# University of Waterloo
# Midterm Examination Solutions
# 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) | 12 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. $\log(\log(\log(n^2)))$

**Solution:**

1.
$$\sum_{i=1}^{n} 3^i = \frac{3^{n+1} - 1}{3 - 1} - 1 = \frac{3^{n+1} - 3}{2} = O(3^n) \tag{7}$$

2.
$$4\sum_{i=1}^{n} i = 4\frac{n(n+1)}{2} = 4\frac{n^2 + n}{2} = O(n^2) \tag{8}$$

3.
$$2232^2 + 2233n^3 + 42234n^5 + 42235 = O(n^5) \tag{9}$$

4.
$$n^2 \log_{23342}(1234^{4321n}) = 4321n^3 \log_{23342}(1234) = O(n^3) \tag{10}$$

5.
$$\log(\log(\log(n^2))) = O(\log(\log(\log(n^2)))) \tag{11}$$

**Ordering:** (5) $O(\log(\log(\log(n^2))))$ < (2) $O(n^2)$ < (4) $O(n^3)$ < (3) $O(n^5)$ < (1) $O(3^n)$.

**Grading scheme:** Subtract 1 mark for each equation that is out of place, down to 0 marks.

**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.

**Solution:**

Let $25 + 4n^2 + 9n^3 \leq Kn^3$. Then, let $n_0 = 1$. It follows that $25 + 4n^2 + 9n^3 = 38 \leq K$. From there, $38 \leq K$. Hence, for $K = 38$ and $n \geq n_0 = 1$, it follows that $T(n) = O(n^3)$.

**Grading scheme:** 2 marks for deriving $K$ and $n_0$, and 1 mark for the correct final answer.

**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.

**Solution:**

$$\begin{aligned} T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=5}^{n} ij = \\ \sum_{i=1}^{n} \sum_{j=1}^{n} ij(n-4) = \\ \sum_{i=1}^{n} i(n-4)\frac{n(n+1)}{2} = \\ \frac{(n-4)n^2(n+1)^2}{4} = \\ \frac{(n^3-4n^2)(n^2+2n+1)}{4} = \\ \frac{(n^5-2n^4-7n^3-4n^2)}{4} = \\ O(n^5). \end{aligned} \tag{12}$$

**Grading scheme:** 0.5 marks for each step or 1 mark for two combined steps, for up to 3 marks for six derivation steps. 1 mark for the correct final answer.

# 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:**

**Solution:**

[*Question*1] 3 and 2. *p = 3 as given. *q = 2 as given.

[*Question*2] 0x555119. r + *q will require that r be incremented by two int locations since *q = 2. This will result in r being incremented by 8 since the size of int is 4 bytes.

[*Question*3] Syntax error. Adding pointers is illegal, so this will not compile.

[*Question*4] Runtime error (NULL pointer exception). Dereferencing a NULL pointer is illegal.

**Grading scheme:** 1 mark for each correct value. 1 mark for each appropriate explanation.

# 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 $n \geq 0$.

**Solution:**

$T(0) = a$

$T(n) = 3T(n/6) + b$

where $a$ is a constant that represents the number of primitive operations used in the base case, and $b$ is a constant that represents the number of primitive operations needed to execute each cycle.

**Grading scheme:** 1 mark for the base case. 3 marks for the recursive case.

**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.

**Solution:**

$$T(n) = 3\ T(n/6) + b = 3\ 3\ T(n/6^2) + 3b + b = 3\ 3\ 3\ T(n/6^3) + 9b + 3b + b = \\ ... = 3^i T(n/6^i) + \sum_{j=0}^{i-1} 3^j b = 3^i T(n/6^i) + \frac{3^i - 1}{2} b \quad (13)$$

When $n/6^i = 1$, let $i = c$. It follows that $n/6^c = 1$ and $n = 6^c$, so $c = \log_6(n)$.

From there,

$$T(n) = 3^i T(n/6^i) + \frac{3^i - 1}{2} b = 3^{\log_6(n)} T(1) + \frac{3^{log_6(n)} - 1}{2} b = \\ n^{\log_6(3)} T(1) + \frac{n^{\log_6(3)} - 1}{2} b = 3n^{\log_6(3)} T(0) + n^{\log_6(3)} b + \frac{n^{\log_6(3)} - 1}{2} b = \\ 3an^{\log_6(3)} + bn^{\log_6(3)} + \frac{n^{\log_6(3)} - 1}{2} b = O(n^{\log_6(3)}) = O(n). \quad (14)$$

**Grading scheme:** 2 marks for the unfolding to the general form, 2 marks for the computation of $c = \log_6(n)$, 1 mark for the computation of $n^{\log_6(3)}$, and 2 marks for the final derivation of O(n).

# 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. **Solution:**

```
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
    string patientName, patientAddress, patientDOB;
    PatientRecord* next, prev;
    friend class RecordsManager;

public:
    PatientRecord(unsigned int newCatID, unsigned int newPatID,
        string newName, string newAddress, string newDOB);

    unsigned int getCategoryID();
    unsigned int getPatientID();
    string getPatientName();
    string getPatientAddress();
    string getPatientDOB();
```

```
    void setCategoryID(unsigned int newCatID);
    void setPatientID(unsigned int newPatID);
    void setPatientName(string newName);
    void setPatientAddress(string newAddress);
    void setPatientDOB(string newDOB);
};

class RecordsManager {
    // fill in other required member attributes and methods below
    // members should be declared as public or private as appropriate
    PatientRecord* head, tail;
    int size;
public:
    int getSize();

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

**Grading scheme:** 2 marks for *PatientRecord* attributes including *prev* and *next*, 2 marks for the *PatientRecord* methods, and 1 marks for the *PatientRecord* attributes and methods including *head* or *tail*.

**b. (7 marks)** Write the method $RecordsManager :: insertRecord(PatientRecord* record)$. 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.

**Solution:**

```
bool RecordsManager::insertRecord(PatientRecord* record) {
    // Insert at the start if the list is empty [2 marks]
    if(!head) { // 2 marks for handling the head node
        head = record;
        tail = record;
    } else {
        PatientRecord* cur = head;

        // Find the right CID // 1 mark
        while (cur->next && cur->next->getCID() < record->getCID())
            cur = cur->next;

        // Find the right PID // 1 mark
        while (cur->next && cur->next->getCID() <= record->getCID() &&
                cur->next->getPID() < record->getPID())
            cur = cur->next;

        // Check for duplicate values // 1 mark
        if (cur->next && cur->next->getCID() == record->getCID() &&
                cur->next->getPID() == record->getPID())
            return false;

        // Insert record
        if (cur->next) { // 2 marks
            record->next = cur->next;
            cur->next->prev = record;
            cur->next = record;
            record->prev = cur;
        } else {
            cur->next = record;
            record->prev = cur;
            tail = record;
        }
```

```
        }
        // Increment the size
        ++size;

        return true;
    }
```