COMP 6481: Programming & Problem Solving
Fall 2021

Department of Computer Science and Software Engineering
Concordia University

Assignment # 3

Due date and time: Sunday December 5th , 2021 by 11:00 AM (MORNING)
The due date is sharp and strict. NO EXTENSION WILL BE ALLOWED BEYOND THIS TIME!

Part I

Please read carefully: You must submit the answers to all the questions below. However, this part will not be marked. Nonetheless, failing to submit this part fully will result in you missing 50% of the total mark of the assignment.

Assignment MUST be typed. However, you are *allowed* to hand-draw any images and incorporate them into your typed submission. Hand-drawn images must be very clear; otherwise they will be discarded by the markers. All text must be typed and not hand-written.

Question 1

Assume the utilization of *linear probing* for hash-tables. To enhance the complexity of the operations performed on the table, a special *AVAILABLE* object is used. Assuming that all keys are positive integers, the following two techniques were suggested in order to enhance complexity:

i) In case an entry is removed, instead of marking its location as AVAILABLE, indicate the key as the negative value of the removed key (i.e. if the removed key was 16, indicate the key as -16). Searching for an entry with the removed key would then terminate once a negative value of the key is found (instead of continuing to search if AVAILABLE is used).

ii) Instead of using AVAILABLE, find a key in the table that should have been placed in the location of the removed entry, then place that key (the entire entry of course) in that location (instead of setting the location as AVAILABLE). The motive is to find the key faster since it now in its hashed location. This would also avoid the dependence on the AVAILABLE object.

Will either of these proposal have an advantage of the achieved complexity? You should analyze both time-complexity and space-complexity. Additionally, will any of these approaches result in misbehaviors (in terms of functionalities)? If so, explain clearly through illustrative examples.

Question 2

Show the steps that a radix sort takes when sorting the following array of 3-tuple Integer keys (notice that each digit in the following values represent a key):

783 992 472 182 264 543 356 295 692 491 947

Question 3

Draw the binary search tree whose elements are inserted in the following order:

50 72 96 94 107 26 12 11 9 2 10 25 51 16 17 95

Question 4

Describe an efficient algorithm for computing the height of a given AVL tree. Your algorithm should run in time *O(log n)* on an AVL tree of size *n*. In the pseudocode, use the following terminology: `T.left, T.right`, and `T.parent` indicate the left child, right child, and parent of a node T and `T.balance` indicates its balance factor (-1, 0, or 1).

For example if T is the root we have `T.parent=nil` and if T is a leaf we have `T.left` and `T.right` equal to nil. The input is the root of the AVL tree. Justify correctness of the algorithm and provide a brief justification of the runtime.

Question 5

Given the following elements:

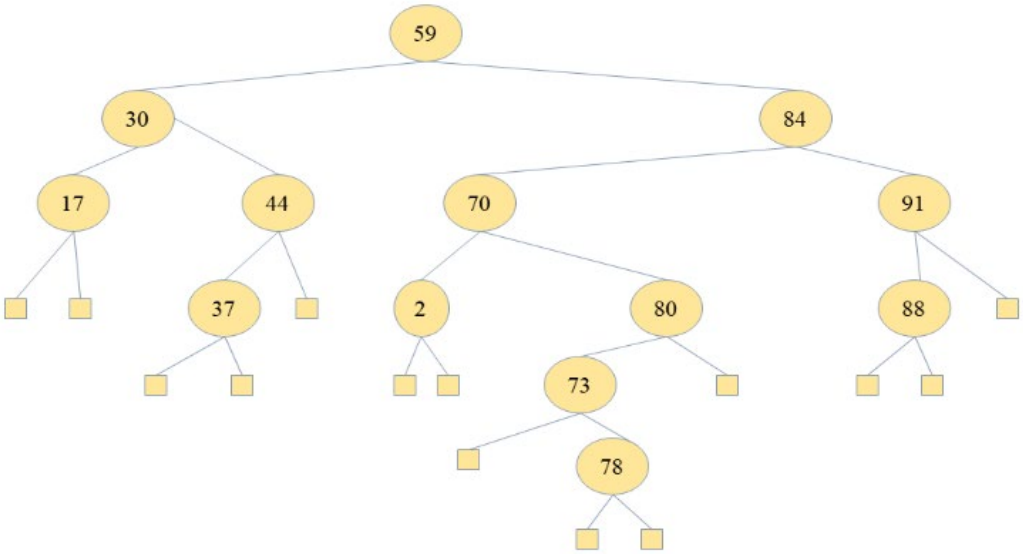
29 38 74 78 24 75 42 33 21 62 18 77 30 16

Trace the steps when sorting these values into ascending order using:

a) Merge Sort,
b) Quick Sort (using (middle + 1) element as pivot point),
c) Bucket Sort – We know that the numbers are less than 99 and there are 10 buckets.

Question 6

Given the following tree, which is assumed to be an AVL tree:



i) Are there any errors with the tree as shown? If so, indicate what the error(s) are, correct these error(s), show the corrected AVL tree, then proceed to the following questions (Questions ii to iv) and start with the tree that you have just corrected. If no errors are there in the above tree, indicate why the tree is correctly an AVL tree, then proceed to the following questions (Questions ii to iv) and continue working on the tree as shown above.

ii) Show the AVL tree after ***put(74)*** operation is performed. What is the complexity of this operation?

iii) Show the AVL tree after ***remove(70)*** is performed. What is the complexity of this operation?

iv) Show the AVL tree after ***remove(91)*** is performed. Show the progress of your work step-by-step. What is the complexity of this operation?

Part II: Programming Questions (50 marks):

The North American Student Tracking Association (NASTA) maintains and operates on multiple lists of *n* students. Each student is identified by a **unique** 8-digit code, called StudentIDentificationCode (SIDC); (e.g. # SIDC: 47203235). Some of these student lists are local to villages, towns and remote areas, where *n* counts only to few hundred students, and possibly less. Others are at the urban cities or provincial levels, where *n* counts to tens of thousands or more.

NASTA needs your help to design a clever "student tracking" ADT called **CleverSIDC**. Keys of CleverSIDC entries are long integers of 8 digits, and one can retrieve the keys/values of an CleverSIDC or access a single element by its key. Furthermore, similar to *Sequences*, given a CleverSIDC key, one can access its predecessor or successor (if it exists).

CleverSIDC adapts to their usage and keep the balance between memory and runtime requirements. For instance, if an CleverSIDC contains only a small number of entries (e.g., few hundreds), it might use less memory overhead but slower (sorting) algorithms. On the other hand, if the number of contained entries is large (greater than 1000 or even in the range of tens of thousands of elements), it might have a higher memory requirement but faster (sorting) algorithms. CleverSIDC might be almost constant in size or might grow and/or shrink dynamically. Ideally, operations applicable to a single CleverSIDC entry should be $O(1)$ but never worse than $O(n)$. Operations applicable to a complete CleverSIDC should not exceed $O(n^2)$.

You have been asked to **design** and **implement** the CleverSIDC ADT, which automatically adapts to the dynamic content that it operates on. In other words, it accepts the size (total number of students, *n*, identified by their 8 digits SIDC number as a key) as a parameter and uses an appropriate (set of) data structure(s), or other data types, from the one(s) studied in class in order to perform the operations below efficiently[1]. You are NOT allowed however to use any of the built-in data types (that is, you must implement whatever you need, for instance, linked lists, expandable arrays, hash tables, etc. yourself).

The **CleverSIDC** must implement the following methods:

- **SetSIDCThreshold (Size)**: where $100 \leq \text{Size} \leq \sim 500{,}000$ is an integer number that defines the size of the list. This size is very important as it will determine what data types or data structures will be used (i.e. a Tree, Hash Table, AVL tree, binary tree, sequence, etc.);
- **generate()**: randomly generates new non-existing key of 8 digits;
- **allKeys(CleverSIDC)**: return all keys in CleverSIDC as a **sorted sequence;**
- **add(CleverSIDC,key,value[2])**: add an entry for the given key and value;
- **remove(CleverSIDC,key)**: remove the entry for the given key;
- **getValues(CleverSIDC,key)**: return the values of the given key;
- **nextKey(CleverSIDC,key)**: return the key for the successor of key;
- **prevKey(CleverSIDC,key)**: return the key for the predecessor of key;
- **rangeKey(key1, key2)**: returns the number of keys that are within the specified range of the two keys *key1* and *key2*.

[1] The lower the memory and runtime requirements of the ADT and its operations, the better will be your marks.

[2] Value here could be any info of the student. You can use a single string composed of Family Name, First Name, and DOB.

1. Write the pseudo code for at least 4 of the above methods.
2. Write the java code that implements all the above methods.
3. Discuss how both the *time* and *space* complexity change for each of the above methods depending on the underlying structure of your CleverSIDC (i.e. whether it is an array, linked list, etc.)?

You have to submit the following deliverables:

a) A detailed report about your design decisions and specification of your CleverSIDC ADT including a rationale and comments about assumptions and semantics.

b) Well-formatted and documented Java source code and the corresponding class files with the implemented algorithms.

c) Demonstrate the functionality of your CleverSIDC by documenting at least 5 different, but representative, data sets. These examples should demonstrate all cases of your CleverSIDC ADT functionality (e.**g., all operations of your ADT for different sizes).** You have to additionally test your implementation with benchmark files that are posted along with the assignment.

Submitting Assignment 3

You need to submit Part I and Part II separately.

Part I: Part I must be submitted individually (No groups are allowed) under the submission folder: Assignment 3 – Part I.

Part II:

- **For Part II, a group of 2 (maximum) is allowed. No additional marks are given for working alone.**
- **If working alone, you need to zip (see below) and submit your zipped file under the submission folder: Assignment 3 - Part II.**
- **If working in a group, only one submission is to be made by either of the two members (do not submit twice). You need to zip (see below) and submit your zipped file, under the submission folder: Assignment 3 - Part II.**

Submission format: All assignment-related submissions must be adequately archived in a ZIP file using your ID(s) and last name(s) as file name. The submission itself must also contain your name(s) and student ID(s). Use your "official" name only - no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. If working in a group, the file name must include both IDs and last names.

IMPORTANT: For Part II of the assignment, a demo for about 5 to 10 minutes will take place with the marker. You (or **both** members, if groups are permitted) **must** attend the demo and be able to explain their program to the marker. Different marks may be assigned to teammates based on this demo. The schedule of the demos will be determined and announced by the markers, and students must reserve a time slot for the demo (only one time-slot per group). **Now, please read very carefully:**

- **If you fail to demo, a zero mark is assigned regardless of your submission.**
- **If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a penalty of 50% will be applied.**
- **Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.**