Assignment 4

COMPSCI 2CO3: Data Structures and Algorithms–Fall 2025

Deadline: October 24, 2025

Department of Computing and Software
McMaster University

Please read the *Course Outline* for the general policies related to assignments.

Plagiarism is a *serious academic offense* and will be handled accordingly.
All suspicions will be reported to the *Office of Academic Integrity*
(in accordance with the Academic Integrity Policy).

This assignment is an *individual* assignment: do not submit work of others. All parts of your submission *must* be your own work and be based on your own ideas and conclusions. Only *discuss or share* any parts of your submissions with your TA or instructor. You are *responsible for protecting* your work: you are strongly advised to password-protect and lock your electronic devices (e.g., laptop) and to not share your logins with partners or friends!

If you *submit* work, then you are certifying that you are aware of the *Plagiarism and Academic Dishonesty* policy of this course outlined in this section, that you are aware of the Academic Integrity Policy, and that you have completed the submitted work entirely yourself. Furthermore, by submitting work, you agree to automated and manual plagiarism checking of all submitted work.

Late submission policy. Late submissions will receive a late penalty of 20% on the score per day late (with a five hour grace period on the first day, e.g., to deal with technical issues) and submissions five days (or more) past the due date are not accepted. In case of technical issues while submitting, contact the instructor *before* the deadline.

Problem 1. Consider two sets-of-pairs F and S. For example, set F can hold student information (pairs *student identifier* and *name*) and list S can hold course enrollment information (pairs *student identifier* and *course identifier*). When processing data, a typical operation on F and S is computing the *join*:

$$\text{Join}(F, S) = \{(a, b, d) \mid (a, b) \in F \wedge (c, d) \in S \wedge a = c\},$$

e.g., if F holds student information and S holds course enrollment information, this join will combine student information of students s (from F) with course identifiers of the courses taken by student s (from S). Note that $\text{Join}(F, S)$ defines a *set*. Hence, there are no duplicate values.

P1.1. Consider the following algorithm:

Algorithm ComputeJoin(F, S)**:**

```
1: result := an empty list of triples.
2: for (a, b) ∈ F do
3:    for (c, d) ∈ S do
4:       if a = c then
5:          Add (a, b, d) to result.
6:       end if
7:    end for
8: end for
9: return result.
```

Argue whether this algorithm correctly computes Join(F, S) and argue what the complexity of this algorithm is.

P1.2. Write an algorithm that, given sets-of-pairs F and S, *correctly computes* the above join Join(F, S). The complexity of your algorithm must be at-most $O(|F| + |S| + |result|)$ in which $|result|$ is the size of the result (the number of triples $(a, b, c) \in \text{Join}(F, S)$).

You may assume that F and S are represented by lists-of-pairs that are sorted on lexicographical order. We say that a list-of-pairs L is sorted on lexicographical order if pairs $(fst, snd) \in L$ are ordered on their first component *fst* and, when pairs have identical first components, on their second component *snd*. Hence, L is sorted on lexicographically order if, for all $0 \leq i < j < |L|$ with $L[i] = (fst_i, snd_i)$ and $L[j] = (fst_j, snd_j)$, we either have $fst_i < fst_j$ or we have $(fst_i = fst_j \wedge snd_i \leq snd_j)$.

P1.3. Argue why your solution to Problem P1.2 correctly computes Join(F, S) and argue why your algorithm has a worst-case complexity of $O(|F| + |S| + |result|)$.

Problem 2. Randomized QuickSort is a *fast* sorting algorithm in practice, even if we cannot guarantee a running time of worst-case $O(|L| \log_2(|L|))$ to sort lists L.

P2.1. Assume that for every list L, we can magically find a *perfect pivot value* $v \in L$ such that exactly $\lfloor |L|/2 \rfloor$ values in L are smaller than v. This magic method has a worst-case runtime of $O(|L|)$. Provide a recurrence $T(n)$ for the worst-case runtime complexity of QuickSort using this magic method and prove that $T(n) = O(|L| \log_2(|L|))$.

P2.2. Note that the Select algorithm from the slides can be used to find a *perfect pivot value*. What is the worst-case runtime complexity of the Select algorithm from the slides to find a perfect pivot value? Provide a recurrence $T(n)$ for the worst-case runtime complexity of QuickSort using the Select algorithm from the slides to always use a perfect pivot value and provide the complexity of this QuickSort variant (by proving that, in the worst case, $T(n) = O(f(n))$ for some function $f(n)$).

P2.3. After hard working, we could not easily find a method that can find a *perfect pivot value*. We did find a method ThreeTen to find a not-too-bad pivot value $v \in L$ such that on large-enough lists at-least 30% of the values in L are smaller than v and 30% of the values in L are larger than v.

Provide a recurrence $T(n)$ for the complexity of Select using the method ThreeTen to pick the pivot and provide the complexity of this Select variant (by proving that, in the worst case, $T(n) = O(f(n))$ for some function $f(n)$).

P2.4. What does the existence of method ThreeTen imply with respect to the complexity of QuickSort?

Assignment Details

Write a report in which you solve each of the above problems. Your submission:

1. must start with your name, student number, and MacID;
2. must be a PDF file;
3. must have clearly labeled solutions to each of the stated problems;
4. must be clearly presented;
5. must *not* be hand-written: prepare your report in LaTeX or in a word processor such as Microsoft Word (that can print or export to PDF).

Submissions that do not follow the above requirements will get a grade of zero.

Grading

Each problem counts equally toward the final grade of this assignment.