



Technical Support

For assistance with technical issues, please click to visit the [Online Course Support Center](#).

Navigation

Home

- [My home](#)
- [Site pages](#)
- [My profile](#)
- ▼ [Current course](#)
 - ▼ [CS111C-META-SPRING-2017](#)
 - [Participants](#)
 - [General](#)
 - [January 17 - January 23](#)
 - [January 24 - January 30](#)
 - [January 31 - February 6](#)
 - [February 7 - February 13](#)
 - [February 14 - February 20](#)
 - [February 21 - February 27](#)
 - [February 28 - March 6](#)
 - [March 7 - March 13](#)
 - [March 14 - March 20](#)
 - [March 21 - March 27](#)
 - [March 28 - April 3](#)
 - [April 4 - April 10](#)
 - [April 11 - April 17](#)
 - [April 18 - April 24](#)
 - [April 25 - May 1](#)
 - ▼ [May 2 - May 8](#)
 - [Week 15 Checklist](#)
 - [Week 15: Heaps and Balanced Trees](#)

Lab F: Trees and Big Data

For this lab, first modify a binary search tree so that it allows duplicates. Then, use this new binary search tree to organize records from a big data set.

Part A: Binary Search Tree with Duplicates (68 points)

The class `BinarySearchTreeWithDups` represents a binary search tree in which duplicate entries are allowed.


- A duplicate entry is placed in the entry's right subtree, as described in the textbook Section 25.4.
- Essentially, if you hit a duplicate, go into the right subtree. Then continue to look for the right place to put the entry, just as you normally would in a BST.
- In this implementation, we will assume the `getEntry` method returns the first match it finds and the `remove` method removes the first match it finds. So the only modification required is the `add` method.


The class `BinarySearchTreeWithDups` extends `BinarySearchTree`,

- The class shell is provided with "???" as placeholders for the parts you will modify.
- There are many classes required to get this lab to compile. All are included in the provided zip. **You should only modify the `BinarySearchTreeWithDups` class.**
- Begin by closely reviewing `BinarySearchTree` and `BinaryTree` classes. Make sure you understand how these classes work before you implement `BinarySearchTreeWithDups`. You must have a good understanding of how the regular BST class works before you can make modifications. I cannot stress this enough!

Implement `BinarySearchTreeWithDups`


1. Write an iterative `addEntryHelperIterative` method. (20 points)
 - In the `BinarySearchTreeWithDups` class, we override the `add` method to call a new private `addEntryHelper` method.
 - You will write this helper method.
 - The helper method allows duplicate entries to be added, using the algorithm described above.
 - Important: This method must be written iteratively (not recursively) in order for Part B to run.
 - I recommend reviewing the `addEntry` method in `BinarySearchTree` class and then thinking about how to modify it.
2. Write a `getAllEntriesRecursive` method. (12 points)
 - This method returns an `ArrayList` of all the entries in a tree that match a target.
 - Use recursion. You can add a private helper method if necessary.
3. Write a `getAllEntriesIterative` method. (12 points)
 - Use iteration.

 Questions on Week 15 Readings and Exercises


 Practice/Discussion Questions: Week 15 (Heaps and ...

 Homework W15: Heaps and B-Trees

 Homework W15: Extra Credit

 Lab F: Trees and Big Data

 Lab F Driver and Java Files

 Questions on Lab F (Trees and Big Data)

► May 9 - May 15

► May 16 - May 22

► My courses

Administration

► Course administration

► My profile settings

Activities

 Assignments

 Checklists

 Forums

 Questionnaires

 Quizzes

 Resources

4. Write a `getAllEntriesLessThanRecursive` method. (12 points)

- This method returns an `ArrayList` of all entries in the tree less than a target.
- Use recursion. You can add a private helper method if necessary.
- Note that the elements in the tree implement `Comparable`, so you can invoke the `compareTo` method.

5. Write a `getAllEntriesLessThanIterative` method. (12 points)

- Use iteration.

Test your program

- Use the driver program to test your methods. You might consider adding more tests to the driver.
- The first part of the driver shows how the regular BST class functions. The second part uses your implementation of the `BinarySearchTreeWithDups`.
- The zip file includes pictures of the two trees created by the driver.
- Make sure Part A is working properly before moving on to Part B!

Part B: Big Data (32 Points)

Use the `BinarySaarchTreeWithDups` class to process a big data file.

- The data file is a list of San Francisco police incident reports for Larceny/Theft from 2003 to 2015 (downloaded from <https://data.sfgov.org/Public-Safety/SFPD-Incidents-from-1-January-2003/tmnf-yvry>)
 - There are over 370,000 records in the file.
- I have provided two classes to process the file:
 - `PoliceReport` represents a single report.
 - `ReportProcessor` reads in the data file, creates a tree, and then uses the `getAllEnrties` method to create lists that match certain criteria.
 - In eclipse, place the data file in the same folder as the `src/bin` folders (so one level above the `.java` files).
- Review these files to become familiar with how they work.
 - At first, `PoliceReport` objects are compared/ordered by date (represented as a `String`). Two reports with the same date are considered "the same." You can see this in the `equals` and `compareTo` methods of the `PoliceReport` class.
 - Later, you will change this so that reports are compared/ordered by day of the week and incident number.
 - Note: I use a somewhat clunky/hacked method of identifying search criteria (creating a "dummyRecord" with only a single criteria on which to match). The proper way to do this would be to use `Comparator` objects or, better yet, the new Java 8 methods of filtering and matching streams. However, these programming concepts are beyond what you are expected to know for our course, so I used the more simpler (but rather inelegant!) approach. Please do not take this as an endorsement of this approach!
- Run and modify these files as described below and submit written answers to the following 7 questions.

1. Run the `ReportProcessor`.

- Q1: How long did it take to build the tree?
- Q2. For each of the 6 dates, list the number of incidents, the time it took to find those incidents using the tree, and the time it took to find those using the list.

2. In the ReportProcessor, comment out Part B-1. Un-comment Part B-2. In the PoliceReport file, comment out the two methods in Part B-1 and uncomment out the two methods in Part B-2.

- You are now comparing and ordering reports by incident number. Incident numbers are unique, so there will no longer be duplicates in your tree.

3. Run the Report Processor.

- Note: if you get an error, go back and make sure you made the changes in commented code in **both** the PoliceReport and ReportProcessor files.
- Q3: How long did it take to build the tree? Why do you think was it faster/slower than in Q1?
 - Hint: think about balance and depth/structure! Consider completing the extra credit now- it might help you answer this question!
- Q4. For each of the 10 incident reports, list the time it took to find the report using the tree and the time it took to find the report using the list.

4. In the ReportProcessor, comment out Part B-2. Un-comment Part B-3. In the PoliceReport file, comment out the two methods in Part B-2 and uncomment out the two methods in Part B-3.

- You are now comparing and ordering reports by day of the week.

5. Run the ReportProcessor.

- Q5: How long did it take to build the tree? Why do you think was it faster/slower than in Q1?
- Q6. For each of the 7 days of the week, list the time it took to find those incidents using the tree and the time it took to find those incidents using the list. Why do you think it was faster/slower than in Q2?
- Hint: think again about balance and depth/structure!

6. In the ReportProcessor, comment out the line marked "??? ITERATIVE" and un-comment out the line marked "??? RECURSIVE."

7. Run the Report Processor.

- Q7: What happens? The number of elements in the tree has not changed. So why does this happen?

Extra Credit (20 Points)

Write a calculateLeftHeight and calculateRightHeight method in BinarySearchTreeWithDups. These methods calculate the height of the left and right subtrees (from the root). You can use recursion or iteration- it is your choice.

Re-run Part B-1 and B-2. List the left and right height of the trees created with these two different ways of comparing/ordering the reports.

Submission


Zip **all** files together and upload them for submission.

- Include the BinarySearchTreeWithDups class.
- Include the driver programs, interface files, and all supporting files in your zip, even though these files were not edited.
- Include your written answers to Part B in the zip.



Submission status

Submission status

Submitted for grading

Grading status	Graded
Due date	Monday, May 15, 2017, 11:55 PM
Time remaining	Assignment was submitted 2 days early
Last modified	Saturday, May 13, 2017, 11:36 PM
File submissions	 LabF by Faustina.zip
Submission comments	► Comments (0)

Feedback

Grade	94.00 / 100.00
Graded on	Monday, May 22, 2017, 5:39 PM
Graded by	 Gregory Gorlen
Feedback comments	<div> Excellent work.</div> <div>-6 in the methods to get all values less than, you can and should use the sorted nature of the tree to only search ...</div>

You are logged in as [Faustina Nyaung](#) (Log out)

CS111C-META-SPRING-2017