University of Waterloo
Midterm Examination
Winter 2016

Student Name: _____

Student ID Number: _____

Course Section (circle one): **BME 122 | MTE 140**

Instructors:	Alexander Wong and Igor Ivkovic
Date of Exam	February 26th 2016
Time Period	11:30am-1:20pm
Duration of Exam	110 minutes
Pages (including cover)	14 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: (11 marks)	Question 2: (8 marks)	Question 3: (8 marks)	Question 4: (11 marks)	Question 5: (12 marks)
Total: (50 marks)				

Useful Formulas

- 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),

$$S = n\left(\frac{a_1 + a_n}{2}\right) \tag{1}$$

- $$\sum_{i=k}^{n} 1 = (n - k + 1) \tag{2}$$

- $$\sum_{i=1}^{n} i = \left(\frac{n(n+1)}{2}\right) \tag{3}$$

- $$\sum_{i=1}^{n} i^2 = \left(\frac{n(n+1)(2n+1)}{6}\right) \tag{4}$$

- $$\sum_{i=0}^{n} r^i = \left(\frac{r^{n+1} - 1}{r - 1}\right) \tag{5}$$

- $$\log_b x = \frac{\log_c x}{\log_c b} \tag{6}$$

1 Question 1. Algorithm Complexity (11 marks)

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. $3 + 9 + 27 + 81 + 243 + \cdots + 3^{n-1} + 3^n$
2. $4 + 8 + 12 + 16 + 20 + \cdots + 4n$
3. $2232^2 + 2233n^3 + 42234n^5 + 42235$
4. $n^2 \log_{23342}(1234^{4321n})$
5. $\log(\log(\log(n^2)))$

b. (3 marks)

Let the runtime efficiency of an algorithm be defined as: $T(n) = 25 + 4n^2 + 9n^3$. Prove that $T(n)$ is $O(n^3)$ using the Big-O formal definition. Show all steps in your proof.

c. (4 marks)

Suppose that the runtime efficiency of an algorithm is defined as:

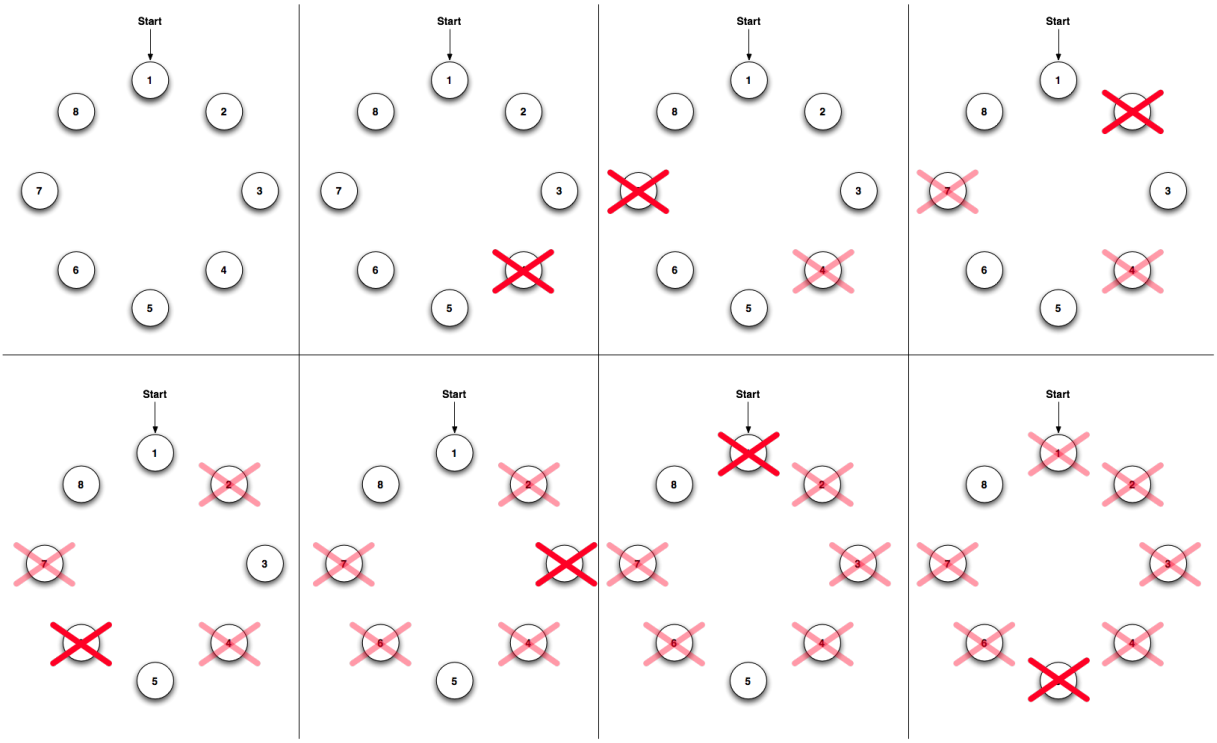
$$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=5}^{n} ij.$$

Determine the algorithm's order of growth in terms of the Big-O notation by simplifying the given expression. Show all steps of your work. You do not have to provide a formal proof using the Big-O formal definition.

2 Question 2. Algorithm Design (8 marks)

A version of the Josephus Problem can be represented as the following game: n players are sitting in a circle, with players numbered from 1 to n. The game starts with person 1 taking the game token. After that, the token is passed around. After m passes, the person holding the token is eliminated. The passing of the token and the elimination after m passes continues until one person remains. The last remaining person is the winner.

For instance, for $n = 6$ and $m = 0$, the elimination proceeds as $[1, 2, 3, 4, 5]$ with person 6 as the winner. For $n = 8$ and $m = 3$, the elimination proceeds as $[4, 7, 2, 6, 3, 1, 5]$ with person 8 as the winner, as shown in the diagram below.



Design an algorithm that takes as input two integers, n and m as specified above, and outputs a single integer representing the position of the game winner. You may write your function in pseudocode (e.g., "for each element in a list L"), but you need to specify each step unambiguously (e.g., "return L[i];").

You may utilize operations of any of the abstract data types (ADTs) discussed in class, such as List ADT, without implementing the details. However, you need to clearly explain how the chosen ADT is implemented (e.g., as a doubly linked list), and you need to make appropriate function calls (e.g., insert(value, position)).

Once you have designed the algorithm, state its runtime efficiency in terms of the Big-O notation as a function of n, and briefly explain how you have derived this classification.

3 Question 3. Pointers and Dynamic Memory (8 marks)

Examine the following code fragment. For each line marked with the [*Question*∗] tag, state what is the output, and briefly explain why that output occurs. If a line represents a compilation error, state that instead, and briefly explain why the error occurs. Assume that no lines will be skipped.

```
    int* p = new int(3);
    int* q = new int(2);
    int* r = new int(42);
    cout << *p << " and " << *q << endl; // [Question1]

    cout << r << endl; // outputs 0x555111
    cout << r + *q << endl; // [Question2]

    cout << p + r << endl; // [Question3]

    Node *node = new Node(5); // Node includes the data item of int type
    delete node;
    node = NULL;
    cout << node->getData() << endl; //[Question4]
    ...
}
```

Output and explanations:

[*Question*1]

[*Question*2]

[*Question*3]

[*Question*4]

4 Question 4. Divide and Conquer Algorithms (11 marks)

Consider the following function named *woofSuper*:

```
void woofSuper(int K, int firstWoof, int lastWoof) {
    int segment = (lastWoof - firstWoof) / 6;
    if (lastWoof <= firstWoof)
        cout << "WOOF!" << endl;
    else if (K <= (firstWoof + segment * 3)) {
        woofSuper(K, firstWoof, firstWoof + segment);
        woofSuper(K, firstWoof + segment + 1, firstWoof + segment * 2);
        woofSuper(K, firstWoof + segment * 2 + 1, firstWoof + segment * 3);
    } else {
        woofSuper(K, firstWoof + segment * 3 + 1, firstWoof + segment * 4);
        woofSuper(K, firstWoof + segment * 4 + 1, firstWoof + segment * 5);
        woofSuper(K, firstWoof + segment * 5 + 1, lastWoof);
    }
}
```

a. (4 marks)

Determine the recurrence relationship for the runtime efficiency $T(n)$ of this function, where $n = lastWoof - firstWoof$ for $n \geq 0$.

b. (7 marks)

Solve the recurrence relationship for this function 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 do not have to provide a formal proof using the Big-O formal definition. Show all steps in deriving your solution.

5 Question 5. Data Structure Design (12 marks)

ACME Inc is a software company that focuses on the healthcare domain. We are tasked with programming a data structure for storage of patient records for one of ACME Inc's software products called PRx.

Each patient record will be stored as an instance of the *PatientRecord* class that encapsulates the needed data and behaviour. Each record has a category ID, which is stored as an unsigned integer (e.g., 4562), and a patient ID, which is also stored as an unsigned integer (e.g., 424242). Inside each *PatientRecord*, we will also store the patient's name, address, and date of birth; all of these are to be stored as *string* values.

RecordsManager class will be used to handle *PatientRecord* objects, and it will include relevant pointers to a collection of *PatientRecord* objects. As new patient records are obtained, they are to be inserted into the appropriate *RecordsManager* instance, and the data structure has to facilitate fast insertion of new records without delay for data structure resizing. The records also have to remain in a sorted ascending order, where they are first ordered by the category ID and then by the patient ID.

a. (5 marks) Use doubly linked list from *Assignment* #1 as the basis for your implementation, and provide the declarations for member attributes and methods, including getter/setter methods as needed, that are needed to make *PatientRecord* and *RecordsManager* function. For *PatientRecord*, you should also include at least one parametric constructor.

```
class PatientRecord {
    unsigned int categoryID;
    unsigned int patientID;
    // fill in other required member attributes and methods below
    // members should be declared as public or private as appropriate
```

```
};

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

    bool insertRecord(PatientRecord* record);
    bool removeRecord(unsigned int categoryID, unsigned int patientID);
    PatientRecord* findRecord(unsigned int categoryID, unsigned int patientID);
};
```

b. (7 marks) Write the method $PatientRecord :: insertRecord(PatientRecord* \ record)$. The method inserts the given record into the linked list, so that the list remains in a sorted ascending order. The records need to be first ordered by the category ID and then by the patient ID. The combination of the category ID and patient ID needs to remain unique, so if another record already exists with exactly the same category ID and patient ID, the insertion should fail and return $false$; otherwise, the method should return $true$.

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. Include appropriate error checking in your code, and adequately document your code. Marks will be deducted for implementations that are difficult to read, or that are not adequately documented.

```
bool PatientRecord::insertRecord(PatientRecord* record) {
```

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.