High Scores II

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:

- implement a set using a sorted array.
- implement a set using a binary search tree.
- implement a set by delegating to objects from the Java Collections Framework.
- explain the relative merits of these implementations.
- process information more effectively as a team.

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

Running the program

This program is not a game *per se*, but could be used to maintain a list of high scores for any game. Each member of your team should run HighScores.java and play with the program briefly.

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

You may need to go back and run the program 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. Once there are several scores stored, are they sorted in increasing order, in decreasing order, or not at all?

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. If your team is done before other teams, discuss the following open-ended question:

3. What happens if you enter a score that is already present? What alternative behaviors can you imagine? In what situations would each behavior be the best choice?

Set

Examine Set.java.

4. What methods are specified by this interface?

5. The code says that `Set<T> extends Iterable<T>`. Without seeking further information, write down what your team thinks this might mean.

6. How could you test your hypothesis?

7. When you are confident that you know what it means for an interface to extend another interface, write it down.

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. If your team is done before other teams, discuss the following open-ended question:

8. Draw a UML class diagram of the relationships between all of the classes and interfaces in this project, as well as the built-in classes TreeSet and HashSet.

SortedArraySet

Examine SortedArraySet.java.

9. Assuming the first `size` elements of `data` are sorted, what is the order of the worst-case running time of `find`? (Hint: you've seen this algorithm before!)

10. What is the order of the worst-case running time of `contains`?

11. What does the while loop in `add` accomplish?

12. What is the order of the worst-case running time of `add`?

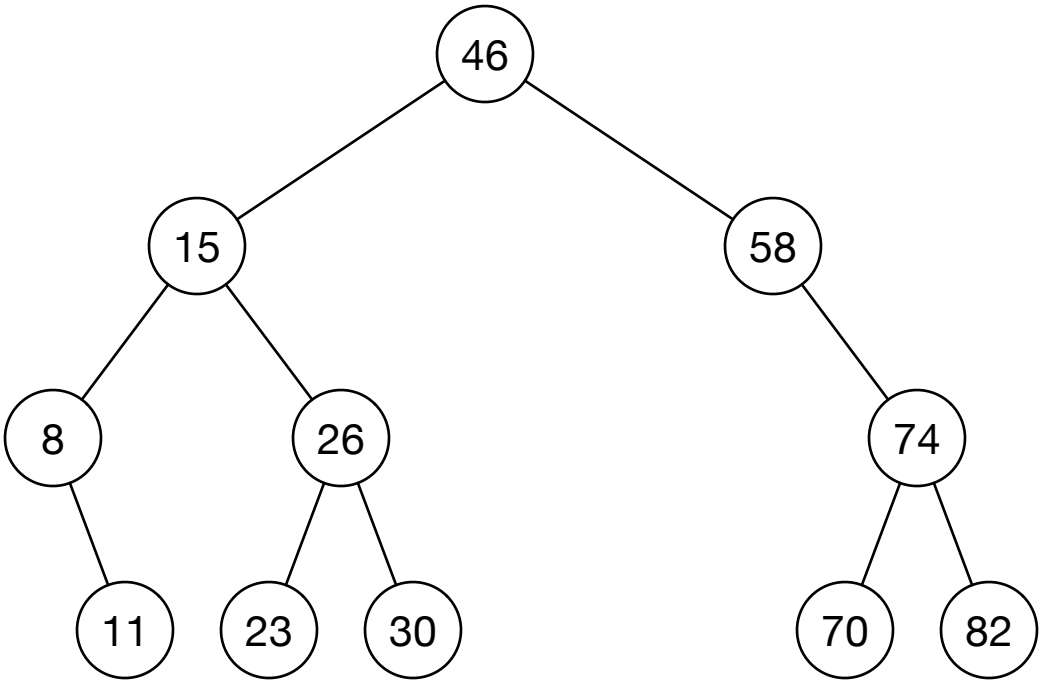
13. What is the order of the worst-case running time of `remove`?

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. If your team is done before other teams, discuss the following open-ended question:

14. Would an unsorted implementation work just as well? Explain.

Binary search tree

The diagram below shows a binary search tree.



15. What is true of everything to the left of 46?

16. What is true of everything to the right of 46?

17. Within the part of the tree to the left of 46, what is true of everything to the left of 15?

18. Within the part of the tree to the left of 46, what is true of everything to the right of 15?

19. Complete the definition of a binary search tree below.

A binary search tree is either:

- Empty, or
- Consists of an item, a left subtree, and a right subtree. Both subtrees are themselves binary search trees. Everything in the left subtree is _____ the item. Everything in the right subtree is _____ the item.

Suppose you are looking in the tree for a particular item, say, 23. You start by comparing the item you are seeking (23) to 46.

20. Seeing that 23 < 46, what part of the tree do you now know that you won't have to examine in your search for 23?

21. To which number would you now compare 23?

22. List all of the numbers that you would examine in the search for 23.

23. Suppose you now wanted to add 50 to the tree. How could you do so, changing the tree as little as possible but keeping it within the definition of a binary search tree?

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. If your team is done before other teams, discuss the following open-ended question:

24. Describe an algorithm for *removing* an item from a binary search tree.

BinarySearchTreeSet

Examine BinarySearchTreeSet.java.

25. In terms of the trees discussed in the previous section, does each pass through the loop in `contains` move one level *up* or *down* the tree?

26. Again in terms of the tree, when does the loop end, returning false?

27. When does the loop end early, returning true?

28. The one-argument `add` method calls a second, recursive, two-argument `add` method. In what situation does the recursive version *not* create a new BinaryNode?

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. If your team is done before other teams, discuss the following open-ended question:

29. Describe, in plain English, the algorithm carried out by `traverse`. Assume your reader understands recursion but does not know this particular algorithm.

JcfTreeSet and JcfHashSet

Examine JcfTreeSet.java. ("JCF" stands for Java Collections Framework, a library of classes in the java.util package.)

30. What is the type of the instance variable in this class?

31. How do `add`, `contains`, `iterator`, and `remove` get their work done?

32. Discuss the advantages of writing a class this way rather than building it "from scratch", as in SortedArraySet and BinarySearchTreeSet.

Examine JcfHashSet.java.

33. How does this class differ from JcfTreeSet?

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. If your team is done before other teams, discuss the following open-ended question:

34. Speculate on why the designers of the Java Collections Framework would have supplied two classes that provide the same methods.

Comparison of implementations

Modify HighScores.java so that the `main` method calls `iterationTest`. Run the program.

35. Do all four implementations sort the scores in the same way? If so, are they sorted in increasing or decreasing order? If not, how do they differ?

Modify HighScores.java so that the `main` method calls `speedTest`. Run the program.

36. Which implementation is fastest?

37. Which implementation is slowest?

38. Which implementation would you use if you cared about iterating through the scores in order?

39. Which implementation would you use if you *did not* care about iterating through the scores in 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. If your team is done before other teams, discuss the following open-ended question:

40. Describe a plan for modifying the program to associate a player name with each high score.

This work was supported by the Google Education and Relations Fund's CS Engagement Small Grant Program grant #TFR15-00411.

The principal investigator, Peter Drake, wishes to thank the following for their useful comments: the members of the CS-POGIL project, specifically Clif Kussmaul and Helen Hu; Maggie Dreyer; and various anonymous reviewers.