High Scores I

Team Name:

Manager:

Recorder:

Presenter:

Analyst:

This is a Process Oriented Guided Inquiry Learning (POGIL) activity. You and your team will examine a working program. A series of questions will guide you through a cycle of exploration, concept invention, and application. There is strong evidence that this is more effective (and less boring) than a traditional lecture.

By the time you are done with this activity, you and your team should be able to:

- explain and modify the insertion sort algorithm.
- explain and modify the mergesort algorithm.
- assess your team's process and results more effectively.

Your team's recorder is responsible for writing your team's answers to the numbered questions on this form.

After you complete this activity, please fill out the short survey at

http://goo.gl/forms/HXjyuUb2ou

to improve this activity for future users.

Running the program

This program is not a game *per se*, but could be used to sort a list of high scores for any game.

Run HighScores.java and look at the output in the console.

1. Is everyone done running the program and ready to pay attention to the team?

You may need to go back to the game to answer some of the questions to come, but you should do so *deliberately*, because your team's manager assigned one or more people to find something out, not merely because you got bored with the conversation or thought you could answer a question better on your own.

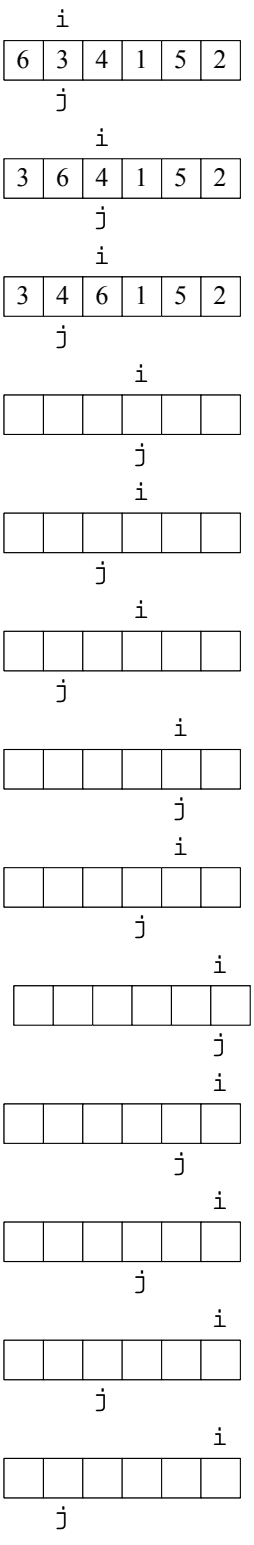
2. Are the numbers sorted in increasing or decreasing order?

3. Do both sorting algorithms (insertion sort and merge sort) produce the same result?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have.

Insertion sort

Suppose we run `testInsertionSortEasy`. The diagrams at right show the state of the array `scores` at the beginning of each pass through the inner for loop (line 10). While `i` and `j` are really just ints, they are shown here above and below the array because they are used as array indices. For example, in the first (top) diagram, `i` and `j` are both 1.



4. Devise and describe a plan for filling in the remaining diagrams. Hint: if it's tedious, you're doing it wrong!

5. Carry out your plan.

6. Did your team check over your answers?

7. What does each pass through the inner loop accomplish?

8. At the beginning of each pass through the outer loop, what is true of the part of the array to the left of element `i`?

9. What does each pass through the outer loop accomplish?

10. Would the method still work correctly if, in the outer loop, `i` started at 0 instead of 1? If so, is there a reason to prefer 1 over 0? If not, why not?

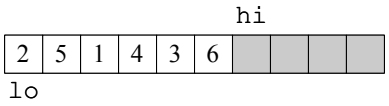
11. How would you modify the method to sort the scores in *decreasing* order instead of increasing order?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have.

Mergesort

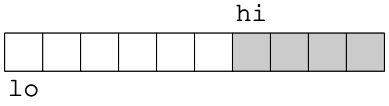
12. The HighScoreList class has two overloaded versions of `mergeSort`. Which one is called directly by the tests?

The diagram to the right shows `scores` at the beginning of a call to the recursive `mergeSort` method.



13. What value is chosen for `mid`?

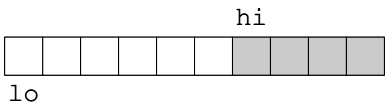
14. Fill in the diagram at right to show the data structure after the *first* recursive call (line 78).



15. How did you determine your answer to the previous question?

16. How confident are you in the result?

17. Fill in the diagram at right to show the data structure after the *second* recursive call (line 79).



18. Draw the array returned by the subsequent call to `merge` (line 81).

19. Why does `merge` need a three-way if/else if/else statement? In other words, why can't it simply check if `scores[b] < scores[a]`?

20. `merge` creates a separate array for its results; the elements of this array are then copied back into `scores` by `mergeSort`. Why couldn't `merge` simply write each element directly into the proper place in `scores`?

21. How would you modify these methods to sort the scores in *decreasing* order instead of increasing order?

22. In `testInsertionSortHard`, modify line 18 to change the length of the array from 1000 to 200000. Do the same on line 37. Close any other applications on the computer (e.g., web browsers). Run the tests. Next to the results of the individual tests in the JUnit view, Eclipse displays the time taken by each test. Which sorting algorithm is faster?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have.

Reflection

23. What were your team's strengths?

24. What were potential areas of improvement?

25. What insights did you gain?

Stop here and wait for the other teams. If your instructor has given you a way to indicate that you have reached this point, use it now. Once all teams are ready, there will be a short discussion involving the whole class. Your team's presenter should be prepared to present any of your team's previous answers to the class. This discussion is also a good time for your team (through your presenter) to ask any questions you have.

Please fill out the survey at http://goo.gl/forms/HXjyuUb2ou.