Cover page for CSC263 Homework #3

(fill and attach this page to your homework)

Submitted by:

(1) Family Name: Kozakov	(2) Family Name: ______
Given Name: Michael	Given Name: ______
Student Number: 999301018	Student Number: ______

Graded Homework should be returned in Tutorial: _______(tutorial room number)

For help with your homework you may consult only the instructor, the teaching assistants, your homework partner (if you have one), your textbook and your class notes. **You may not consult any other source**.

By virtue of submitting this homework I/we acknowledge that I am/we are aware of the policy on homework collaboration for this course.

Computer Science CSC263H | February 28, 2013
St. George Campus | University of Toronto

Homework Assignment #3

Due: March 14, 2013, by 5:30 pm
(in the drop box for this course in Bahen 2220)

1. *On the cover page of your assignment, in addition to your name(s), you must write the location of the tutorial where you want the graded homework to be returned.*
2. *For any question, you may use data structures from class without describing how the operations on these data structures are implemented. You may also use any result (e.g., a worst-case time complexity bound) that we covered in class, or is in the course textbook, by referring to it.*
3. *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.*

Question 1. (26 marks) Consider the disjoint-set forest implementation of Union/Find. In class, we considered a heuristic, called *weighted-union*, that is used to reduce the height of the trees that are formed during the execution of sequences of Unions and Finds. In this question, we will explore another heuristic, called *union-by-rank* that can be used instead of *weighted-union* to achieve the same goal.

With the *union-by-rank* heuristic, the root t of each tree T (formed during the execution of a sequence of Unions and Finds) has a value, denoted $rank(t)$, that is an upper bound on the height of T. The ranks of the (roots of the) trees are used to determine how to do the Union of any two trees as follows:

Initially, every tree T consists of a single node t of height 0, and so $rank(t) = 0$. Now suppose we want to Union two trees T and T' with roots t and t', respectively:

If $rank(t) > rank(t')$, then the root t of T becomes the parent of the root t' of T'.

If $rank(t') > rank(t)$, then the root t' of T' becomes the parent of the root t of T.

If $rank(t) = rank(t')$, then the root t' of T' becomes the parent of the root t of T, *and* $rank(t')$ *is incremented by one.*

This heuristic is also explained in pages 569 to 570 of our textbook CLRS, except that in CLRS's explanation *every* node has a rank. In reality, to do the Unions we only need to maintain the rank of (the root of) each tree.

In the following, we start from n distinct elements (each in a singleton set), and we consider an execution of an arbitrary sequence σ of $n-1$ Unions and $m \geq n$ Finds, using *union-by-rank* but *without* path compression.

a. (8 marks) Prove that for any tree Q formed during the execution of σ, $height(Q) = rank(q)$ (where q is the root of Q),

b. (8 marks) Prove that for any tree Q formed during the execution of σ, $rank(q) \leq \lfloor log_2 n \rfloor$.

c. (10 marks) Prove that the worst-case time complexity to execute σ (over all possible sequences σ of $n-1$ Unions and $m \geq n$ Finds) is $\Theta(m \log_2 n)$.

Question 2. (20 marks)

Consider the Randomized Quicksort (RQS) sorting algorithm that we described in class. In the following:

- Your answers should be ***exact numbers***; do ***not*** use asymptotic notation.
- Explain your answers.

a. (5 marks) What is the ***worst-case*** total number of comparisons (between the elements of the sequence) when we execute RQS on a sequence with 7 distinct integers?

b. (5 marks) What is the ***best-case*** total number of comparisons (between the elements of the sequence) when we execute RQS on a sequence with 7 distinct integers?

c. (4 marks) We execute RQS on the sequence $A = 2, 8, 11, 3, 12, 7, 10, 4, 15$. What is the probability that 4 and 11 are compared to each other?

d. Let E_n be the ***expected*** total number of comparisons (between the elements of the sequence) when we execute RQS on a sequence with n distinct integers.

1. (4 marks) What is E_4?
2. (2 marks) What is the difference between E_5 and E_4, i.e., what is $E_5 - E_4$?

   Hint: there is no need to compute each of E_4 and E_5 separately.

Question 3. (26 marks) In this question you will explore different algorithms to perform the *join* of two relations, a very important operation in relational databases. We consider a simplified setting where we wish to compute the join of relation R with attributes A and C, and relation S with attributes B and C. Notice that the two relations have a common attribute, C. Each relation is a set of pairs. For each pair (a, c) of R, a is a value of attribute A and c is a value of attribute C. Similarly, for each pair (b, c) of S, b is a value of attribute B and c is a value of attribute C. It may be useful to think of R as a two-dimensional array, with two columns, the first corresponding to A and the other corresponding to C: each row of the two-dimensional array corresponds to a pair (a, c) of R; so the number of rows of the array is equal to the number of pairs in R. Similar comments apply to S.

The *join* of R and S, denoted $R \bowtie S$, is a relation with attributes A, B and C (i.e., the union of the attributes of R and S). It consists of the set of all triples (a, b, c) such that (a, c) is a pair in R and (b, c) is a pair in S. Notice that the two pairs that are being joined to produce the triple (a, b, c) of $R \bowtie S$ must have *the same value* in the common attribute C.

Another way of thinking about $R \bowtie S$ is as follows: We consider every possible combination of a pair (a, c) from R and a pair (b, c') from S. If $c = c'$ — i.e., if the two pairs have the same value in the common attribute — then this combination contributes the triple (a, b, c) to the join. If $c \neq c'$ then this combination contributes nothing to the join.

For example, suppose that R is the relation *Teaches* with attributes *ProfName* and *CourseId*; and S is the relation *Takes* with attributes *StudName* and *CourseId*. A pair (p, c) is in *Teaches* if and only if professor p teaches course c. A pair (s, c) is in *Takes* if and only if student s takes course c. The join *Teaches* $\bowtie$ *Takes* is the relation with attributes *ProfName*, *StudName*, *CourseId*; it contains a triple (p, s, c) if and only if professor p teaches student s in course c.

The obvious way to compute $R \bowtie S$ is to consider each combination of pairs from R and S and create a triple from each such combination where the two pairs have the same value in the common attribute. Obviously this algorithm takes time $\Theta(mn)$, where m is the number of pairs in R and n is the number of pairs in S.

In a way, this is the best we can hope for: If all pairs in R and all pairs in S have the same value in the common attribute C, then there will be mn triples in $R \bowtie S$, and we can't create a set of mn triples faster than in $\Omega(mn)$ time! In practice, however, it is very unlikely that every pair in R joins with every pair in S; in most cases, the size of $R \bowtie S$ is much smaller than mn. For instance, in our example of the *Teaches* and *Takes* relations, if these happened to describe, respectively, the courses taught by U of T faculty and the courses taken by U of T students in a particular semester, the first relation would contain about 4,000 pairs (approximately 2,000 professors each teaching 2 courses) and the second relation would contain about 250,000 pairs (approximately 50,000 students, each taking 5 courses). In this case the product mn is approximately one billion, while the actual size of *Teaches* $\bowtie$ *Takes* will be approximately 250,000 — more than three orders of magnitude smaller! (There could be more records in *Teaches* $\bowtie$ *Takes* than in *Takes*, in case a course is co-taught by several professors, but such instances are rare, so typically the size of *Teaches* $\bowtie$ *Takes* will be only a little bit larger than that of *Takes*.)

In view of this example, we would like to have an algorithm that computes $R \bowtie S$ in time $\Theta(k + f(m, n))$, where k is the actual size of $R \bowtie S$ and $f(m, n)$ is as small a function as possible.

In the following two questions you are asked to devise and describe such algorithms. You should describe each algorithm in *clear and precise* English, and *illustrate the data structure(s)* that you use with simple example(s); no pseudocode is necessary (but you can use pseudo-code as an additional explanation). To the extent your algorithm makes use of algorithms or data structures discussed in this course or in its prerequisites, you can simply state what algorithm(s) and data structure(s) you use without describing them in detail.

In the following, assume that each of R and S is given as a two dimensional array, where each column corresponds to an attribute and each row corresponds to a pair of the relation. You can **not** make any assumption on the size or distribution of values contained in R or S (i.e., the values associated with attributes A, B, or C). In particular, some of these values may be very large (e.g., they may be 128-bit integers), and so it is **not** feasible to use arrays of size proportional to these values.

Let k be the size of the join $R \bowtie S$, and N be the size of the larger of the two given relations R and S, i.e., $N = max(n, m)$.

a. (10 marks) Describe an algorithm that computes $R \bowtie S$ in *worst-case* time $\Theta(k + N \log N)$. Your should describe your algorithm as we explained above (*start with a few English sentences explaining the high-level idea of your algorithm*).

Explain why the worst-case time complexity of your algorithm is $\Theta(k + N \log N)$.

b. (16 marks) Describe an algorithm that computes $R \bowtie S$ in *expected* time $\Theta(k + N)$ *under some reasonable assumptions that you should state.* Your should describe your algorithm as we explained above (*start with a few English sentences explaining the high-level idea of your algorithm*).

Explain why the expected running time of your algorithm is $\Theta(k + N)$.

What is the *worst-case* time complexity of this algorithm? Justify your answer.

Hint: Note that R or S may have many pairs with the *same* value c in attribute C. You should think about this case carefully when you formulate your assumptions, and when you analyse the time complexity of your algorithm.