ECE 242: Data Structures and Algorithms- Fall 2015

Project 2: FunWithWords

Due Date: **11:50pm on October 7, 2015**

Description

This project is intended to familiarize you with operations on arrays such as Sorting and Searching Algorithms. The goal of the project is to design an efficient dictionary database and implement a variety of useful applications such as: searching a word, spell checking a text, or find anagrams of a word. The items of the dictionary will be read from an input file. At first they are not ordered and you must then implement and test the performance of two sorting algorithms: (i) insertion sort and (ii) enhanced insertion sort. A shuffle algorithm that generates random permutations of the dictionary items, must also be implemented in order to allow multiple testing and timings of the sorting algorithms. At the first execution the code output include a menu containing all options:

```
>java DictionaryApplication

Welcome to the Dictionary App
=============================

Current Dictionary is 'N/A' of size 0 and unsorted

Menu
====
1-Load dictionary file
2-Sort dictionary using Insertion sort
3-Sort dictionary using Enhanced Insertion sort
4-Shuffle dictionary
5-Search word
6-SpellChecker
7-Anagram
0-Exit

Command:
```

All the functionalities of the DictionaryApp are presented and discussed in details in the following.

1- Load Dictionary

At first there is no dictionary loaded, so the name of the database is set to “N/A”, its size to “0” and it is unsorted by default. If the option “1” is selected, the code will ask for the name of the file that contains the database. The project comes with two files:

1. a short database “shortdb.txt” of only 10 words (good for testing/debugging)
2. a much larger database “largedb.txt” that contains 99,171 words (for production).

The file database looks like (for example 'shortdb.txt'):

```
10
dictionary
simon
```

```
crazy
of
animal
barter
this
cases
file
code
```

It is important to note that the first row contains the number of words in the database. This dictionary is not sorted. By default when the dictionary is loaded with option 1, it is always considered "unsorted". Here the screenshot after option 1 is selected and the name of the file is entered.

```

Command: 1
Enter dictionary filename:
shortdb.txt
<<OK dictionary shortdb.txt loaded >>


Current Dictionary is 'shortdb.txt' of size 10 and unsorted

Menu
====
1-Load dictionary file
2-Sort dictionary using Insertion sort
3-Sort dictionary using Enhanced Insertion sort
4-Shuffle dictionary
5-Search word
6-SpellChecker
7-Anagram
0-Exit
```

Option 1 is a requirement for all other options. The code will not accept other options if the dictionary is not loaded, for example:

```

Welcome to the Dictionary App
=============================

Current Dictionary is 'N/A' of size 0 and unsorted

Menu
====
1-Load dictionary file
2-Sort dictionary using Insertion sort
3-Sort dictionary using Enhanced Insertion sort
4-Shuffle dictionary
5-Search word
6-SpellChecker
7-Anagram
0-Exit

Command: 5
```

```
>>>>>>>>> Dictionary must be loaded first <<<<<<<<<
Selection Invalid!- Try Again
```

2- Sort dictionary using Insertion sort

The title option says it all. In addition to the action of sorting the words in the dictionary, the code will return the time taken by the insertion sort algorithm (in milliseconds) and the sorted dictionary will be saved into a new file (for example 'shortdb.txt-sorted' if the dictionary file is 'shortdb.txt'). Here is a sample output after option 2 is selected.

```
Command: 2
<<OK dictionary sorted in 0ms and save in file 'shortdb.txt-sorted' >>


Current Dictionary is 'shortdb.txt' of size 10 and sorted

Menu
====
1-Load dictionary file
2-Sort dictionary using Insertion sort
3-Sort dictionary using Enhanced Insertion sort
4-Shuffle dictionary
5-Search word
6-SpellChecker
7-Anagram
0-Exit

Command:
```

Note also the status of the dictionary in the line 'Current Dictionary...' that changed from 'unsorted' to 'sorted'. Finally, options 5, 6, 7 cannot be executed if the dictionary is not sorted first. For example:

```
Command: 5

>>>>>>>>> Dictionary must be sorted first <<<<<<<<<
Selection Invalid!- Try Again
```

3- Sort dictionary using Enhanced Insertion sort

This option should provide the same output and features than Option 2 (timing, saving the sorted dictionary, etc.). The only difference lies in the sorting algorithm. The new algorithm is based on the same idea as insertion sort: insert the new element into the proper position of already sorted elements. However, it uses a binary search instead of a linear search to identify the position of the item to be inserted. The basic of the enhanced insertion sort algorithm is presented further in this document. This algorithm is expected to run faster than insertion sort.

4- Shuffle dictionary

You will be using the Fisher-Yate algorithm to generate random permutation of the items. The new "unsorted" dictionary will be saved into a new file (for example 'shortdb.txt-unsorted' if the dictionary file is 'shortdb.txt'). Once this option is selected, the output should look like:

```
Command: 4
<<OK dictionary shuffled and save in file 'shortdb.txt-unsorted' >>


Current Dictionary is 'shortdb.txt' of size 10 and unsorted

Menu
====
1-Load dictionary file
2-Sort dictionary using Insertion sort
3-Sort dictionary using Enhanced Insertion sort
4-Shuffle dictionary
5-Search word
6-SpellChecker
7-Anagram
0-Exit

Command:
```

Note that the status of the dictionary is changed to 'unsorted' if it was 'sorted' before.

5- Search word

Once the array is sorted, you will use a binary search to search the dictionary for the specific word of your choice. If the word is found, the code will return the position of the word in the ordered array as well as the number of steps it took to get it. The code will let you know if the word is not found as well. A couple of examples below:

```
Command: 5
Enter word to search:
code
<<OK word found>> using Binary search in 4 steps and at position 3
```

```
Command: 5
Enter word to search:
coding
word not found !
```

6- SpellChecker

Using the sorted dictionary, this option allows the user to check the spelling of all the words in a text file of his/her choice. Three example text files are included for testing: (i) letter.txt, (ii) song.txt, (iii) poem.txt. Once this option is selected, the user is asked to enter the name of the text file. The code will check the spelling of all the words (line by line and along the lines of the file). If a word is not found in the dictionary, it will flag it as incorrectly spelled and return it into brackets '(' ')'. Obviously this spellchecker needs to rely on a large dictionary such as the one from 'largedb.txt' You may still find that there are common words that are not found in the dictionary and thus will be flagged as incorrectly spelled. Do not worry about this. However make sure to remove periods and commas from words before trying to find them in the dictionary. You will also decapitalize the words (using lowercase) to help the search. More details on how to proceed will be provided further in the document. Some examples (using the sorted 'largedb.txt' dictionary):

```
Command: 6
Enter text filename:
letter.txt

Dear (Studants,)
this is a sample file similar to the one which will
be used to test your projects. I (sugest) you use it
for testing your code and (developping) your software.
Have fun!
E. (Polizzi)
```

```
Command: 6
Enter text filename:
song.txt

Wonderful World -a bit modified-
by (Sam) (Cooke)

Do not know much about (hystori)
Do not know much (biologi)
Do not know much about (sience) book
Do not know much about the French I took

But I do know that I love you
And I know that if you love me too
What a (wanderful) world this would be

Do not know much about (geographi)
Do not know much (trigonometri)
Do not know much about (halgebra)
Do not know what a slide rule is for

But I do know one and one is two
And if this one could be with you
What a (wanderful) world this would be

Now, I (don't) claim to be an A student
But (I'm) trying to be
For maybe by being an A student, baby
I can win your love for me

(...)

La (ta) (ta) (ta) (ta) (ta) (ta,) History
(Hmm-mm-mm,) Biology
La (ta) (ta) (ta) (ta) (ta) (ta,) Science book
(Hmm-mm-mm,) French I took

(...)
```

```
Command: 6
```

```
Enter text filename:
poem.txt

Can’t
by (Edgar) A. Guest

Can’t is the worst word (that’s) written or spoken;
Doing more harm here than slander and lies;
On it is many a strong spirit broken,
And with it many a good purpose dies.
It springs from the lips of the thoughtless each morning
And robs us of courage we need through the day:
It rings in our ears like a (timely-sent) warning
And laughs when we falter and fall by the way.

Can’t is the father of feeble endeavor,
The parent of terror and half-hearted work;
It weakens the efforts of artisans clever,
And makes of the toiler an indolent shirk.
It poisons the soul of the man with a vision,
It stifles in infancy many a plan;
It greets honest toiling with open derision
And mocks at the hopes and the dreams of a man.

Can’t is a word none should speak without blushing;
To utter it should be a symbol of shame;
Ambition and courage it daily is crushing;
It blights a man’s purpose and shortens his aim.
Despise it with all of your hatred of error;
Refuse it the (lodgment) it seeks in your brain;
Arm against it as a creature of terror,
And all that you dream of you some day shall gain.

Can’t is the word that is foe to ambition,
An enemy ambushed to shatter your will;
Its prey is forever the man with a mission
And bows but to courage and patience and skill.
Hate it, with hatred (that’s) deep and undying,
For once it is welcomed ’twill break any man;
Whatever the goal you are seeking, keep trying
And answer this demon by saying: "I can."
```

7- Anagram

This last option allows the user to enter a word or a set of character/letters (like in the scrabble game), and the code is supposed to return all the words that can be formed using permutations of the letters (looking at your dictionary of course). More details and hints are provided further on how to proceed. The results should look like:

```
Command: 7
Enter word to analyze:
tea

ate
eat
eta
```

```
tea
```

```
Command: 7
Enter word to analyze:
eruipt
```

Note that no word has been found in the latter case.

REQUIREMENT- WHAT TO DO

1. Complete the code. Skeletons of the DictionaryApplication class and FastDictionary class, are provided. Since the emphasis of the project is more algorithms than data structures, all needed methods have been included within the class FastDictionary for simplicity.

2. In a separate 'one-page' document that should be submitted as a **pdf** file:

   - perform the analysis of the enhanced insertion sort. You will provide the big-O complexity for the number of comparisons, the number of shifts, and for the whole algorithm. You will compare these complexities with the ones obtained with insertion sort.
   - For the large 'largedb.txt' database and both insertion and enhanced insertion sort algorithms, provide a table showing multiple timing runs for the unsorted array (you can use repeatedly option 4 to shuffle the array), the average time of these tests (mean), and the time it would take if the array is already ordered to start with. Example of table layout:

Time (ms)	Test1	Test2	Test3	Test4	Test5	Mean	Ordered
Insertion Sort							
Enhanced Insertion Sort							
Speed-up							

     The last row is the speed-up factor between the times of both algorithm speed-up=(Time insertion sort)/(Time Enhanced insertion sort)
   - Provide an 'informative' plot of N (number of items) vs Time for both algorithms (only one plot). To obtain these results, you could load multiple times the dictionary 'largedb.txt' while changing the first row of the file (number of items). I suggest you use N=1000, N=2000, N=4000, N=8000, N=16,000, N=32,000, N=64,000, N=99,171

3. Submit a single zip file composed of: All your source files (*.java) including EasyIn.java, all (*.txt) files, your README file, your "pdf" report file.

COMPLEMENTS and HINTS

1. You may use the `EasyIn.java` class for input.

2. The dictionary database must be read from file line by line, you could take inspiration from the first method in the ReadEasy.java class provided with Project 1.

3. The text to test the SpellChecker must be read from file line by line and along the lines, you could take inspiration from the second method in the ReadEasy.java class provided with Project 1.

4. In order to save data into a file: here is an example on how to write a String and an integer into the file 'lecture':

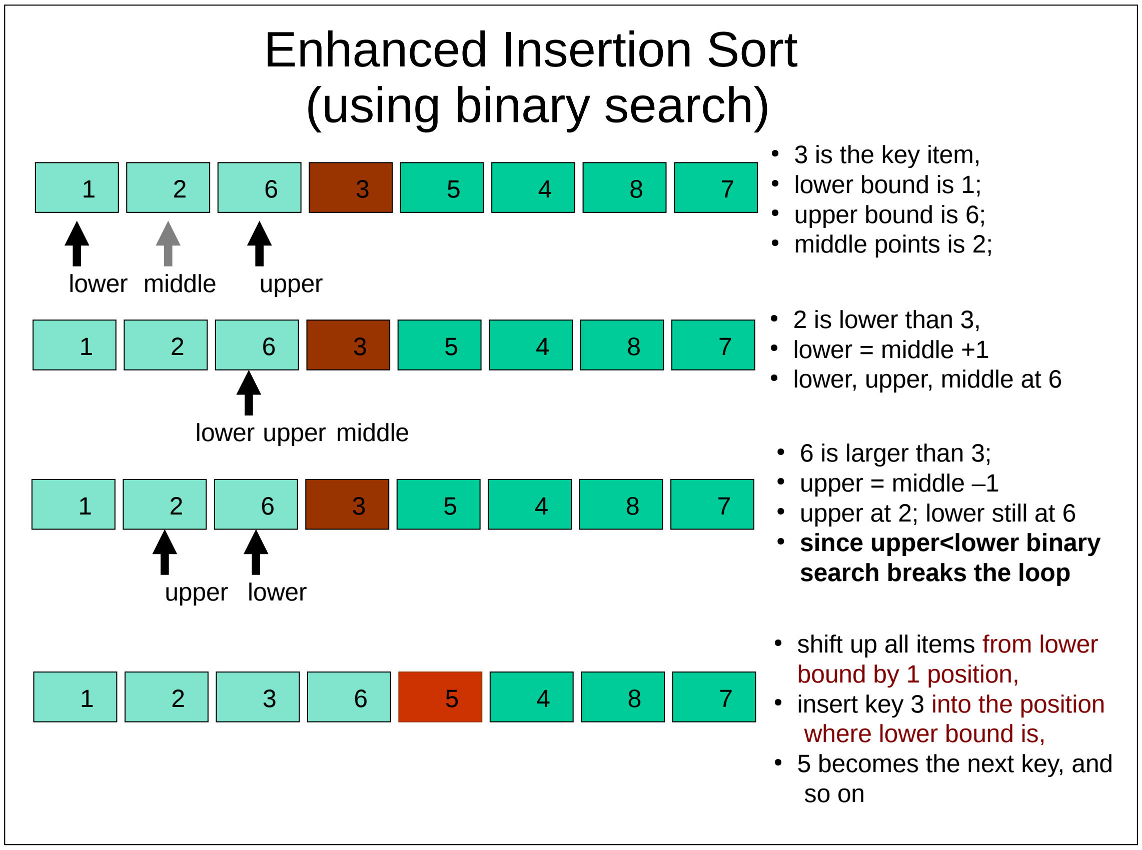
```
String course=''ECE''
int n=242;
try {
            PrintWriter out = new PrintWriter(``lecture'');
            out.println(course);
            out.println(n);
            out.close();
        }
        catch (IOException e)
           {
               e.printStackTrace();
            }
    }
```

5. For the SpellChecker, you can use the following command to remove the punctuation within a String "word" and transform it into lowercase.

```
String word1=word.replaceAll("[^\\p{L}]", "").toLowerCase();
```

6. For the anagram, one could proceed by first obtaining all permutations of your word, and search each one of them in the dictionary. However, this is rather involved and will see it later in class. For this project, the following solution is preferred: (i) sort the characters in your key word using your "insertion sort" routine already implemented (for example: the word 'dbca' will give 'abcd'); (ii) scan through all the dictionary for words that have the same String length than your key word, (iii) if found, sort the corresponding word by characters and compare with your sorted key word.

7. One of the main objective of this project is to implement the enhanced insertion sort. The figure below provides an overview of the algorithm (single step) while considering an array of integer.



8. Finally, you will implement a 'single main' binarysearch method (you could define various binarySearch methods but the "main work" should appear in only one of them). This binarysearch method is called from multiple other methods in the code including the enhanced insertion sort method.

Submissions

You will work in groups of two (working alone is accepted but not recommended). **Each group should submit one copy of the solution. Please do not forget to include the name of both partners!** Include the **name**, **student ID**, and **email address** of both partners. This information should be included in a README file. See comments above regarding all other files that need to be included into a single zip file: source, pdf report, etc.

Grading Policy

This project will be graded out of 100 points:

1. Your program should compile successfully. (20 points)
2. Your program should implement basic functionality and run correctly. (60 points)
3. Overall programming style: source code should have proper identification, and comments. (20 points)

Extra Credit

You can earn 10 additional points by extending the functionality of your dictionary. For the SpellChecker, once the word is flagged you could propose a "fast" automatic correction (replacement with the "closest meaningful" word). Example: "Studant" into "Student"; "sugest" into "suggest", etc. No help from TA or Professor will be provided for the extra credit.