Remedial Exam

The Art of Computer Programming

30 March 2026

You have 3 hours to answer this remedial exam. You are allowed 10 A4 sheets (i.e., 20 pages) with the content of your choice. No electronic devices or any other material is allowed. Do not forget to add your name on all your answer sheets and to number them.

The exam consists of a single problem divided into several parts, which are largely independent; it is graded out of 20 points.

A *word counter* stores words (character strings) together with the number of times they have been inserted. It supports the following operations:

constructor() Create an empty word counter.

add(word) Insert one occurrence of `word`. If the word is already present, its counter is increased by 1.

count(word) Return the number of occurrences of `word` (or 0 if `word` is not present).

remove_one(word) Remove one occurrence of `word`. If its counter becomes 0, the word is removed completely.

Part A: Warm-up (1 point)

Q1. (1 point) Consider the following sequence of operations:

constructor(), add("cat"), add("dog"), add("cat"), add("bird"), remove_one("dog").

After this sequence of operations, what are the values returned by:

count("cat"), count("dog"), count("bird"), count("fish")?

Part B: Implementation with an Unsorted Python List (4 points)

We implement a word counter in Python using a list of pairs `(word, count)` in arbitrary order, with the constraint that each word is stored at most once.

Q2. (1.5 points) Complete the following Python class by writing the content of the constructor and of the methods `add` and `count`.

```
class WordCounter:
    def __init__(self):
        pass

    def add(self, word):
        pass

    def count(self, word):
        pass
```

Q3. (1 point) Write a Python method `remove_one(self, word)` for this class.

Q4. (1.5 points) Give the worst-case time complexity of `add`, `count`, and `remove_one`, in terms of the number n of distinct words stored in the data structure. Briefly justify each answer.

Part C: Algorithm Design on Sequences (4 points)

Suppose now that we are given, as input, a finite sequence of words $(w_0, w_1, \ldots, w_{m-1})$, and we want to compute the number of *distinct* words occurring in it.

For example:

- on input (`"a"`, `"b"`, `"a"`, `"c"`), the answer is 3;
- on input (), the answer is 0;
- on input (`"x"`, `"x"`, `"x"`), the answer is 1.

Q5. (2 points) Write an algorithm, in pseudocode or in Python, C, or C++, that computes this number of distinct words using the word counter data structure. Your algorithm may maintain an additional integer variable.

Q6. (1 point) Argue briefly why your algorithm is correct.

Q7. (1 point) Assume that the operations of the word counter have the same complexity as in Part B. Give the worst-case time complexity of your algorithm in terms of m, the length of the input sequence, and justify it.

Part D: Sorted Array in C++ (4 points)

We now implement the same word counter in C++ using a `std::vector` storing pairs of the form `(std::string, unsigned)`, but this time kept *sorted by word* (using lexicographic order on strings). All operations need to keep the vector sorted by word.

Q8. (1 point) Explain how binary search can be used to implement count(word) in this representation.

Q9. (2 points) Write C++ code, for a method `unsigned count(const std::string &word)` using binary search.

Q10. (1 point) Give the worst-case time complexity of count(word) and add(word) in this representation, in terms of the number of distinct words n. Justify briefly.

Part E: Hash-table implementation (3 points)

We now want to move to an implementation of a word counter based on a data structure that uses a hash table.

Q11. (1 point) Which data structure of Python can be used for that purpose? Which data structure of the C++ standard template library can be used for that purpose?

Q12. (1 point) Write the implementation of add(word) and count(word) in either Python or C++ for such a data structure.

Q13. (1 point) What is the expected time complexity of these two operations (in terms of the number of distinct words)?

Part F: Correctness and Software Engineering (4 points)

Q14. (1 point) Give a reasonable precondition and postcondition for the operation remove_one(word).

Q15. (1 point) Consider the binary-search-based implementation of `count(word)` from Part D. State a suitable loop invariant for the search.

Q16. (2 points) Write two unit tests in **pytest style** for the Python implementation of Part B:

- one test checking that inserting the same word twice makes its count equal to 2;
- one test checking that removing one occurrence of a word with count 1 makes its count equal to 0.

You may assume that the class is named `WordCounter` as in Part B.