# University of Waterloo
# Midterm Examination Sample Solutions
# Spring 2015

| | |
|---|---|
| **Student Name:** | _____ |
| **Student ID Number:** | _____ |
| **Course Section:** | **MTE 140** |

| | |
|---|---|
| Instructor: | Igor Ivkovic |
| Date of Exam | June 18th 2015 |
| Time Period | 11:00am-1:00pm |
| Duration of Exam | 2 hours |
| Pages (including cover) | 13 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: (12 marks) | Question 3: (8 marks) | Question 4: (10 marks) | Question 5: (15 marks) |
| Total: (55 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 (10 marks)

**a. (5 marks)**

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

1. $1 + 2 + 3 + 4 + 5 + \cdots + (n-1) + n$
2. $n \log(\log(\log(42n)))$
3. $1 + 2 + 4 + 8 + 16 + \cdots + 2^{n-1} + 2^n$
4. $n \log(4242n)$
5. $444n^4 + 333n^3 + 222n^2$

**Solution:**

1.

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

2.

$$n \log(\log(\log(42n))) = O(n \log(\log(\log(n)))) \tag{8}$$

3.

$$\sum_{i=0}^{n} 2^i = \frac{2^{n+1}-1}{2-1} = 2^{n+1} - 1 = O(2^n) \tag{9}$$

4.

$$n \log(4242n) = O(nlog(n)) \tag{10}$$

5.

$$444n^4 + 333n^3 + 222n^2 = O(n^4) \tag{11}$$

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

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

**b. (5 marks)**

Suppose that the runtime efficiency of an algorithm is defined as:
$T(n) = 4log_2(n) + 8log_2(n) + 12log_2(n) + ... + 4(n-1)log_2(n) + 4nlog_2(n)$.
Determine the algorithm's order of growth in terms of the Big-O notation, and prove that this is indeed the true order of growth using the Big-O formal definition. Show all steps in your proof.

**Solution:**

$$\begin{aligned} T(n) = 4log_2(n) + 8log_2(n) + 12log_2(n) + ... + 4(n-1)log_2(n) + 4nlog_2(n) = \\ 4log_2(n)\sum_{i=1}^{n} i = \\ \frac{4log_2(n)n(n+1)}{2} = \\ 2n^2log_2(n) + 2nlog_2(n) \end{aligned} \tag{12}$$

Let $2n^2log_2(n) + 2nlog_2(n) \leq Kn^2log_2(n)$.

Then, let $n_0 = 2$. It follows that $2n^2log_2(n)+2nlog_2(n) = 12 \leq 4K$. From there, $3 \leq K$

Hence, for $K = 3$ and $n \geq 2 = n_0$, it follows that $T(n) = O(n^2log_2(n)) = O(n^2log(n))$.

**Grading scheme:** 2 marks for deriving $2n^2log_2(n) + 2nlog_2(n)$, 2 marks for deriving $K$ and $n_0$, and 1 mark for the correct final answer.

# 2 Question 2. Algorithm Design (12 marks)

Design a linear-time $O(n)$ algorithm that iterates through a sorted sequential list $L$ of size $n$. The list contains distinct positive integers in increasing order that represent $x$ coordinates in a plane, such as the list $[1, 3, 11, 16, 21, 35]$ of size 6. The algorithm outputs the coordinate for which the maximum distance from any other coordinate in the list is minimal. In $[1, 3, 11, 16, 21, 35]$, the coordinate 16 is the closest to all other coordinates, and for 16 the maximum distance to any other coordinate is 19. The second-best coordinate is 21 for which the maximum distance to any other coordinate is 20. You may write your function in pseudocode (e.g., "for each element in L"), but you need to specify each step unambiguously (e.g., "return L[i];").

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. For full marks, your algorithm needs to run in linear-time $O(n)$. For partial marks, your algorithm may grow faster than linear time $O(n)$.

**Solution:**

```
int findClosestCoordinate(int L[], int size) {
    if (size <= 0) return -1; // empty list
    if (size <= 2) return L[0]; // one-element or two-element lists
                                // return the first element

    int mid = (L[0] + L[size-1]) / 2; // compute the mid value

    int iterator = 0;
    while (L[iterator] < mid) // iterate until the mid value is reached
        ++iterator;

    // decide if the element after the mid or before mid is the closest
    if (L[iterator] - L[0] < L[size-1] - L[iterator-1])
        return L[iterator];
    else
        return L[iterator-1];
}
```

The program has only one loop that at worst goes from the first to the last element (i.e., n - 1 + 1 = n cycles), and in each cycle, the loop runs a fixed number (b) of operations. This results in $bn$ operations. Outside of the loop, there is a fixed number (a) of operations. Hence, the overall run-time of this algorithm is $bn + a = O(n)$.

**Grading scheme:** 1 mark for handling the empty list, 2 marks for computing the mid value, 2 marks for iterating until the mid point, 4 marks for deciding if the element before or after the mid is the closest, 2 marks for discussing Big-O complexity and 1 mark for providing the correct value.
For solutions that are worse than O(n), 1 mark for handling the empty list, 4 marks for correct implementation, 2 marks for discussing Big-O complexity and 1 mark for providing the correct value.

# 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. You can assume that each line that crashes the program will be commented out or skipped, so all of the given lines will be attempted at least once.

```
    int* p = new int(15);
    int* q = new int(25);
    *q = 42;
    *p = *q;
    cout << *p << " and " << *q << endl; // [Question1]

    cout << p + q << endl; // [Question2]

    *q = 1;
    cout << p << endl; // output 0x555111
    cout << p + *q << endl; // [Question3]

    Node *node = new Node(5); // Node includes the data item of int type
    delete node;
    Node *node2 = new Node(7);
    cout << node->getData() << endl; //[Question4]
                // For this last line, we intentionally wrote node-> and not node2->
                // Also note that this line will not crash the program in Dev-C++
    ...
}
```

**Output and explanations:**

**Solution:**

[*Question*1]
42 and 42. *q = 42 copies 42 into the memory to which q points. *p = *q copies 42 into the memory to which p points.
[*Question*2]
Syntax error. Adding pointers is illegal, so this will not compile.
[*Question*3]
0x555115. p + *q will require that p be incremented by one int location since *q = 1. However, this will result in p being incremented by 4 since the size of int is 4 bytes.
[*Question*4]
7. node is a dangling pointer. After the heap memory for node gets deallocated, it will get filled by another Node object with new Node(7), and inside that object, data = 7. Since node still points to the same memory location, accessing its data element will return 7.

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

# 4 Question 4. Divide and Conquer Algorithms (10 marks)

Consider the following function named $woofPlus$:

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

$T(0) = a$

$T(n) = 4T(n/4) + b$

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. 3 marks for the recursive case.

**c. (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:**

$$T(n) = 4\ T(n/4) + b = 4\ 4\ T(n/4^2) + 4b + b = 4\ 4\ 4\ T(n/4^3) + 16b + 4b + b = \\ ... = 4^i T(n/4^i) + \sum_{j=0}^{i-1} 4^j b = 4^i T(n/4^i) + \frac{4^i - 1}{3} b \quad (13)$$

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,

$$T(n) = 4^i T(n/4^i) + \frac{4^i - 1}{3} b = \\ 4^{\log_4(n)} T(1) + \frac{4^{log_4(n)} - 1}{3} b = nT(1) + \frac{n-1}{3} b = \\ 4nT(0) + nb + \frac{n-1}{3} b = 4an + bn + \frac{n-1}{3} b = O(n). \quad (14)$$

**Grading scheme:** 2 marks for the unfolding to n = 1 case, 2 marks for the computation of $c = \log_4(n)$, and 2 marks for the final derivation of O(n).

# 5 Question 5. Data Structure Design (15 marks)

We are tasked with programming a data structure interface for storage of shipping containers at a port. The shipping containers arrive by ships, and are stored via cranes on top of each other. The last container to be stored is the first one to be removed.

Each container will be stored as an instance of *ShippingContainer* class that encapsulates the needed data and behaviour. Each container has an RFID tag, so once it is deposited, we will also store its x, y, and z coordinates as *double* values. For each container, we will also store unique ID, owner's name, country of origin, and date deposited; all of these to be stored as *string* values.

*ContainerHandler* class will be used to handle *ShippingContainer* objects, and it will include relevant pointers to a collection of container objects.

**a. (2 marks)** Based on the abstract data types (ADT) presented in class, which ADT would be most appropriate for this problem? Explain your answer.
**Solution:**

Answer: Stack ADT. The question specified Last-In First-Out (LIFO) principle of storing and retrieving containers, and stacks are based on the LIFO principle.

**Grading scheme:** 1 mark for the correct answer, and 1 mark for the explanation.

**b. (6 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 *ShippingContainer* and *ContainerHandler* function. For *ShippingContainer*, you should also include at least one parametric constructor. **Solution:**

```
class ShippingContainer {
    string containerID;
    // fill in other required member attributes and methods below
    // members should be declared as public or private as appropriate
    double x, y, z;
    string ownersName, originCountry, dateDeposited;
    ShippingContainer* next, prev;

public:
    ShippingContainer(string newID, double newX, double newY, double newZ,
        string newOwner, string newCountry, string newDepositDate);

    string getContainerID();
```

```
    double getCurX(); int getCurY(); int getCurZ();
    string getOwnersName();
    string getOriginCountry();
    string getDateDeposited();

    void setOwnerID(string newID);
    void setCurX(double newX); void setCurY(double newY);
    void setCurZ(double newZ);
    void setOwnersName(string newOwnersName);
    void setOriginCountry(string newOriginCountry);
    void setDateDeposited(string newDateDeposited);
};

    class ContainerHandler {
    // fill in other required member attributes and methods below
    // members should be declared as public or private as appropriate
    ShippingContainer* head, tail; // only one of head or tail is needed
                                   // since the list will be used as the stack
                                   // from whatever side is declared as the top
    int size;
public:
    int getSize();

    bool insertContainer(ShippingContainer* container);
    bool removeContainer();
    ShippingContainer* findContainer(string containerID);
};
```

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

**c. (7 marks)** Write the method $ContainerHandler :: findContainer(string\ containerID)$. The method searches for the container with the matching ID, and returns a pointer reference to it. If such container cannot be found, it returns $NULL$ instead. You can assume that the operators == and ! = are available for strings.

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:**

```
ShippingContainer* ContainerHandler::findContainer(string newContainerID) {
    if(head) {// check that the list is not empty [2 marks]
                // also acceptable if accessed from the tail side
                // but in that case need to use node->prev below
        Node *node = head;
        while(node) { // loop iterator [1 mark]
            // check the current node's conainerID [2 marks]
            if(node->containerID == newContainerID)
                return node;
            node = node ->next; // move to the next node [2 marks]
        }
    }
    return NULL;
}
```

**Grading scheme:** See the grading scheme above.

# 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.