**Overall Assignment Description**

This assignment consists of two parts, both dealing with recursion. In the first, you will write a program that recursively searches a directory tree. In the second, you will write a program that implements a binary search of a list of strings.  Below are the descriptions of the two parts.  Here are supporting files:

* [SearchAlgorithm.javaPreview the document](https://canvas.uw.edu/courses/1331881/files/58017706/download?wrap=1)
* [BinSearchDriver.javaPreview the document](https://canvas.uw.edu/courses/1331881/files/58017725/download?wrap=1)
* [longwords.txtPreview the document](https://canvas.uw.edu/courses/1331881/files/58017699/download?wrap=1)

**Part 1:  Recursively Searching Files and Directories**

**Summary**

Build a class and a driver for use in searching your computer’s secondary storage (hard disk or flash memory) for a specific file from a set of files indicated by a starting path. Lets start by looking at a directory listing. Note that every element is either a file or a directory.

**Introduction and Driver**

In this assignment, your job is to write a class that searches through a file hierarchy (a tree) for a specified file. Your **FindFile** class will search a directory (and all subdirectories) for a target file name.

For example, in the file hierarchy pictured above, the file “lesson.css” will be found once in a directory near the root or top-level drive name (e.g. “C:\”) . Your **FindFile** class will start at the path indicated and will search each directory and subdirectory looking for a file match. Consider the following code that could help you build your **Driver**.java:

String targetFile = “lesson.css”;  
String pathToSearch =”  
C:\\WCWC”; FindFile finder = new FindFile(MAX\_NUMBER\_OF\_FILES\_TO\_FIND);  
Finder.directorySearch(targetFile, pathToSearch);

**File Searching**

In general, searching can take multiple forms depending on the structure and order of the set to search. If we can make promises about the data (this data is sorted, or deltas vary by no more than 10, etc.), then we can leverage those constraints to perform a more efficient search. Files in a file system are exposed to clients of the operating system and can be organized by filename, file creation date, size, and a number of other properties. We’ll just be interested in the file names here, and we’ll want perform a brute force (i.e., sequential) search of these files looking for a specific file. The way in which we’ll get file information from the operating system will involve no ordering; as a result, a linear search is the best we can do. We’d like to search for a target file given a specified path and return the location of the file, if found. You should sketch out this logic linearly before attempting to tackle it recursively.

**FindFile Class Interface**

* FindFile(int maxFiles): This constructor accepts the maximum number of files to find.
* void directorySearch(String target, String dirName): The parameters are the target file name to look for and the directory to start in.
* int getCount(): This accessor returns the number of matching files found
* String[] getFiles(): This getter returns the array of file locations, up to maxFiles in size.

**Requirements**

* Your program should be recursive.
* You should build and submit at least two files:  FindFile.java and Driver.java.
* Throw an exception (IllegalArgumentException) if the path passed in as the starting directory is not a valid directory.
* MAX\_NUMBER\_OF\_FILES\_TO\_FIND is the maximum number of files to find for a single call of the method.
* Throw an exception if you've found the MAX\_NUMBER\_OF\_FILES\_TO\_FIND and catch and handle this in your main driver.  Your program shouldn't crash but rather exit gracefully in the unusual situation that we've discovered the maximum number of files we were interested in, reporting each of the paths where the target files were found.
* The only structures you can use in this assignment are basic arrays and your Stack, Queue, or ArrayList from the previous homeworks.  Do not use built-in data structures like Java's ArrayList.  To accomplish this, put in the following constructor and method to your ArrayList, Stack, or Queue:  
  public ArrayList(Object[] input) {  
      data = input;  
      numElements = input.length;  
  }  
    
  public Object get(int index) {  
      return data[index];  
  }

**Notes and Hints**

* Consider looking into the File class in Java for helpful methods like isDirectory() and toString():  [http://docs.oracle.com/javase/7/docs/api/java/io/File.html (Links to an external site.)](http://docs.oracle.com/javase/7/docs/api/java/io/File.html)
* Some elements of the File class you may find useful (taken from the above documentation):
  + import java.io.File: Import the File class to use it
  + File f = new File(dirName): Create a File object
  + String[] fileList = f.list(): This lists files in the current directory of the File object f as Strings
  + File aFile = new File(dirName + "\\" + fileList[i]): Notice the concatenation in the new File object
  + if (aFile.isDirectory()): Checks whether it is a directory
* In your recursive call, make sure you don't lose the directory path by concatenating it with the file name.
* Test your FindFile.java class.  Here are a few tests to try.  These **are not exhaustive**:
  + Try to find a file that exists once.
  + Try to find a file that doesn't exist.
  + Try to find a file that exists twice.
  + Try to find a file that exists MAX\_NUMBER\_OF\_FILES\_TO\_FIND.

**Part 2:  Iterative and Recursive Linear and Binary Searches on an Array**

**Summary**

Build two classes (LinearSearch, BinarySearch) that inherit from the provided parent class SearchAlgorithm, and a third exception class. Each of the subclasses classes will implement only one public method that facilitates the appropriate search over an array of Strings. Also, you will need to build one exception class (10 lines of code or less) that services these two searches in the event the item isn’t found in the String array.

**Introduction**

The best introduction to this homework is probably to start in the BinSearchDriver code that will be used to test your two search classes. Walking through this code, you can see that we will create two objects for use in searching through a large array of words. This array is populated from a sequential text file (longwords.txt), and all of this code is written for you in the driver. Your job is to construct two *subclasses* that derive from the superclass SearchAlgorithm; these two classes will each need to provide a version of the abstract search method declared in SearchAlgorithm.  Note that you will not need to change any of the code in main for the driver, and in fact, should only need to modify the FILE\_AND\_PATH class constant to point at your local version of the input file (longwords.txt). Beyond that, the driver is self-contained, and should compile and execute without modification once your two classes {BinarySearch, LinearSearch} are built and in the same working directory. Lets discuss the two subclasses more in detail next, starting with an item-by-item (linear) search.

**The LinearSearch Class**

This class should loop through the words from beginning to end, comparing the current string with the target string at each step. Note that you must call the utility function incrementCount() each time you make a comparison (ie, each time in the loop) so that when your search is complete, you have an accurate count of the comparisons required by your search strategy. You must implement this iteratively and then attempt to implement this recursively. Report on the results in your submission.

**The BinarySearch Class**

The binary search algorithm can be accomplished in a number of ways, but the basic algorithm is outlined below. You must implement this iteratively and recursively.

* LowIndex = 0
* HighIndex = arraySize – 1
* While LowIndex is less than or equal to HighIndex
  + If the target word is before the word at MidIndex, then set HighIndex to MidIndex - 1
  + If the target word is after the word at MidIndex, then set LowIndex to MidIndex + 1
  + Set MidIndex to be equal to half way between the low and high index
  + If the target word matches the word at MidIndex, return MidIndex (first case)
* If the target word was not found, throw an ItemNotFoundException (you create this class)

**The ItemNotFoundException Class**

This class should be under 15 lines of code, and is an exercise in inheriting from classes provided to you by the Java API. Your class should have only two methods (both constructors) and no data items; see the slides on exceptions for an example of such a class.

**Recursion**

Provide the iterative implementation for the linear and binary search in the search() methods that you override, and put the recursive versions of these same methods in the recSearch() methods, respectively. Note that when you are testing your recursive methods, you must extend the driver to call them since the driver BinSearchDriver.java only calls the method search() in SearchAlgorithm (and derived classes) and does not call recSearch().  However, you will not, as it says in the Overall Hints, Etc. section, be turning in the BinSearchDriver driver that you modified for testing purposes.

**Notes**

* Your classes ***must*** inherit from the SearchAlgorithm class to compile and run.
* SearchAlgorithm tells you what the methods have to return and what the input is.  Note that for the search and recSearch methods, you don't pass in array bounds as in the Savitch Display 11.6 example of a binary search.
* Use the extra utility functions in SearchAlgorithm to track the number of comparisons that have been executed in each search.
* Interested in determining if string1 comes before string2 alphabetically? This ordering can be determined using the compareTo() function; if string1.compareTo(string2) returns a positive int, then string 1 > string2. If negative, then string1 < string2.
* If you fail to find the target word in the set, be sure to throw an ItemNotFoundException
* Classes whose names are given in this assignment description should be given that name.  If you submit multiple versions, Canvas will rename your files with a "-1", etc. suffix, but that's okay.  We can handle that change.  But the base name of your class has to be as described.
* Your three classes will be graded by compiling them with the provided SearchAlgorithm and BinSearchDriver classes *without modification*.  (Please note that this is not a restrictive statement but a minimum statement that we expect your three classes to work with the classes we provide. Another way to say it is that we may use other drivers to test your work (such as the recSearch methods), but that at minimum your classes need to work with the classes we provide, in the form you receive them.)

**Overall Hints, Etc.**

* One of the subprograms you write in one of the above parts will create a stack overflow.  Hint: This is why it says "attempt" and "report on the results" in the pertinent part of the assignment description.
* You can use wrapper methods if you'd like.
* **If the assignment description says to name your classes, variables, or methods a certain name, that is a requirement.** It is a part of the interface definition.  Any commercial software product will have strict definitions of its interface that must be followed.  Otherwise, the software is much less useful.
* Regarding the data structures you use, you can use only arrays if you want, but if you do, you'll have to figure out how to deal with the fact arrays have a fixed size. If the number of elements you're interested in goes beyond that size, your program should handle that automatically.
* Handling errors:  At this point, you should **not** be handling errors by calling System.exit.  All errors should be handled by throwing exceptions.  These exceptions should not be caught unless it makes sense to do so at that level, and the result of a catch should **not** be termination of the program.
* Whenever there are lines of code estimates given, they refer to lines of executable code; commenting is not included in those estimates.
* Remember to follow the Coding Style Guidelines.
* Your code should not depend on absolute path specifications or any other environment-specific specifications.  The program should **not** ask for user input from the console.  Remember to reference any files by the names we've given them (e.g., longwords.txt).  However, for the FindFile search tests, we recognize that you will be testing on your own computer's directory tree and that the root you specify will be an environment-specific name.
* **Make sure you comment all methods and the class with javadoc comments.**  This includes constructors, getters, setters, etc.  If you have files based on code someone else has written (e.g., is based on a skeleton), you have to javadoc comment the methods someone else wrote also.  You cannot receive full-credit for the rubric's commenting criterion if you do not have a **javadoc comment for every method,** no matter how small.

**What to Submit**

* You will upload all the Java files for both parts of the assignment in this one assignment.
* Note that in Part 1, you are turning in, at a minimum, two files named FindFile.java and Driver.java. Thus, **I am assuming your file search driver program is named Driver.java.** Do not give your binary search driver the same name.
* In fact, **you should not be writing a driver at all for your binary search program.** As it details in the description, your binary search program should work with the provided driver. Thus, **do not upload a binary search driver.** You should also not upload the SearchAlgorithm class file, as that will also be used without modification.

Please **submit your files inside a zip archive called Recursion.zip.**

**About This Document**

Original assignment by Rob Nash, Autumn 2014. Minor edits and additions by Johnny Lin, Spring 2017-September 2019.

**Rubric**

Recusion HW Rubric (1)

| Recusion HW Rubric (1) | | |
| --- | --- | --- |
| **Criteria** | **Ratings** | **Pts** |
| This criterion is linked to a Learning Outcome Clean and Descriptive Comments |  | 8.0 pts |
| This criterion is linked to a Learning Outcome File Search Code Compiles and Properly Executes |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Binary Search Code Compiles and Properly Executes |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Programs Use Recursion |  | 8.0 pts |
| This criterion is linked to a Learning Outcome File Search: Throws Exceptions when MAX\_NUMBER of Files Is Found and Exceptions Are Properly Handled |  | 2.0 pts |
| This criterion is linked to a Learning Outcome File Search: FindFile Class |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Binary Search: LinearSearch Subclass |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Binary Search: BinarySearch Subclass |  | 4.0 pts |
| This criterion is linked to a Learning Outcome Binary Search: ItemNotFound Exception Subclass |  | 4.0 pts |
| Total Points: 42.0 | | | |

**Submission**

Submitted!

Nov 7 at 11:57pm

[Submission Details](https://canvas.uw.edu/courses/1331881/assignments/4923158/submissions/3729452)

[Download Recursion.zip](https://canvas.uw.edu/courses/1331881/assignments/4923158/submissions/3729452?download=59660036)

Grade: 42 (42 pts possible)

Graded Anonymously: no

[View Rubric Evaluation](https://canvas.uw.edu/courses/1331881/assignments/4923158/submissions/3729452#rubric)

**Comments:**

Nice work!