Data Structures and Algorithms – (COMP SCI 2C03)
Winter, 2021
Assignment-II

Due at 11:59pm on March 15th, 2021

- **No late assignment accepted.**
- Make sure to submit a version of your assignment ahead of time to avoid last minute uploading issues.
- Submit one assignment solution as a PDF file per group on Avenue.
- If the solution submitted by two groups is the same. Both teams will get a zero mark on the assignment.
- Present your algorithms in Java or Pseudocode (Pseudocode is preferred).
- It is advisable to start your assignment early.

This assignment consists of 8 questions, and is worth 55 marks.

1. Modify BinarySearchST (Algorithm 3.2 in textbook on page 379) so that inserting a key that is larger than all keys in the table takes constant time (so that building a table by calling put() for keys that are in order takes linear time). [6 marks]

2. Given a standard binary search tree (BST) (where each key is greater than the keys in its left subtree and smaller than the keys in its right subtree), design a linear-time algorithm to transform it into a reverse BST (where each key is smaller than the keys in its left subtree and greater than the keys in its right subtree). The resulting tree shape

should be symmetric to the original one. It is not required of you to output the tree from your code (if you implement it in JAVA). Marks will be given based on the logic and correctness of the algorithm. [7 marks]

3. Give a non-recursive algorithm that performs an inorder tree walk for a tree containing n nodes. The algorithm should run in $O(n)$ time. [6 marks]

4. What is the largest possible number of internal nodes in a red-black tree with black height k? What is the smallest possible number? [5 marks]

5. **Bonus Question:** Show that red-black BSTs are not memoryless: for example, if you insert a key that is smaller than all the keys in the tree and then immediately delete the minimum, you may get a different tree. [7 marks]

6. (a) Modify LinearProbingHashST (given on page 470 of the textbook) to use a second hash function to define the probe sequence. Specifically, first replace (i + 1) % M (both occurrences) by (i + k) % M where k is a nonzero key-dependent integer that is relatively prime to M (it is easier to choose M as prime) - $(i + k)\%M$ would be your first hash function $h_1(k)$. Then, device a second 'good' hash function $h_2(k)$ to identify successive probe positions. You are not required to work on this question in JAVA. Simply modifying the `hash`, `put` and the `get` functions to accommodate double hashing will suffice. However, for those interested in implementing it in JAVA, the code for LinearProbingHashST can be found at - `https://algs4.cs.princeton.edu/34hash/LinearProbingHashST.java.html` [7 marks]

   Alternate version of the question: Modify LinearProbingHashST (given on page 470 of the textbook) to use a second hash function to define the probe sequence. Specifically, first replace (i + 1) % M (both occurrences) by $(h_1(k) + ih_2(k))\%M$ where k is a nonzero key-dependent integer that is relatively prime to M (it is easier to choose M as prime). $h_1(k)$ would be your first hash function. You may choose to use the $hash()$ function defined on page 470 of the textbook as $h_1(k)$. You are then required to

device a second 'good' hash function $h_2(k)$ to identify successive probe positions. You are not required to work on this question in JAVA. Simply modifying the `hash`, `put` and the `get` functions to accommodate double hashing will suffice.

(b) Give a trace of the process of inserting the keys E A S Y Q U T I O N in that order into an initially empty table of size M =11, using the hash functions described in 6(a). For this question, since the keys are drawn from the English alphabet, you may choose a mapping from the set of English Alphabets to natural numbers and then insert the keys in the hash table. For instance, you choose the following mapping $MAP : \{A, B, \ldots, Y, Z\} \rightarrow \{1, 2, \ldots 25, 26\}$. Alternatively, it is also acceptable to use the hashcode() method for the keys provided in JAVA (`key.hashCode()`) as shown on page 470.[5 marks]

7. Prove that every connected graph (with at least two vertices) has a vertex whose removal (including all adjacent edges) will not disconnect the graph, and write a DFS method that finds such a vertex. Hint : Consider a vertex whose adjacent vertices are all marked. [6 marks]

8. Explain why the following algorithm does not necessarily produce a topological order: Run BFS, and label the vertices by increasing distance to their respective source. [6 marks]