University of Waterloo
Midterm Examination Solutions
Spring 2017

Student Name:	_____
Student ID Number:	_____
Course Section (circle one):	**MTE 140**

Instructors:	Igor Ivkovic
Date of Exam	June 16th 2017
Time Period	8:30am-10:30am
Duration of Exam	120 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 10 minutes of the exam. Plan your time wisely to finish the exam on time.

Question 1: (12 marks)	Question 2: (12 marks)	Question 3: (12 marks)	Question 4: (14 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 (12 marks)

(approx. 25 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. Briefly explain your answer.

1. $1157n^2 + 3447n^7 + 5555 + 4284n^9$
2. $1 + 5 + 25 + 125 + \cdots + 5^{n-1} + 5^n$
3. $\log(\log(\log(n)))$
4. $6 + 18 + 54 + \cdots + 2\ 3^{n-1} + 2\ 3^n$
5. $n^4 \log_{1157}(4433^{1157n})$

Solution:
Ordering: (3) $O(\log(\log(\log(n))))$ < (5) $O(n^5)$ < (1) $O(n^9)$ < (4) $O(3^n)$ < (2) $O(5^n)$.

Grading scheme: Subtract 1 mark for each pair that is incorrect, down to 0 marks.

b. (3 marks)

Specify the runtime efficiency of the given code fragment using the Big-O notation. Explain your answer.

```
for (int i = 2; i <= n; i = i * i) {
    for (int k = 2; k <= n; k = k * k) {
        std::cout << "Hello MTE 140" << std::endl;
    }
}
```

Solution:
Each loop represents $O(\log(\log(n)))$ runtime, so the two nested loops represent the runtime of $O(\log^2(\log(n)))$.

To compute the runtime, observe that i grows based on the following series: 2^{2^0}, 2^{2^1}, 2^{2^2}, 2^{2^3}, and so on. Hence, express the loop as a formula: 2^{2^j}. Let $j = c$ and let 2^{2^c} = n as n would be the largest allowed value for the loop. From there it follows that c = $\log_2(\log_2(n))$ = $O(\log(\log(n)))$. Repeat the process for the inner loop, and multiply the two resulting expressions to obtain $O(\log^2(\log(n)))$.

c. (5 marks)

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

$$T(n) = \sum_{i=5}^{n-2} 3 \sum_{j=1}^{n} 6j^2 \sum_{k=2}^{n-5} 2.$$

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. Prove that your answer is correct either by using the Big-O formal definition or by using limits as $n \to \infty$.

Solution:

$$\begin{aligned} T(n) = \sum_{i=5}^{n-2} 3 \sum_{j=1}^{n} 6j^2 \sum_{k=2}^{n-5} 2 = \\ \sum_{i=5}^{n-2} 3 \sum_{j=1}^{n} 6j^2 2(n-6) = \\ \sum_{i=5}^{n-2} 6n(n+1)(2n+1)(n-6) = \\ 6n(n+1)(2n+1)(n-6)^2 = \\ 6(2n^3+3n^2+n)(n^2-12n+36) = \\ 12n^5 - 126n^4 + 222n^3 + 576n^2 + 216n = \\ O(n^5). \end{aligned} \tag{7}$$

(Option A.)

We must show that $12n^5 - 126n^4 + 222n^3 + 576n^2 + 216n \leq Kn^5$ for all $n >= n_0$.

Let $n_0 = 1$. It follows that $12 - 126 + 222 + 576 + 216 = 900 =\leq K$.

Hence, for $K = 900$ and $n \geq n_0 = 1$, it follows that $T(n) = O(n^5)$.

(Option B.)

$\lim_{n\to\infty} \frac{12n^5-126n^4+222n^3+576n^2+216n}{n^5} = \lim_{n\to\infty} 12 = constant.$

Hence, it follows that $T(n) = O(n^5)$.

2 Question 2. Algorithm Design (12 marks)

(approx. 35 minutes)

a. (10 marks)

You are given an array of characters, S, of length, $sizeS$. The array stores a sequence of characters drawn from $[m, t, e]$ where each of the three letters is present one or more times, and the letters appears in the order indicated (i.e., m's before t's and t's before e's). For example, *mtttteee* and *mmtee* are valid while *tmmmee* and *eeettmm* are not.

Write a function *int countSequences(char * S, int sizeS)* that will iterate through a valid sequence S, and return the count of how many different subsequences can be formed from S, so that each of m, t, and e appears at least once in the correct order. Subsequences are different if the array indices of the selected characters are different. For example, the subsequences for *mmte* are *mte*, *mte*, and *mmte*, so the answer is 3. Similarly, the subsequences for *mmtee* are *mte*, *mte*, *mte*, *mte*, *mmte*, *mmte*, *mtee*, *mtee*, and *mmtee*, so the answer is 9.

For full marks, the run-time performance of your algorithm needs to be linear (i.e., $O(n)$ where n is the length of S). 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.]

Solution:

```
int countSequences(char *s, int sizeS) {
    int mCount = 0, tCount = 0, eCount = 0;

    for (unsigned int i = 0; i < sizeS; i++) {
        if (s[i] == 'm')
            ++mCount;

        else if (s[i] == 't')
            ++tCount;

        else if (s[i] == 'e')
            ++eCount;
    }

    return ((1 << mCount) - 1) * ((1 << tCount) - 1) * ((1 << eCount) - 1);
    // computes (2^mCount - 1) x (2^tCount - 1) x (2^eCount - 1)
}

//Alternative solution
int countSequences2(char *s, int sizeS) {
    int mCount = 0, tCount = 0, eCount = 0;

    for (unsigned int i=0; i<sizeS; i++) {
        if (s[i] == 'm')
            mCount = (1 + 2 * mCount);

        else if (s[i] == 't')
            tCount = (mCount + 2 * tCount);

        else if (s[i] == 'e')
            eCount = (tCount + 2 * eCount);
    }

    return eCount;
}

int main() {
    char *s = new char[3] {'m','t','e'};
    cout << countSequences(s,3) << endl;
    cout << countSequences2(s,3) << endl;
    delete [] s; // output is 1 and 1

    s = new char[5] {'m','m','t','e','e'};
    cout << countSequences(s,5) << endl;
    cout << countSequences2(s,5) << endl;
    delete [] s; // output is 9 and 9
}
```

b. (2 marks)

Once you have designed the algorithm, explain in natural language — not code — how would you test the algorithm. That is, list and briefly explain at least five test cases that you would use to ensure correctness of your algorithm and enhance its robustness. For example, one of the test cases given above could be explained as "test a non-empty character array where m and e are each present twice and t is present once in the correct order". You are not responsible for implementing the code changes that would reflect your test cases.

Solution:

```
[Test for S being NULL; 0.5 mark]
1. S is NULL, sizeS is 0

[Test for cases where the array size and sizeS values do not match; 0.5 mark]
2. S is empty, sizeS is not 0
3. S is not empty, sizeS is 0

[Test for S with invalid combinations; 0.5 mark]
4. S is not empty, sizeS is 3, S equals etm
5. S is not empty, sizeS is 5, S equals teemm

[Test for S being empty; 0.5 mark; optional or as alternative to above]
6. S is not NULL and empty, sizeS is 0

[Test for S with valid combination of varied sizes; 0.5 mark]
7. S is not empty, sizeS is 3, S equals mte
8. S is not empty, sizeS is 4, S equals mmte
9. S is not empty, sizeS is 4, S equals mtte
10. S is not empty, sizeS is 4, S equals mtee
...

[Other equivalence classes may be accepted as appropriate]
```

3 Question 3. Divide and Conquer Algorithms (12 marks)

(approx. 25 minutes)
Consider the following function:

```
void woofNG(int K, int firstWoof, int lastWoof) {
    int segment = (lastWoof - firstWoof) / 4;
    if (lastWoof <= firstWoof) {
        cout << "WOOF Complete!" << endl;
        return;
    }

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

   if (K <= (firstWoof + segment * 2)) {
        woofNG(K, firstWoof + 1, firstWoof + segment);
        woofNG(K, firstWoof + segment + 1, firstWoof + segment * 2);
    } else {
        woofNG(K, firstWoof + segment * 2 + 1, firstWoof + segment * 3);
        woofNG(K, firstWoof + segment * 3 + 1, lastWoof);
    }
}
```

a. (3 marks)
Draw the call tree for this function when $woofNG(42, 45, 75)$ is called.

```
                (42,45,75)
     (42,46,52)              (42,53,59)
(42,47,47) (42,48,48)  (42,54,54)  (42,55,55)
```

Grading scheme: 0.5 mark for each correct entry.

b. (3 marks)
Determine the recurrence relationship for the runtime efficiency $T(n)$ of this function, where $n = lastWoof - firstWoof$ for $n \geq 0$.

Solution:
$T(0) = a$
$T(n) = 2T(n/4) + n/4 + b$
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. 2 marks for the recursive case.

b. (6 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:

$$\begin{aligned}
T(n) = 2T(n/4) + n/4 + b = \\
2\ 2T(n/4^2) + n/8 + n/4 + 2b + b = \\
2\ 2\ 2T(n/4^3) + n/16 + n/8 + n/4 + 4b + 2b + b = \\
... = 2^i T(n/4^i) + n/4 \sum_{j=0}^{i-1} (1/2)^j + b \sum_{j=0}^{i-1} 2^j = \\
2^i T(n/4^i) + n/4 \frac{(1/2)^i - 1}{1/2 - 1} + b(2^i - 1) = \\
2^i T(n/4^i) + n/2(1 - (1/2)^i) + b(2^i - 1)
\end{aligned} \tag{8}$$

When $n/4^i = 1$, let $i = c$. It follows that $n/4^c = 1$ and $n = 4^c$, so $c = \log_4(n)$.

From there,

$$\begin{aligned}
T(n) = 2^i T(n/4^i) + n/2(1 - (1/2)^i) + b(2^i - 1) = \\
2^{\log_4(n)} T(1) + n/2(1 - (1/2)^{\log_4(n)}) + b(2^{\log_4(n)} - 1) = \\
n^{1/2}(2T(0) + 1/4 + b) + n/2 - n/2\ n^{-1/2} + bn^{1/2} - b = \\
2an^{1/2} + 1/4n^{1/2} + bn^{1/2} + n/2 - n^{1/2}/2 + bn^{1/2} - b = \\
O(n).
\end{aligned} \tag{9}$$

Grading scheme: 2 marks for the unfolding to the general form, 2 marks for the computation of $c = \log_4(n)$, 2 mark for the final derivation of $O(n)$.

4 Question 4. Data Structure Design (14 marks)

(approx. 35 minutes)
We are designing a web application for a small art auction house that handles various types of artwork.

Each artwork is represented by an instance of *Artwork* class. For each piece of artwork, we need to store the artist name, year it was made, title, and type of art (e.g., painting, sculpture, photograph); year it was made is stored as an unsigned integer while all other attributes are stored as *string* values.

ArtManager class is used as a container to handle *Artwork* objects, and it includes relevant attributes/fields; *ArtManager* is modeled as a doubly linked list. Individual *Artwork* objects need to be kept sorted first by the artist name, then by the year they were made, and finally by the title.

a. (4 marks) Using doubly linked list from *Assignment* #1 as the basis for your implementation, provide the declarations for member attributes and methods, including getter/setter methods, that are needed to make *Artwork* and *ArtManager* function. For *ArtManager*, you should also include at least one parametric constructor. **Solution:**

```
class Artwork {
    // fill in other required member attributes and methods below
    // members should be declared as public or private as appropriate
    unsigned int yearMade;
    string artistName, artTitle, artType;
    Artwork* next;
    Artwork* prev;
friend class ArtManager;
public:
    Artwork() : next(NULL), prev(NULL) { }
    Artwork(string nArtistName, unsigned int nYearMade,
        string nArtTitle, string nArtType) :
        artistName(nArtistName), yearMade(nYearMade),
        artTitle(nArtTitle), artType(nArtType),
        next(NULL), prev(NULL) { }

    unsigned int getYearMade();
    string getArtistName();
    string getArtTitle();
    string getArtType();

    void setYearMade(unsigned int nYearMade);
    void setArtistName(string nArtistName);
    void setArtTitle(string nArtTitle);
    void setArtType(string nArtType);
};
```

```
class ArtManager {
    // fill in other required member attributes and methods below
    // members should be declared as public or private as appropriate
    Artwork* head;
    Artwork* tail;
    int size;

public:
    ArtManager();
    ArtManager(Artwork* nArtWork);
    ~ArtManager();

    int getSize();
    void print() const;

    bool insertArtwork(Artwork* record);
    bool removeArtwork(string artistName, string title);
    Artwork* findArtwork(string artistName, string title);
};
```

b. (3 marks) Implement one of the constructors and the destructor for the $ArtManager$ class. Include appropriate initialization and deinitialization of member variables, respectively. **Solution:**

```
 // 1 mark for the constructor
ArtManager::ArtManager() : head(NULL), tail(NULL), size(0) {};
OR
ArtManager::ArtManager(Artwork* nArtWork) : head(nArtWork), tail(nArtWork), size(0) {};

ArtManager::~ArtManager() {
    while (head)  {
        Artwork* tmp = head;
        head = head->next;
        delete tmp;
    }
} // 2 marks for the destructor
```

c. (7 marks) Write the method $ArtManager :: insertArtwork(Artwork* record)$. The method inserts the given artwork record into the linked list, so that the list remains in a sorted ascending order. The records need to be first ordered by the artist name, then by the year artwork was made, and then by the title. The combination of the artist name and title needs to remain unique, so if another record already exists with exactly the same artist name and title, 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 ArtManager::insertArtwork(Artwork* record) {
    if(!record)
        return false;

    if(!head) { // 0.5 mark for handling empty list
        head = record;
        tail = record;
    } else {
        Artwork* cur = head;

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

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

        // Find the right artTitle // 1 mark
        while (cur && cur->artistName <= record->artistName
            && cur->yearMade <= record->yearMade && cur->artTitle < record->artTitle)
            cur = cur->next;
```

```
            // Check for duplicate values // 0.5 mark
            if (cur && cur->artistName == record->artistName
                && cur->yearMade == record->yearMade && cur->artTitle == record->artTitle)
                return false;

            // Insert record
            if (cur == head) { // 1 mark
                head = record;
                record->next = cur;
                cur->prev = record;
            } else if (cur) { // 1 mark
                cur->prev->next = record;
                record->prev = cur->prev;
                record->next = cur;
                cur->prev = record;
            } else { // 1 mark
                tail->next = record;
                record->prev = tail;
                tail = record;
            }

        }
        // Increment the size; award 0.5 as an alternative to duplicates or empty list
        ++size;

        return true;
    }
```

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.