University of Waterloo
Midterm Examination
Winter 2019

Student Name:	______________________
Student ID Number:	______________________
Course Section:	**BME 122**

Instructors:	Dr. Igor Ivkovic
Date of Exam	Winter 2019
Time Period	2:00pm-4:00pm
Duration of Exam	120 minutes
Pages (including cover)	12 pages (two scrap pages)
Exam Type	Closed Book

NOTE: No calculators or other electronic devices are allowed. Do not leave the exam room during the first 30 minutes and the last 15 minutes of the exam. Plan your time wisely to finish the exam on time.

Question 1: (10 marks)	Question 2: (15 marks)	Question 3: (10 marks)	Question 4: (15 marks)
Total: (50 marks)			

Useful Formulas

You may remove this page from the exam. If you do remove it, write your Student Number on it, and hand it in with your exam.

- For $S = a_1 + (a_1 + d) + ... + (a_n - d) + a_n$
  (S is a series that goes from a_1 to a_n in d-size increments),
  $$S = n\left(\frac{a_1 + a_n}{2}\right) \tag{1}$$
- $$\sum_{i=k}^{n} 1 = (n - k + 1) \tag{2}$$
- $$\sum_{i=1}^{n} i = \left(\frac{n(n+1)}{2}\right) \tag{3}$$
- $$\sum_{i=1}^{n} i^2 = \left(\frac{n(n+1)(2n+1)}{6}\right) \tag{4}$$
- $$\sum_{i=0}^{n} r^i = \left(\frac{r^{n+1} - 1}{r - 1}\right) \tag{5}$$
- $$\log_b x = \frac{\log_c x}{\log_c b} \tag{6}$$
- $$x^{\log_b y} = y^{\log_b x} \tag{7}$$
- $$\log_b xy = \log_b x + \log_b y \tag{8}$$
- $$\log_b \frac{x}{y} = \log_b x - \log_b y \tag{9}$$

1 Question 1. Algorithm Complexity (10 marks)

(approx. 20 minutes)

a. (4 marks)

Provide a code fragment in pseudocode that would exactly exhibit the runtime of: $O(log(n^2) + n\ log(n))$ with no expression simplification.

For instance, for O(n) runtime, you could write the following:

```
loop from 1 to n {
    call swap(a, b) that performs constant number of steps
}
```

b. (6 marks)

Let the runtime efficiency of an algorithm be defined as:
$T(n) = 21log_4(343) + 42log_4(343) + 84log_4(343) + ... + 21log_4(343)2^{n-1}$.

Derive and express using Big-O notation a tight upperbound for $T(n)$ using limits as $n \to \infty$ or using the Big-O formal definition. Show all steps in deriving your solution.

2 Question 2. Algorithm Design (15 marks)

(approx. 40 minutes)

a. (10 marks)

A friend sent you a message composed of letters b, m, and e, where each letter appears at least once and in that order. For example, valid strings would be *bmee*, *bbmmee*, and *bmeeee*.

Unfortunately, this text message got distorted in transmission, and some letters may have been deleted and additional letters may have been inserted.

Write the function

```
bool has_original_message(string message, string garbled_text)
```

that will check if the original message, which you have later received through other means, is contained in the garbled text. If the message is contained in the text, then your function will return $true$. Otherwise, your function will return $false$.

For example, if the original message equals <u>*bmee*</u> and the text equals <u>*bmttteee*</u> then your function will return $true$. Similarly, if the original message equals <u>*bbmmee*</u> and the text equals <u>*bcbbccmmdmeee*</u> then your function will return $true$. However, if the original message equals <u>*bme*</u> and the text equals <u>*bttt*</u> then your function will return $false$.

You may assume that both strings will contain only lower-case letters; that the order of b, m, and e appearances will not change (e.g., no m's will appear before bs); and that the garbled text will always be larger in size than the original message.

Hint: Use a temporary array of integers to record number of occurrences of b, m, and e, respectively. When going the through the message string, increment the corresponding position by one on each letter. When going through the garbled text, decrement the corresponding position by one when encountering b, m, and e, respectively. Use the temporary array at the end to check if any of its elements are still greater than zero.

You may utilize operations of any of the abstract data types (ADTs) discussed in class so far, such as List ADT, without implementing the details. You may also write your function in pseudocode without focusing on syntactic details (e.g., you may write "$for\ i\ =\ 0\ to\ (n-1)$"), but you need to specify each step and each function call unambiguously. To access string elements, you may use the $str[index]$ operator. To access string size, you may use $str.size()$.

For full marks, the run-time performance of your algorithm needs to be $O(n + m)$ where n is the size of the original message and m is the size of the garbled text. That is, you may iterate through each string in its own loop but those loops cannot be nested. For partial marks, the run-time performance of your algorithm can be worse than linear (e.g., $O(n^2)$). [The amount of partial marks awarded will be decided when grading.]

b. (5 marks)

Once you have designed the algorithm, explain in natural language — not code — how would you comprehensively test the algorithm. That is, list ten or more test cases that you would use to ensure correctness of your algorithm and enhance its robustness. Group the test cases into categories and provide short description for each category.

For example, one of the test case given above could be explained as "test a scenario where the given message is not contained in the garbled text." You do not have to modify the code to pass the test cases.

3 Question 3. Divide and Conquer Algorithms (10 marks)

(approx. 20 minutes)
Consider the following function:

```
void woof_star(vector<int>& data, int first, int last) {
    cout << "woof_star called with [" << first << ", " << last << "]" << endl;

    int partition_size = (last - first) / 5;

    if (last <= first + 1) {
        ++data[last];
        cout << "YAS!" << endl;
        return;
    }

    for (int index = (partition_size + 1); index <= (partition_size * 2); ++index) {
        cout << "WOOF ";
    }
    cout << endl;

    woof_star(data, first + 1, first + partition_size);
    woof_star(data, first + partition_size * 4 + 1, last);
}
```

a. (3 marks)
Draw the call tree for this function when $woof\_star(data, 18, 32)$ is called.

b. (7 marks)

Let $n = last - first$ for $n \geq 0$ and let the recurrence relationship for this function be defined as: $T(1) = a, T(n) = 2T(n/5) + n/5 + b \ for \ n > 1$.

Solve the recurrence relationship by unrolling the recurrence (use back substitution), and give the $T(n)$'s order of growth in terms of the Big-O notation as a function of n. You may apply the heuristics once you have fully simplified the expression. Show all steps in deriving your solution.

4 Question 4. Data Structure Design (15 marks)

(approx. 40 minutes)

We are designing a mobile app for a service similar to Netflix that will offer a subscription-based access to training content authored by our company. The app will allow users to create a list of their favourite viewing items that are offered by the service. For each favourite item, the app will store the unique item ID as an unsigned integer, item name as a string, item rating as an integer that represents the number of stars (from one star to five stars in one-star increments), item category as a string (e.g., data analytics, healthcare), and user-defined notes as a string.

a. (4 marks) As the reference model for our design, we will make use of the **doubly linked list from Lab Assignment #2** where $FavItem$ objects will be managed by the $FavsManager$. $FavsManager$ needs to include **pointers to first and last elements**. To that end, complete the C++ class declaration provided below to make $FavItem$ and $FavsManager$ function. For $FavItem$, include the empty constructor, parametric constructor, and *operator* ==. For $FavsManager$, include the empty constructor, *void push_back*($FavItem$& *new_item*), and *void pop_nth_from_end*(*unsigned int index*). Provide only method signatures at this point and no method implementation. Do not forget to include the *size* variable.

```
class FavItem {
    unsigned int itemID;

    friend class FavsManager;

    // fill in other required member attributes and methods below
    // members should be declared as public or private as appropriate









};

class FavsManager {
    // fill in other required member attributes and methods below
    // members should be declared as public or private as appropriate









};
```

b. (5 marks) Implement the method *void push_back(FavItem& new_item)* that will insert the referenced *FavItem* element at the end of the list using dynamic memory, and update the *size* variable. You must use exact C++ syntax for this function.

```
void FavsManager::push_back(FavItem& new_item) {
```

c. (6 marks) Implement the method *void pop_nth_from_end(unsigned int index)*. The method will take as input an *index* value, remove the index-th element from the end of the list, free up its memory, and update the *size* variable.

For example, if the list stores the following item IDs: $141 \rightarrow 123 \rightarrow 15$ and if $index = 2$, then the function will remove the second item from the end and produce the following list: $141 \rightarrow 15$. If the $index = 0$ or if *index* exceeds list size, then no change will be made.

You must use exact C++ syntax for this function. You may only use *iostream* and *string* libraries in your implementation and no other external libraries. However, you may write helper functions of your own. You also need to document your code and explain the rationale for key steps.

```
void FavsManager::pop_nth_from_end(unsigned int index) {
```

Scrap Paper

You may remove this page from the exam. If you do remove it, write your Student Number on it, and hand it in with your exam.