
University of Waterloo

Midterm Examination

Winter 2015

Student Name: _____

Student ID Number: _____

Course Section (circle one): BME 122 | MTE 140 | SYDE 223

Instructors: Alexander Wong and Igor Ivkovic

Date of Exam March 2nd 2015

Time Period 5:00pm-7:00pm

Duration of Exam 2 hours

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.

Question 1: (10 marks)	Question 2: (10 marks)	Question 3: (15 marks)	Question 4: (15 marks)
Total: (50 marks)			

Useful Formulas

- For $S = a_1 + (a_1 + d) + \dots + (a_n - d) + a_n$
(S is a series that goes from a_1 to a_n in d -size increments),

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

-

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

-

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

-

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

-

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

-

$$\log_b x = \frac{\log_c x}{\log_c b} \quad (6)$$

1 Question 1. Algorithm Complexity (10 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. $3 + 9 + 27 + 81 + 243 + \dots + 3^{n-1} + 3^n$
2. $15234n \log_{897124}(5551212^{5551212n})$
3. $42232^2 + 42233n^7 + 42234n^5 + 42235$
4. $12342n \log_{234235}(43325n)$.

b. (6 marks)

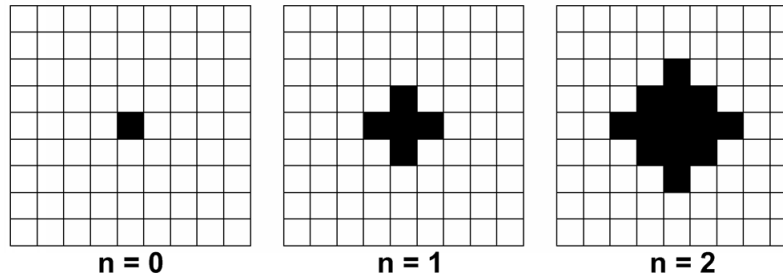
Suppose that the runtime efficiency of an algorithm is defined as:

$$T(n) = 14n^3 + 14n(n-1)^2 + \dots + 126n + 56n + 14n.$$

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.

2 Question 2. Algorithm Design (10 marks)

You are asked to implement a non-recursive $O(n^2)$ function named *drawCells* that starts by drawing one cell on its first pass, and each of its iterations adds new cells all around its sides. For $n = 0$, it just draws one cell. For $n = 1$, it draws the cells adjacent (top/bottom/left/right) to the cell created for $n = 0$. For $n = 2$, it draws the cells adjacent to the cells created for $n = 1$. It then continues iterating until it reaches the given value of n . See the illustration of this algorithm below.



Write code to implement this function as *void drawCells(int n)*. The function will start drawing cells at coordinates $(0,0)$. To draw a cell at coordinates (x,y) and fill its pixels, you are being provided with the function *void fill(x,y)* that you can call as needed. You may only use *iostream* library in your implementation, and no other external libraries. If you need any other helper functions, you will need to implement them.

3 Question 3. Divide and Conquer Algorithms (15 marks)

Consider the following function named *woof*:

```
void woof(int K, int firstDog, int lastDog) {
    int segment = (lastDog - firstDog) / 3;
    if (lastDog <= firstDog)
        cout << "Dog " << firstDog << " says WOOF!\n";
    else {
        if (K <= firstDog + segment)
            woof(K, firstDog, firstDog + segment);
        else if (K <= firstDog + segment * 2)
            woof(K, firstDog + segment + 1, firstDog + segment * 2);
        else
            woof(K, firstDog + segment * 2 + 1, lastDog);
    }
}
```

a. (5 marks)

Draw the call tree for this function when *woof*(35, 9, 47) is called.

b. (4 marks)

Determine the recurrence relationship for the runtime efficiency $T(n)$ of this function, where $n = \text{lastDog} - \text{firstDog}$ for $n \geq 0$.

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.

4 Question 3. Data Structure Design (15 marks)

Automated parking systems are a reality in many cities around the world. The idea is simple: maximize the utility of space available for parking, and minimize the time required to find a parking spot, by using an automated and mechanical system. How does the system work? A driver deposits their car at one of the entrance locations, and instead of a human moving the car, it is the robotic trolley that moves the car to an appropriate location. When the driver comes back to pick up their car, the same robotic trolley moves the car to the entrance location. The driver then enters the car and drives away.

Your job is to implement data structures with methods that will allow this single-level automated parking garage to function. Your implementation needs to be based on a doubly linked list, and it needs to ensure that the cars that have been deposited first are also retrieved first. To that end, the class *Car* is a container data type that stores details about the car, such as *ownerID*, *licencePlate*, and *manufacturer*. All of these attributes are stored as *strings*. Each *Car* also stores the details of the vehicle's current location and the new/destination location for when it is being moved. The current location is represented as integers *curRow* and *curColumn* for the current location's row and column respectively, and the new location is represented as integers *newRow* and *newColumn* for the destination location's row and column respectively. Finally, each *Car* object also stores relevant attributes needed for the linked data implementation to function.

In addition to the class *Car*, you are being asked to implement the class *RoboticTrolley*, which includes pointers to the linked data structure of *Car* objects, and which controls the robotic machine. *RoboticTrolley* also includes pointers to a data structure used to keep track of free spaces in the garage, but you are not responsible for implementing this data structure since it is out of scope of this question.

- 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.
- b. (6 marks) Fill in the missing information for the following interfaces using appropriate linked data representations for *Car* and *RoboticTrolley*. Include all of the necessary member attributes and methods. You should include at least one constructor, and getter/setter methods as needed.

```
class Car {  
    string ownerID;  
    // fill in other required member attributes and methods below  
    // members should be declared as public or private as appropriate
```

```
};
```

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

```
public:  
    bool AssignLocation(Car* car);  
    bool MoveCar(Car* car);  
    bool DepositCar(Car* car);  
};
```

To assign a vehicle to a location, the method `bool RoboticTrolley :: AssignLocation(Car* car)` is used. The method takes a car object pointer as input, and changes its new location values (`newRow` and `newColumn`) to a free location inside the garage. If the car is being moved back to the entrance, it will assign a free location at the entrance instead. If the location assignment fails, for example due to the lack of space, the location values are unchanged, and the method returns `false`; otherwise, the method returns `true`.

To move the vehicle to a location, the method `bool RoboticTrolley :: MoveCar(Car* car)` is used. The method takes a car object pointer as input, and moves the car to the new location (`newRow` and `newColumn`). After the car has been moved, its current location equals its new location. If the car could not be moved to its destination (e.g., due to a mechanical fault), the method returns `false`, and the car is returned back to its current location; otherwise, the method returns `true`.

You do not have to implement `AssignLocation` and `MoveCar` methods, and you may just call them in your implementations below.

- c. (7 marks) Write the method `bool RoboticTrolley :: DepositCar(Car* car)`, which takes a `Car` object pointer as input. It assigns the vehicle to a new physical location, moves the vehicle into the assigned location, and inserts the data item into the linked list. If the car could not be deposited, the method returns `false`; otherwise it returns `true`.

You may only use `iostream` and `string` libraries in your implementation, and no other external libraries. 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 RoboticTrolley::DepositCar(Car* car) {
```

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.