Computer Science CSC263H
St. George Campus

February 11, 2016
University of Toronto

Homework Assignment #3
Due: February 25, 2016, by 5:30 pm

- You must submit your assignment as a PDF file of a typed (**not** handwritten) document through the MarkUs system by logging in with your CDF account at `markus.cdf.toronto.edu/csc263-2016-01`. To work with a partner, you and your partner must form a group on MarkUs.
- The PDF file that you submit must be clearly legible. To this end, we encourage you to learn and use the LaTex typesetting system, which is designed to produce high-quality documents that contain mathematical notation. You can use other typesetting systems if you prefer, but handwritten documents are not accepted.
- By virtue of submitting this assignment you (and your partner, if you have one) acknowledge that you are aware of the policy on homework collaboration for this course.[a]
- For any question, you may use data structures and algorithms previously described in class, or in prerequisites of this course, without describing them. You may also use any result that we covered in class, or is in the assigned sections of the official course textbook, by referring to it.
- Unless we explicitly state otherwise, you should justify your answers. Your paper will be marked based on the correctness and completeness of your answers, and the clarity, precision, and conciseness of your presentation.

[a] "In each homework assignment you may collaborate with at most one other student who is currently taking one of the sections of CSC263H taught this term. If you collaborate with another student on an assignment, you and your partner must submit only one copy of your solution, with both of your names. The solution will be graded in the usual way and both partners will receive the same mark. **Collaboration involving more than two students is not allowed. For help with your homework you may consult only the course instructors, teaching assistants, your homework partner (if you have one), your textbook and your class notes. You may not consult any other source.**"

Question 1. (22 marks) Consider an abstract data type that consists of a *set* S of integers on which the following operations can be performed:

Add(i) Adds the integer i to S. If this integer already is in S, then S does not change

Sum(t) Returns the sum of all elements of S that are less than or equal to the integer t. If all the elements of S are greater than t, then return 0.

Describe how to implement this abstract data type using an augmented AVL tree. Each operation should run in $O(\log n)$ worst-case time, where $n = |S|$. Since this implementation is based on a data structure and algorithms described in class and in the AVL handout, you should focus on describing the extensions and modifications needed here.

a. Give a precise and full description of your data structure. In particular, specify what data is associated with each node, specify what the key is at each node, and specify what the auxiliary information is at each node. In particular, what is (are) the augmented field(s), and what identity should this (these) fields satisfy. Illustrate this data structure by giving an example of it (with a small set of your own choice).

b. Describe the algorithm that implements each one of the two operations above, and explain why each one takes $O(\log n)$ time in the worst-case. *Your description and explanation should be in clear and concise English.* For the operations SUM, you should also give the algorithm's high-level pseudocode.

Question 2. (18 marks) Describe an algorithm for the following problem. The input to the algorithm is two unsorted sequences $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_n$ of n distinct positive integers each, and a positive integer z. The algorithm should determine whether or not there are i and j, $1 \leq i, j \leq n$, such that $z = x_i + y_j$. The ***expected*** time complexity of your algorithm should be $O(n)$, under some assumptions that we discussed in class.

HINT: What data structures or algorithms that we learned in class involve reasoning about probability?

Note: Do ***not*** assume that the numbers in the array are small, or they have a small range, e.g., they could be arbitrary integers. So a solution based on a direct access table or on linear time sorting is ***not*** acceptable.

a. Describe your algorithm clearly and concisely in English, and then give the pseudo-code.

b. Explain why the ***expected*** running time of your algorithm is $O(n)$. Explicitly state every assumption that you need for your complexity analysis.

c. What is the ***worst-case*** running time of your algorithm? Use the Θ notation and justify your answer.

[***The questions below will not be corrected/graded. They are given here as interesting problems that use material that you learned in class.***]

Question 3. (0 marks)

A *Scheduler* $\mathcal{S}$ consists of a set of *threads*; each thread is a tuple $t = (id, status)$ where id is a distinct positive integer and $status \in \{A, R, S\}$; intuitively, A means active, R means ready to be scheduled, and S means stalled. The operations that Scheduler $\mathcal{S}$ supports are:

NewThread(t): Given a thread $t = (id, A)$ where id is larger than that of any thread currently in $\mathcal{S}$, add thread t to $\mathcal{S}$.

Find(i): If $\mathcal{S}$ has a thread $t = (i, -)$ then return t, else return -1.

Completed(i): If $\mathcal{S}$ has a thread $t = (i, -)$ then remove t from $\mathcal{S}$, else return -1.

ChangeStatus($i, stat$): If $\mathcal{S}$ has a thread $t = (i, -)$ then set the *status* of t to *stat*, else return -1.

ScheduleNext: Find the thread t with smallest id among all the threads whose *status* is R in $\mathcal{S}$, set the *status* of t to A and return t; if $\mathcal{S}$ does not have a thread whose *status* is R then return -1.

In this question, you must describe how to implement the Scheduler $\mathcal{S}$ described above using an *augmented AVL tree* such that each operation takes $O(\log n)$ time in the worst-case, where n is the number of threads in $\mathcal{S}$. Since this implementation is based on a data structure and algorithms described in class and in the AVL handout, you should focus on describing the extensions and modifications needed here.

a. Give a precise and full description of your data structure. In particular, specify what data is associated with each node, specify what the key is at each node, and specify what the auxiliary information is at each node. Illustrate this data structure by giving an example of it (with a small set of threads of your own choice).

b. Describe the algorithm that implements each one of the five operations above, and explain why each one takes $O(\log n)$ time in the worst-case. *Your description and explanation should be in clear and concise English.* For the operations ChangeStatus($i, stat$) and ScheduleNext, you should also give the algorithm's high-level pseudocode.

Question 4. (0 marks) You are given a list L of n, not necessarily distinct, integers. Your task is to devise an algorithm that outputs a list of the distinct integers in L, in order of non-increasing frequency (the frequency of an element in L is the number of times it occurs in L). For example, if L is $2, 5, 4, 4, 2, 3, 4, 3$, then a suitable output is $4, 3, 2, 5$; the only other suitable output for this list is $4, 2, 3, 5$. In this example, L contains only small integers, but this is not always the case. You cannot make any assumption on the integers in L (e.g., their maximum size, or distribution). In particular, some integers of L can very large, and so it is not feasible to use tables of size proportional to these integers.

a. Give a Hash Table-based algorithm for this problem whose *expected* time complexity is $O(n)$, under some Hash Table assumptions that you must clearly state. Explain why your algorithm is correct and achieves this time complexity under these assumptions.
What is the *worst-case* time complexity of your algorithm? Use the Θ notation and justify your answer.

Hint: As part of your algorithm, find a way to sort n numbers in the range 0 to n in $O(n)$ time in the worst-case.

b. Give an algorithm for this problem whose *worst-case* time complexity is $O(n \log n)$. Describe why your algorithm in clear and concise English, and explain why it achieves this time complexity.