University of Waterloo
Midterm Examination Solutions
Winter 2019

Student Name:	_____
Student ID Number:	_____
Course Section:	**SYDE 223**

Instructors:	Dr. Igor Ivkovic
Date of Exam	Winter 2019
Time Period	11:30am-1:30pm
Duration of Exam	120 minutes
Pages (including cover)	15 pages (two scrap 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: (15 marks)	Question 3: (10 marks)	Question 4: (15 marks)
Total: (50 marks)			

Useful Formulas

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.

- 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),

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

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

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

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

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

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

- $$x^{\log_b y} = y^{\log_b x} \tag{7}$$

- $$\log_b xy = \log_b x + \log_b y \tag{8}$$

- $$\log_b \frac{x}{y} = \log_b x - \log_b y \tag{9}$$

1 Question 1. Algorithm Complexity (10 marks)

(approx. 20 minutes)

a. (4 marks)

Provide a code fragment in pseudocode that would exactly exhibit the runtime of: $O(log(log(n^2)) + n\ log(n))$ with no expression simplification.

For instance, for O(n) runtime, you could write the following:

```
loop from 1 to n {
    call swap(a, b) that performs constant number of steps
}
```

Solution:

```
int n = 45, step_count = 0;
for (int k = 2; k < n * n; k = k * k) { // 1 mark for k < n^2, 1 mark for k = k * k
    ++step_count;
    cout << "step1" << endl;
}
cout << step_count << endl; // corresponds to log_2(log_2(n^2))

step_count = 0;
for (int i = 0; i < n; ++i) { // 1 mark for this loop
    for (int k = 1; k < n; k = k * 2) { // 1 mark for this loop
        ++step_count;
        cout << "step2" << endl;
    }
}
cout << step_count << endl; // corresponds to n log_2(n)
```

Grading scheme: See above.

b. (6 marks)

Let the runtime efficiency of an algorithm be defined as:
$T(n) = 21log_4(343) + 42log_4(343) + 84log_4(343) + ... + 21log_4(343)2^{n-1}$.

Derive and express using Big-O notation a tight upperbound for $T(n)$ using limits as $n \to \infty$ or using the Big-O formal definition. Show all steps in deriving your solution.
Solution:

$$\begin{aligned} T(n) = 21\ log_4(343) + 42log_4(343) + 84log_4(343) + ... + 21log_4(343)2^{n-1} = \\ [1\ mark]\ 21\ log_4(343)(1 + 2 + 4 + ... + 2^{n-1}) = \\ [1\ mark]\ 21\ log_4(343)\sum_{i=0}^{n-1} 2^i = \\ [1\ mark]\ 21\ log_4(343)\frac{2^n - 1}{2 - 1} = \\ [0.5\ mark]\ 21\ log_4(343)2^n - 21\ log_4(343) = [0.5\ mark]\ O(2^n). \end{aligned} \tag{10}$$

(Option A.)
$\lim_{n\to\infty} \frac{21\ log_4(343)2^n - 21\ log_4(343)}{2^n} = \lim_{n\to\infty} 21\ log_4(343) = constant.$

Hence, it follows that $T(n) = O(2^n)$.

(Option B.)

We must show that $21\ log_4(343)2^n - 21\ log_4(343) \leq K2^n$ for all $n >= n_0$.

From $21\ log_4(343)2^n - 21\ log_4(343) \leq K2^n$, it follows that $21\ log_4(343) - \frac{21\ log_4(343)}{2^n} \leq K$.

Let $n >= n_0 = 1$. It follows that as $n \uparrow$, $21\ log_4(343) - \frac{21\ log_4(343)}{2^n} \leq 21\ log_4(343) < 105$.

Hence, for $K = 105$ and $n \geq n_0 = 1$, it follows that $T(n) = O(2^n)$.

Grading scheme: 4 marks for the derivation of 2^n as tight upper bound. 2 marks for the correct verification using limits as $n \to \infty$ or using the Big-O formal definition.

2 Question 2. Algorithm Design (15 marks)

(approx. 40 minutes)

a. (10 marks)

A friend sent you an important text message. Unfortunately, this text message got distorted in transmission. What you have received instead is a string of random characters.

Write the function

```
int has_original_message(string message, string garbled_text)
```

that will use recursion to check if the original message, which you have later received through other means, is contained in the garbled text.

If the message is contained in the text without changing the order of the letters, then your function will return the number of extra characters that got inserted beyond the original message. If the message is not in the text, then your function will simply return -1.

For example, if the original message equals *run* and the text equals *erfuinnns* then your function needs to return 6, which is the number of characters that need to be removed from the text to derive the original message. Similarly, if the original message equals *bob* and the text equals *bcccddoffbbbb* then your function needs to return 10. However, if the original message equals *let* and the text equals *tellbbg* then your function needs to return -1. Furthermore, if the original message equals *clue* and the text equals *cluuu* then your function needs to return -1.

You may assume that both strings will contain only lower-case letters, and you may assume that the garbled text will always be larger in size than the original message.

Hint: Consider writing the overloaded recursive function

```
bool has_original_message(string message, string text, int size1, int size2,
int& distance)
```

that will be called from your non-recursive function. This function will return *true* if the message is contained within the garbled text, and will also count the difference in number of characters using the *distance* accumulator.

You may utilize operations of any of the abstract data types (ADTs) discussed in class so far, such as List ADT, without implementing the details. You may also write your function in pseudocode without focusing on syntactic details (e.g., you may write "$for\ i\ =\ 0\ to\ (n-1)$"), but you need to specify each step and each function call unambiguously. To access string elements, you may use the $str[index]$ operator. To access string size, you may use $str.size()$.

For full marks, the run-time performance of your algorithm needs to be $O(n)$ where n is the size of the garbled text, and you must use recursion to solve this problem. For partial marks, the run-time performance of your algorithm can be worse than linear (e.g., $O(n^2)$), and you can use iteration to solve this problem. [The amount of partial marks awarded will be decided when grading.]

Solution:

```
// PURPOSE: Recursively checks if given message is contained within text
//    INPUTS: message - message string to be found
//            text - garbled text within which the message may be contained
//            size1 - current size of the message string
//            size2 - current size of the text string
//            distance - current distance between the message and text strings
//    RETURNS: true, if the message is found within the text string; false, otherwise
bool has_original_string(string message, string text,
                         int size1, int size2, int& distance)  {
    // base case: down to no characters in both strings; return true [1 mark]
    if (size1 == 0 && size2 == 0) return true;

    // base case: down to no characters in garbled text; return false [1 mark]
    if (size2 == 0) return false;

    // general case: last characters match in both strings [3 marks]
    if (size1 > 0 && message[size1 - 1] == text[size2 - 1])
        return has_original_string(message, text, size1 - 1, size2 - 1, distance);

    // general case: last characters do not match [3 marks]
    else
        return has_original_string(message, text, size1, size2 - 1, ++distance);
}

// PURPOSE: Checks if message is contained within garbled text
//    INPUTS: message - message string to be found
//            garbled_text - text within which the message may be contained
//    RETURNS: if the message is found within the garbled_text string,
//            it returns the distance between the message and garbled_text strings
//            else, it returns -1
int has_original_string(string message, string garbled_text) {
    // intialize variables
    int distance = 0;
    bool is_subsequence = true;

    // make the recursive call [1 mark]
    is_subsequence = has_original_string(message, garbled_text,
                        message.size(), garbled_text.size(), distance);

    // return the corresponding distance value [1 mark]
    return (is_subsequence ? distance : -1);
}
```

Grading scheme: See above. Award up to 5 marks for the solutions that are worse than O(n) or that are not recursive. Subtract marks for solutions that are very hard to read or understand, or that do not provide all the required code steps.

```
int main() {
    string message("run");
    string garbled_text("erfuinnns");
    cout << "Testing run & erfuinnns: ";
    cout << has_original_string(message, garbled_text) << endl;
    // OUTPUT: 6

    message = "bob";
    garbled_text = "bcccddoffbbbb";
    cout << "Testing bob & bcccddoffbbbb: ";
    cout << has_original_string(message, garbled_text) << endl;
    // OUTPUT: 10

    message = "";
    garbled_text = "";
    cout << "Testing  & : ";
    cout << has_original_string(message, garbled_text) << endl;
    // OUTPUT: 0

    message = "let";
    garbled_text = "tellbbg";
    cout << "Testing let & tellbbg: ";
    cout << has_original_string(message, garbled_text) << endl;
    // OUTPUT: -1

    message = "clue";
    garbled_text = "cluuu";
    cout << "Testing clue & cluuu: ";
    cout << has_original_string(message, garbled_text) << endl;
    // OUTPUT: -1

    message = "a";
    garbled_text = "bb";
    cout << "Testing a & bb: ";
    cout << has_original_string(message, garbled_text) << endl;
    // OUTPUT: -1

    return 0;
}
```

b. (5 marks)

Once you have designed the algorithm, explain in natural language — not code — how would you comprehensively test the algorithm. That is, list ten or more test cases that you would use to ensure correctness of your algorithm and enhance its robustness. Group the test cases into categories and provide short description for each category.

For example, one of the test case given above could be explained as "test a scenario where the inverted message is contained in the garbled text." You do not have to modify the code to pass the test cases.

Solution:

```
[Test for cases where message or garbled_text is empty; 1 mark]
1. message is empty, garbled_text is empty
2. message is empty, garbled_text is not empty
3. message is not empty, garbled_text is empty

[Test for cases where characters other than lower-case letters are present; 0.5 mark]
4. message contains characters other than lower-case letters, garbled_text does not
5. garbled_text contains characters other than lower-case letters, message does not
6. both inputs contain characters other than lower-case letters

[Test for cases where message and garbled_text are of different relative sizes; 0.5 mark]
7. message is smaller than garbled_text, neither is empty
8. garbled_text is smaller than message, neither is empty
9. message and garbled_text are of equal size, neither is empty

[Test for cases where message is contained in garbled text; no interleaving; 0.5 mark]
(e.g., let and letting, let and aletting, let and alet)
10. message is contained at the start of garbled text
11. message is contained at the end of garbled text
12. message is contained in garbled text but not at the start or at the end

[Test for cases where message is contained in garbled text; some interleaving; 0.5 mark]
(e.g., let and lebtting, let and albetting, let and alebt)
13. message is contained at the start of garbled text with some character interleaving
14. message is contained at the end of garbled text with some character interleaving
15. message is contained midway in garbled text  with some character interleaving

[Test for cases where message is contained in garbled text; all interleaving; 0.5 mark]
(e.g., let and lbbectting, let and albedddtting, let and aldddebt)
16. message is contained at the start of garbled text with some character interleaving
17. message is contained at the end of garbled text with some character interleaving
18. message is contained midway in garbled text with some character interleaving

[Test for cases where message is repeated in garbled text; 0.5 mark]
19. message is repeated twice in garbled text
20. message is repeated five times in garbled text
21. message is repeated ten times in garbled text

[Test for cases where inverted message is present in garbled text; 0.5 mark]
22. inverted message is contained at the start of garbled text
23. inverted message is contained at the end of garbled text
```

```
24. inverted message is contained midway in garbled text
25. inverted message is repeated and contained in garbled text

[Test for cases where message is not present in garbled text; 0.5 mark]
26. no characters from message are present in garbled text
27. some characters from message are present in garbled text, order is preserved
28. some characters from message are present in garbled text, order not preserved
29. all but one character from message are present in garbled text, order is preserved
30. all but one character from message are present in garbled text, order not preserved
```

Grading scheme: If only the description or one test case with description are provided, award the marks as specified above. If the description + two or more test cases are provided, award 1 mark for that category. Other categories may be accepted as appropriate. To qualify for full marks, at least ten distinct cases need to be provided with appropriate descriptions.

3 Question 3. Divide and Conquer Algorithms (10 marks)

(approx. 20 minutes)
Consider the following function:

```
void woof_star(vector<int>& data, int first, int last) {
    cout << "woof_star called with [" << first << ", " << last << "]" << endl;

    int partition_size = (last - first) / 5;

    if (last <= first + 1) {
        ++data[last];
        cout << "YAS!" << endl;
        return;
    }

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

    woof_star(data, first + 1, first + partition_size);
    woof_star(data, first + partition_size * 4 + 1, last);
}
```

a. (3 marks)
Draw the call tree for this function when $woof\_star(data, 17, 31)$ is called.

```
                        (data, 17, 31)
      (data, 18, 19) [1 mark]           (data, 26, 31) [1 mark]
                        (data, 27, 27) [0.5 mark]     (data, 31, 31) [0.5 mark]
```

Grading scheme: See above.

b. (7 marks)

Let $n = last - first$ for $n \geq 0$ and let the recurrence relationship for this function be defined as: $T(1) = a, T(n) = 2T(n/5) + n/5 + b \; for \; n > 1$.

Solve the recurrence relationship 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 may apply the heuristics once you have fully simplified the expression. Show all steps in deriving your solution.

Solution:

$$\begin{aligned}
T(n) = 2\ T(\frac{n}{5}) + \frac{n}{5} + b = \\
2\ 2\ T(\frac{n}{5^2}) + 2\frac{n}{5^2} + \frac{n}{5} + 2b + b = \\
2\ 2\ 2\ T(\frac{n}{5^3}) + 2^2\frac{n}{5^3} + 2\frac{n}{5^2} + \frac{n}{5} + 4b + 2b + b = \\
... = 2^i\ T(\frac{n}{5^i}) + \frac{n}{5}\sum_{j=0}^{i-1}(\frac{2}{5})^j + b\sum_{j=0}^{i-1}2^j = \\
2^i\ T(\frac{n}{5^i}) + \frac{n}{5}\frac{(\frac{2}{5})^i - 1}{\frac{2}{5} - 1} + b(2^i - 1) = \\
2^i\ T(\frac{n}{5^i}) + \frac{n}{3}(1 - (\frac{2}{5})^i) + b(2^i - 1)
\end{aligned} \tag{11}$$

When $\frac{n}{5^i} = 1$, let $i = c$. It follows that $\frac{n}{5^c} = 1$ and $n = 5^c$, so $c = \log_5(n)$.

From there,

$$\begin{aligned}
T(n) = 2^i\ T(\frac{n}{5^i}) + \frac{n}{3}(1 - (\frac{2}{5})^i) + b(2^i - 1) = \\
2^{\log_5(n)}\ T(1) + \frac{n}{3}(1 - (\frac{2}{5})^{\log_5(n)}) + b(2^{\log_5(n)} - 1) = \\
n^{\log_5(2)}\ a + \frac{n}{3}(1 - (\frac{2^{\log_5(n)}}{5^{\log_5(n)}})) + b(n^{\log_5(2)} - 1) = \\
n^{\log_5(2)}\ a + \frac{n}{3}(1 - (\frac{n^{\log_5(2)}}{n})) + b(n^{\log_5(2)} - 1) = \\
(a + b - \frac{1}{3})n^{\log_5(2)} + \frac{n}{3} - b = \\
O(n).
\end{aligned} \tag{12}$$

Grading scheme: 1 mark for the initial steps, 2 marks for derivation and simplification of the general form, 1 mark for the computation of $c = \log_5(n)$, 2 marks for the application of $c = \log_5(n)$ and full simplification, and 1 mark for the derivation of $O(n)$.

4 Question 4. Data Structure Design (15 marks)

(approx. 40 minutes)
We are designing a mobile app for a service similar to Netflix that will offer a subscription-based access to training content authored by our company. The app will allow users to create a list of their favourite viewing items that are offered by the service. For each favourite item, the app will store the unique item ID as an unsigned integer, item name as a string, item rating as an integer that represents the number of stars (from one star to five stars in one-star increments), item category as a string (e.g., data analytics, healthcare), and user-defined notes as a string.

a. (4 marks) As the reference model for our design, we will make use of the **doubly linked list from Lab Assignment #2** where *FavItem* objects will be managed by the *FavsManager*. *FavsManager* needs to include **pointers to first and last elements**. To that end, complete the C++ class declaration provided below to make *FavItem* and *FavsManager* function. For *FavItem*, include the empty constructor, parametric constructor, and *operator* ==. For *FavsManager*, include the empty constructor, *void push_back*(*FavItem*& *new_item*), and *void pop_nth_from_end*(*unsigned int index*). Provide only method signatures at this point and no method implementation. Do not forget to include the *size* variable.
Solution:

```
class FavItem {
    unsigned int itemID;
    friend class FavsManager;

    // fill in other required member attributes and methods below
    // members should be declared as public or private as appropriate
    string item_name, item_category, user_notes; // [0.5 mark] for attributes
    unsigned int rating;
    FavItem* prev; // [0.5 mark]
    FavItem* next; // [0.5 mark]

public:
    FavItem(); // [0.5 mark] for the constructors and operator==
    FavItem(unsigned int new_itemID);
    FavItem(unsigned int new_itemID, string new_item_name, unsigned int new_rating,
            string new_item_category, string new_user_notes);
    bool operator==(const FavItem& rhs) const;
};

class FavsManager {
// fill in other required member attributes and methods below
// members should be declared as public or private as appropriate
    FavItem* first; // [0.5 mark]
    FavItem* last; // [0.5 mark]
    int size; // [0.5 mark]
public:
    FavsManager(); // [0.5 mark] for service methods
    int get_size() const;
    void print();
    void push_back(FavItem& new_item);
    void pop_nth_from_end(unsigned int index);
};
```

b. (5 marks) Implement the method *void push_back(FavItem& new_item)* that will insert the referenced *FavItem* element at the end of the list using dynamic memory, and update the *size* variable. You must use exact C++ syntax for this function. **Solution:**

```
void FavsManager::push_back(FavItem& new_item) {
    // allocate new element on the heap [0.5 mark]
    FavItem* new_element = new FavItem(new_item);

    // base case: empty list [1 mark]
    if (!first) {
        first = last = new_element;

    // general case: non-empty list
    } else {
        FavItem* cur = first;
        while (cur) { // check for duplicates
            if (cur->itemID == new_item.itemID)
                return;
            else
                cur = cur->next;
        }

        // update relevant pointers
        last->next = new_element; // [1 mark]
        new_element->prev = last; // [1 mark]

        // update last element
        last = new_element; // [1 mark]
    }

    // update size [0.5 mark]
    ++size;
}
```

Grading scheme: See above.

c. (6 marks) Implement the method *void pop_nth_from_end(unsigned int index)*. The method will take as input an *index* value, remove the index-th element from the end of the list, free up its memory, and update the *size* variable.

For example, if the list stores the following item IDs: 141 → 123 → 15 and if *index* = 2, then the function will remove the second item from the end and produce the following list: 141 → 15. If the *index* = 0 or if *index* exceeds list size, then no change will be made.

You must use exact C++ syntax for this function. 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. You also need to document your code and explain the rationale for key steps. **Solution:**

```
void FavsManager::pop_nth_from_end(unsigned int index) {
    // check for invalid index [0.5 mark]
    if (index == 0 || index > size)
        return;
    // base case: index == 1 && size == 1 [1 mark]
    if (index == 1 && size == 1) {
        delete last;
        first = last = NULL;
    // base case: index == 1 && size > 1 [1 mark]
    } else if (index == 1) {
        FavItem* cur = last;
        last->prev->next = NULL;
        last = last->prev;
        delete cur;
    // base case: index == size && size > 1 [1 mark]
    } else if (index == size) {
        FavItem* cur = first;
        first->next->prev = NULL;
        first = first->next;
        delete cur;
    // general case: index < size && size > 1
    } else {
        unsigned int count = 1;
        FavItem* cur = last;
        // find the corresponding index [1 mark]
        while (count < index && cur) {
            cur = cur->prev;
            ++count;
        }
        // update the prev and next pointers [1 mark]
        cur->prev->next = cur->next;
        cur->next->prev = cur->prev;
        delete cur;
    }
    // update size [0.5 mark]
    --size;
}
```

Grading scheme: See above.

```
// additional code for testing; otherwise, not needed
FavItem::FavItem(unsigned int new_itemID) :
                    itemID(new_itemID), rating(0), prev(0), next(0) {};
FavsManager::FavsManager() : first(NULL), last(NULL), size(0) {}

void FavsManager::print() {
    FavItem* cur = first;
    while (cur) {
        cout << cur->itemID << " -> ";
        cur = cur->next;
    }
    cout << endl;

    cout << "Size: "  << size << endl;
}

int main() {
    FavsManager fm;
    FavItem test1(141), test2(123), test3(15);
    FavItem test4(55), test5(22), test6(125);

    fm.push_back(test1);
    fm.push_back(test2);
    fm.push_back(test3);
    fm.push_back(test3);
    fm.push_back(test4);
    fm.push_back(test5);
    fm.push_back(test6);
    fm.push_back(test4);
    fm.print();

    fm.pop_nth_from_end(6);
    fm.pop_nth_from_end(1);
    fm.pop_nth_from_end(1);
    fm.pop_nth_from_end(3);
    fm.pop_nth_from_end(2);
    fm.pop_nth_from_end(1);
    fm.print();

    return 0;
}
```