University of Waterloo
Midterm Examination
Winter 2019

Student Name:	_____
Student ID Number:	_____
Course Section:	**SYDE 223**

Instructors:	Dr. Igor Ivkovic
Date of Exam	Winter 2019
Time Period	11:30am-1:30pm
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: $O(log(log(n^2)) + n\ log(n))$ 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 an important text message. Unfortunately, this text message got distorted in transmission. What you have received instead is a string of random characters.

Write the function

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

that will use recursion to 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 without changing the order of the letters, then your function will return the number of extra characters that got inserted beyond the original message. If the message is not in the text, then your function will simply return -1.

For example, if the original message equals *run* and the text equals *erfuinnns* then your function needs to return 6, which is the number of characters that need to be removed from the text to derive the original message. Similarly, if the original message equals *bob* and the text equals *bcccddoffbbbb* then your function needs to return 10. However, if the original message equals *let* and the text equals *tellbbg* then your function needs to return -1. Furthermore, if the original message equals *clue* and the text equals *cluuu* then your function needs to return -1.

You may assume that both strings will contain only lower-case letters, and you may assume that the garbled text will always be larger in size than the original message.

Hint: Consider writing the overloaded recursive function

```
bool has_original_message(string message, string text, int size1, int size2,
int& distance)
```

that will be called from your non-recursive function. This function will return *true* if the message is contained within the garbled text, and will also count the difference in number of characters using the *distance* accumulator.

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 $O(n)$ where n is the size of the garbled text, and you must use recursion to solve this problem. For partial marks, the run-time performance of your algorithm can be worse than linear (e.g., $O(n^2)$), and you can use iteration to solve this problem. [The amount of partial marks awarded will be decided when grading.]

(extra space for completion of the Algorithm Design question)

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 inverted message is 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 $woof\_star(data, 17, 31)$ 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 → 123 → 15 and if *index* = 2, then the function will remove the second item from the end and produce the following list: 141 → 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.