University of Waterloo Midterm Examination Spring 2019

Student Name:	
Student ID Number:	
Course Section:	**MTE 140**

Instructors:	Dr. Igor Ivkovic
Date of Exam	Spring 2019
Time Period	12:30pm-2:20pm
Duration of Exam	110 minutes
Pages (including cover)	12 pages (one page of scrap paper)
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 10 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)

Order the following functions from smallest to largest based on their order of growth in terms of the Big-O notation:

1. $\log(\log(n^2))$
2. $1245^2 + 3547n^4 + 42234n^6 + 151515$
3. $3 + 9 + 27 + 81 + 243 + \cdots + 3^{n-1} + 3^n$
4. $n^2 \log_{1567}(2222^{4545n})$
5. $4 + 8 + 12 + 16 + 20 + \cdots + 4n$

b. (6 marks)

Let the runtime efficiency of an algorithm be defined as: $T(n) = \sum_{i=3}^{n} \sum_{j=1}^{n} \sum_{k=7}^{n} (i + 2j)$.

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)

You and your friend are playing a game in order to hone your mathematical skills.

They generate an unsorted list of *values* drawn from the range $[1, 1000]$ inclusive of 1 and 1000; the list may contain duplicate values. Your task is to read through the list, and extract the minimum and maximum value. After that, your friend would get to ask you for the I-th missing number from the range that starts with the minimum value and ends with the maximum value; if there is no such value, you would tell them -1. If you answer their question correctly three times, you win the game; if not, your friend wins. After that, you switch roles, and then you get to generate the list.

For example, your friend generates the following: $[14, 2, 3, 4, 6]$. The minimum value is 2, the maximum value is 14, and the missing values are $[5, 7, 8, 9, 10, 11, 12, 13]$. Your friend asks you for the $5th$ missing number to which you reply 10, $7th$ missing number to which you reply 12, and the $20th$ missing number to which you reply -1. Hence, you win the game.

Write the function

```
int find_ith_missing_value(vector<int> values, unsigned int missing_index)
```

that will help you play this game and that returns the missing I-th value. The *values* vector is the list that your friend generates while *missing_index* is the index of the missing value. If the index is zero or if it exceeds the range of the given values, return -1.

Hint: Use a temporary (buffer) array of integers to record number of occurrences of integers in the *values* list. Use the temporary array at the end to find the missing values. Record minimum and maximum indices as separate variables as you are iterating through the temporary array. If the *missing_index* is zero or if it exceeds the maximum index, return -1.

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.

For full marks, the run-time performance of your algorithm needs to be linear $O(n)$ where n is the size of the *values* list. That is, you may iterate through the list multiple times using loops but the loops cannot be nested. For partial marks, the run-time performance of your algorithm can be worse than linear (e.g., $O(n^2)$). Note that comparison-based sorting will require $O(n\ log(n))$ time. [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 five or more equivalence classes with ten or more test cases in total that you would use to ensure correctness of your algorithm and enhance its robustness. For each equivalence class, write a short description.

For example, one of the examples given above could be explained as "test a scenario where the values list contains no duplicates and the missing index exceeds the range." You do not have to modify the code to pass the test cases.

3 Question 3. Pointers and Dynamic Memory (10 marks)

(approx. 10 minutes)
Examine the following code fragment. For each line marked with the [*Question*∗] tag, answer the related question in the space below, and provide a short explanation. If a line represents a compilation error, state that instead, and briefly explain why the error occurs. Assume that no lines will be skipped in execution.

```
struct BasketballStar {
    string name; BasketballStar* next;
    BasketballStar(string new_name) : name(new_name), next(NULL) {}
};

BasketballStar** with_pointers_to_championships() {
    BasketballStar** roster = new BasketballStar*[15]();
    BasketballStar star1("Kyle"), star2("Danny"), star3; // [Question1. Will this compile?]

    roster[0] = &star1; // [Question2. Given that roster[0] is passed outside
                        // this function via the return statement,
                        // is this going to result in a dangling pointer?]

    roster[1] = new BasketballStar(star2); // [Question3. Is roster[1] pointing
                        // to heap memory or stack memory after this line?]

    cout << (*(roster + 1)).name  << endl; // [Question4. What is the output of this line?]
    cout << roster[2]->name << endl; // [Question5. What is the output of this line?]

    return roster;
}
```

[*Question*1]

[*Question*2]

[*Question*3]

[*Question*4]

[*Question*5]

4 Question 4. Data Structure Design (15 marks)

(approx. 40 minutes)
We are tasked with developing statistical software that stores information about basketball players and basketball teams currently playing in a basketball league (e.g., NBA, WNBA). For each player, the software will store their unique player ID as an unsigned integer, full name as a string, position played as a string, height as an integer value in centimeters, weight as an integer value in kilograms, and date of birth as a string. For each team, we will store the active roster and backup roster along with the size for each roster. Both rosters are not sorted.

a. (4 marks) As the reference model for our design, we will make use of the **sequential list implementation of List ADT from Practice Exercise #1 and Lab Assignment #1** where *BasketballPlayer* objects will be managed by the corresponding *BasketballTeam* instances. As internal storage inside *BasketballTeam*, you may use the $< vector >$ library or a dynamically-sized array. The default capacity for each roster is 15. Do not create a linked-list implementation.

To that end, complete the C++ class declaration provided below to make *BasketballPlayer* and *BasketballTeam* function. For *BasketballPlayer*, include the empty constructor, parametric constructor, and *operator* ==. For *BasketballTeam*, include the empty constructor, *void insert_new_player*(*BasketballPlayer new_player*), *void remove_player*(*unsigned int playerID*), *void activate_player*(*unsigned int playerID*) and *void deactivate_player*(*unsigned int playerID*). Provide only method signatures at this point and no method implementation. Include the *size* variables if applicable. You do not have to include getters and setters.

```
class BasketballPlayer {
    unsigned int playerID;
    friend class BasketballTeam;
    // fill in other required member attributes and methods below
    // members should be declared as public or private as appropriate









};

class BasketballTeam {
    // 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 insert_new_player(BasketballPlayer new_player) that will insert the *BasketballPlayer* element at the end of the backup roster list, and update the *size* variable if applicable. If the provided *playerID* is already in the list, then no change will be made.

If you are using the $< vector >$ library, you may use $push\_back(element)$ or $resize(value)$ functions as needed. You must use exact C++ syntax for this function.

```
void BasketballTeam::insert_new_player(BasketballPlayer new_player) {
```

c. (6 marks) Implement the method *void remove_player(unsigned int playerID)*. The method will take as input a *playerID* value, remove the corresponding player from the backup roster, free up its memory, move other players ahead in the roster by one so that the list remain contiguous, and update the *size* variable.

For example, if the backup roster stores the following player IDs: $[8, 25, 15, 13]$ and if *playerID* $= 25$, then the function will remove the second player from the list and produce the following: $[8, 15, 13]$. If the *playerID* is not in the list, then no change will be made.

You must use exact C++ syntax for this function. If you are using the $< vector >$ library, you may use *pop_back*() or *resize*(*value*) functions but not *erase*() or *swap*(). You may also use *iostream* and *string* libraries in your implementation but 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 BasketballTeam::remove_player(unsigned int playerID) {
```

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.