University of Windsor

School of Computer Science

60-654 Advanced Computing Concepts - Fall 2017
Lab Assignment 2

Deadline: **October 30, 2017 at 11:59pm.**

Rules: This assignment must be submitted and will be evaluated individually. You are allowed to discuss the problems and some ideas within a group or with other students. However, the source code for the programs should not be identical among those students.

You are required to use Java SE 8 and Eclipse to write your programs and the data structures and implementations as discussed in class.

Objectives: The aim of this assignment is to help understand different types of sorting algorithms and selection, and the main concepts of algorithm design, including recursion, divide and conquer, and dynamic programming. Students will obtain hands-on experience in evaluating sorting algorithms and comparing strings using edit distance, as well as the use of Java SE 8 and Eclipse.

Tasks:

1. Use class Sort.java provided in class, the dual-pivot Quicksort of Java 8 (Arrays.sort), and RadixSort.java provided in class.
2. Do the following for Mergesort, Quicksort, Heapsort and dual-pivot Quicksort:
   a. Create 100,000 random keys (of type long) and sort them. Repeat this 100 times.
   b. Compute the average CPU time taken to sort the keys for the four methods.
   c. Comment on the results and compare them to the average-case complexities discussed in class.
3. Do the following for the four sorting methods of #2, and for Radix sort:
   a. Create 100,000 random strings of length 4 and sort them using the five sorting methods.
   b. Repeat (a) 10 times and compute the average CPU time that takes to sort the keys for the five methods.
   c. Repeat (a) and (b) with strings of length 6, 8, 10.
   d. Create a table with the results and compare the times with the average-case and worst-case complexities as studied in class.
4. Comment on: which sorting method will you use in your applications? in which case? Why?
5. Use the edit distance (class Sequences.java) implementation provided in the source code.
   a. Generate 1,000 pairs of random words of lengths 10, 20, 50 and 100.
   b. Compute the edit distance for all words and find the average CPU time for each pair.
   c. Compare the CPU times obtained for each word length with the running times of the edit distance algorithm.

Submission:

1. You must submit (i) a report (in PDF or word), which contains the answers to all questions, (ii) all Java files and classes needed to run your programs. Your programs should run in Java SE 8 in the Eclipse IDE. You should upload all these files to the Blackboard, as a **single** zip file.
2. In your report, you must provide written answers to all questions, including the programs' outputs, as required. Marks will be deducted if comments/explanations are missing.
3. Add the following note at the beginning of your report:
   "I confirm that I will keep the content of this assignment confidential. I confirm that I have not received any unauthorized assistance in preparing for or writing this assignment. I acknowledge that a mark of 0 may be assigned for copied work." + Name + SID
4. Any submission after the deadline will receive a penalty of 10% for the first 24 hrs, and so on, for up to three days. After three days, the mark will be *zero*.
5. Unlimited resubmissions are allowed. But keep in mind that we will consider the last submission. That means that if you resubmit after the deadline, a penalty will be applied, even if you submitted an earlier version in time.

Real application - challenge (optional):

1. Web dictionary: Write a program/method that takes all 100 HTML files as input, and creates a dictionary using a binary search tree or BST (you can use an AVL or a red-black tree).
2. Using the BST as input, sort the entries in decreasing order of frequency. Find the 10 most and least frequent words and list them in decreasing order of frequency.
3. Some stats: Use BinaryHeap.java from the practice source code and modify it so that each entry in the dictionary contains the word and its frequency. Write a program that takes the BST as input and creates a heap in which the key is the frequency (you may want to create it using the bottom-up approach). Show the 10 most frequent words in the Web files.
4. The spellchecker: Write a program that takes an HTML file as input and checks whether or not each word in that file is the dictionary (BST). Insert in the dictionary the words that are not found – provide a list of these words.