

# Assignment 3

## Sorting Analysis and Anagrams

CSIS 3475

©Michael Hrybyk and others  
NOT TO BE REDISTRIBUTED

# Assignment

- Download **Assignment 3.zip** and import it into an Eclipse workspace using the standard instructions.
- Add your completed implementations of all classes in the ListPackage (AList, LList) and HashingPackage.
- Complete the code in Task1\_SortingAlgorithmEvaluation package
- Complete the code in Task2\_Anagrams package.
- Submit the project using the standard submission instructions
  - Export the projects to a zip archive named **Assignment3 YourName.zip** where YourName must be your first initial and last name.
  - You **MUST** use the submission instructions exactly or you will lose marks.
  - You **MUST** name the archive correctly or you will lose marks.
  - For example, for Michael Hrybyk
    - Assignment3 MHrybyk.zip

# Task1\_SortingAlgorithmEvaluation

- Analyze the amount of time each sorting algorithm takes.
  - Selection Sort
  - Insertion Sort
  - Shell Sort
  - Merge Sort
  - Quick Sort
- All of the implementations can be found in `SortUtilities.java`

# Task1: SortMethodEvaluation.java

- Given a list of exponentially increasing numbers, run and time each sort algorithm implementation on a list.
- Basic loop and logic is provided for Selection Sort.
- Copy the code segment and perform for all other sort implementations.
  - for each implementation
    - first copy the original list of random numbers to the test list
    - set the starting time
    - run the sort method on the test list
    - get the end time
    - display the elapsed time result
    - run the sort method again on the test list that is now sorted, and display elapsed time result.
- Output must correspond to that found in **Sample Timed Output.txt**
  - Given that the lists will contain strings of numbers of slightly different string lengths, and differences in computer clocks, the times will not be exactly the same.

# Task1 – Complete the analysis

- In the file **Sorting Analysis.txt**, complete answers to all questions
- Explain which algorithms have the best performance. Why are there differences in times?
- Explain which algorithms have the best performance on sorted lists. Explain the differences from non-sorted lists.
- How does the time increase for each algorithm as the list size increases? Explain why.
- The sorting utility methods use the list interface methods for AList and LList.
  - How does this affect the performance of each sorting algorithm?
  - What would you do to improve the performance of the sorting methods?
- If you substitute LinkedList in ListUsingLibraryLinkedList, things slow down a lot. Similarly, if you use LList instead of AList.
  - Why is this?
- Show the output from one of your test runs in separate text files
  - AListTimedSortOutput.txt
  - LibraryListSortOutput.txt

# Task2\_Anagrams

- An anagram is a word in a spoken language dictionary that has the same letters as another word in that dictionary.
  - For example, **beard**, **bread**, and **bared** are anagrams of each other.
  - See <https://en.wikipedia.org/wiki/Anagram>
- Given a dictionary of words, create a hash table of keys and values for anagrams in the dictionary.
  - key: set of letters (in order)
  - value: list of all words that are anagrams for that key
- How?
  - For each word in the dictionary
    - sort the letters in the word
    - use this sorted set of letters as the key
    - add the word to the list of anagrams for that key
  - Use a hash table for this
  - Example – bad
    - key: abd
    - add bad to the list for this key

# Task2 - HashMapDictionary

- Complete HashMapDictionary implementing the methods from DictionaryInterface
  - Use the methods from the java library HashMap on the private HashMap hashTable to implement the interface.
    - You will need to research HashMap methods. They are similar to those in DirectoryInterface
  - For getKeyIterator(), use HashMap's keyset().iterator()
  - For getValueIterator(), use HashMaps values().iterator()
- Make sure **all** methods in HashMapDictionary work properly using HashedDictionaryTestDriver.java.
  - Test the iterator methods as well.
- Hint: each HashMapDictionary method only needs one line of code using HashMap library methods.

# Task2 - HashedDictionary

- Complete HashedDictionary implementing the methods from DictionaryInterface
  - Use hashing to place each Entry in an array. An internal array must be used.
  - This is similar to what was provided in class notes and the textbook.
  - You are provided with the constructor and other helper methods.
  - You must use linear probing for the hash table
    - Complete the linearProbe() method and use it.
  - For the add() method, you will need to enlarge the hash table if there is not enough room. Use the helper methods to do this.
  - Make sure you also implement the getKeyIterator() and getValueIterator() methods.
    - Only need to provide hasNext() and next().
- Use HashedDictionaryTestDriver.java to make sure **all** methods in HashedDictionary work properly.
  - Test the iterator methods as well.



# Task2 – Anagrams program

- Once you have HashMapDictionary and HashedDictionary classes written and tested, you must complete the AnagramsUsingDictionaryInterface program in the Task2\_Anagrams package.
- You are given an array of test words to be used. Feel free to add more words to the list for testing.
- Complete the method createAnagrams().
  - First argument is the file containing the list of words in the dictionary, one on each line – **dictionary.txt**
  - Second argument is the DictionaryInterface object that is used to store the anagrams
- After createAnagrams() is called and is successful, output all keys and values from the DictionaryInterface object to **anagrams.txt**.
  - Use the PrintWriter.println() method to output each key and value.
  - Use the DictionaryInterface iterators to move through the hash table.
  - DO NOT USE THE DISPLAY HASH TABLE METHOD IN HashedDictionary. ONLY USE THE ITERATOR METHODS YOU HAVE IMPLEMENTED.
  - The output should match the **anagramsAssignmentOutput.txt** file exactly
  - The sortLetters() helper method is provided – the output of this method will serve as the key for a word.
- Thoroughly test the program with the HashPackage classes and document ALL code.

# Grading

Item	Marks
Assignment properly named and submitted. All code properly formatted and commented	0.2
Sorting Analysis	1.8
HashingPackage ADT implementations	1.0
Anagrams program	2.0
Total	5.0