting design problem, discuss the tradeoffs well and correctly, and then come to a sane conclusion.

Big-O Practice

Observations:

1. O(N) – A single for loop indexed from 0..(N-1)
2. $O(N^2)$ – A nested for loop both from 0..(N-1)
3. O(N) – Back-to-back loops of type #1 (above)
4. If you see loop control where the exit condition depends on an expression that looks like: k = k/2, then it's a good bet that the answer has a $\log_2(N)$ in it

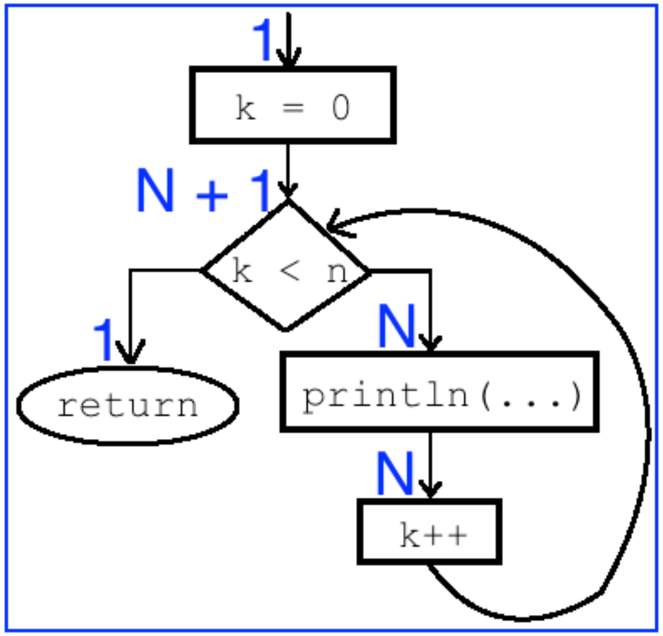
Question #1 – Jump Start

```
public static void fun1(int n) {
  for (int k = 0; k < n; k++) {
    System.out.println("println runs in constant time");
  } // end for
} // fun1
```

fun1's function: $f(N) = 3N^1 + 3N^0$

To get Big-O:

1. $3N^1 + \cancel{3N}^0$ – lower order terms
2. $\overset{1}{\cancel{3}}N^1$ – constant coefficients to 1
3. O(N) – write Big-O


1
k = 0
N + 1
k < n
1
N
return
println(...)
N
k++