York University

AP/ITEC 2620 3.0 – Section M
INTRODUCTION TO DATA STRUCTURES
Winter 2016

Midterm Test

Examiner: S. Chen
Duration: One Hour and 30 Minutes

This exam is closed textbook(s) and closed notes. Use of any electronic device (e.g. for computing and/or communicating) is NOT permitted.

Do not unstaple this test book – any detached sheets will be discarded. Answer all questions in the space provided. No additional sheets are permitted.

Work independently. The value of each part of each question is indicated. The total value of all questions is 60. However, the test is out of 50, so there are 10 extra marks available – it is possible to score more than 100% on this test.

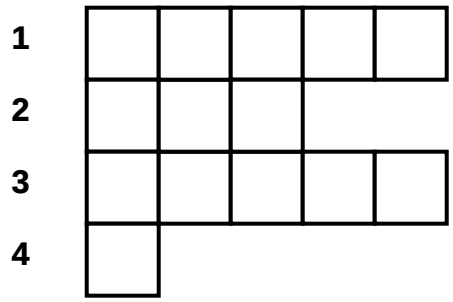
Write your name and student number in the space below. Do the same on the top of each sheet of this exam where indicated.

NOTE: YOU MAY USE PEN OR PENCIL, BUT ANSWERS IN PENCIL WILL NOT BE CONSIDERED FOR REGRADING.

Surname: _____

Given Names: _____

Student Number: _____



Total

Surname:_____ First name:_____ Student #: _____

Question 1 (15 marks) Short Answer (<u>maximum 20 words</u>):
Answer all five parts below.

Part A (**3 marks**): What is the average case time complexity for linear search on a sorted array? Explain (and/or draw a diagram).

Part B (**3 marks**): Assuming that each new element/node must be added starting from the head, what is the average case time complexity to add n values to a linked list that that is initially empty and that will have its values sorted from smallest to largest. Explain (and/or draw a diagram).

Part C (**3 marks**): What is the best case time complexity to delete a value from a full BST? Explain (and/or draw a diagram).

Part D (**3 marks**): An **O**(n$\log$n) algorithm (e.g. mergesort) will always run faster than an **O**(n^2) algorithm (e.g. selection sort) for all values of n. True or False? Explain.

Part E (**3 marks**): An implementation of quicksort has its worst case of **O**(n^2) for an inverse sorted array (e.g. from largest to smallest). What case will it be for an already sorted array (e.g. from smallest to largest)? Explain.

Surname:_____ First name:_____ Student #: _____

Question 2 (15 marks) Complexity Analysis/Estimation:
Answer all three parts below.

Part A (**5 marks**):
Develop a recurrence relationship for the `silly` method that is called as follows:

```
int result = silly (n);

public static int silly (int n)
{
   if (n <= 1)
   {
     return -100;
   }

   int sum = 0;
   for (int i = 0; i < n; i++)
   {
     sum += i;
   }

   return sum + silly (n-2);
}
```

Surname:_____ First name:_____ Student #: _____

Part B (**5 marks**):
What is the complexity of the following recurrence relation? Show the details of your analysis.

```
T(1) = 1
T(n) = T(n-1) * n
```

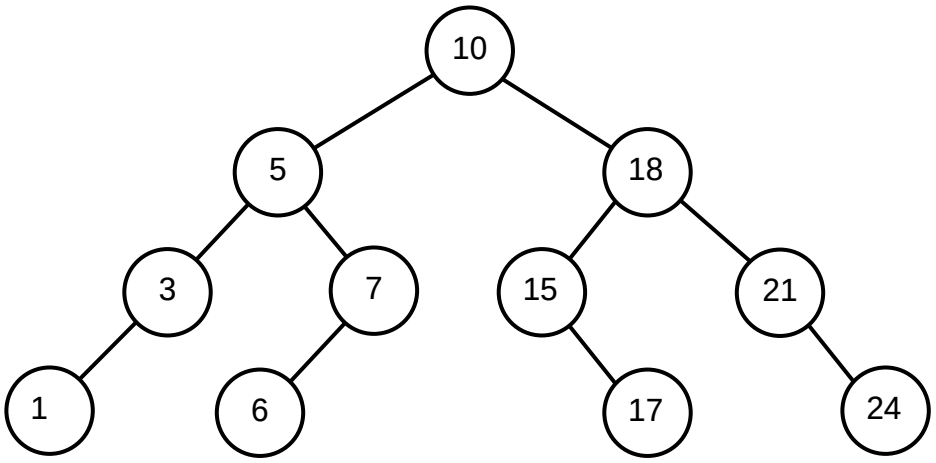
Part C (**5 marks**):
What is the complexity of the given method as a function `n`? Show all details of your analysis.

```
public static int sumIt (int n)
{
   int sum = 0;

   for (i=1; i<=n; i++)
     for (j=1; j<=n; j*=2)
       sum++;

   return sum;
}
```

Surname:_____ First name:_____ Student #: _____

Question 3 (15 marks) Linked Data Structures:
Answer all five parts below.

Tree ***T*** shown below is a BST.



Part A (**2 marks**): Starting with the original tree ***T***, insert **14**. Draw the resulting tree below.

Part B (**2 marks**): Starting with the original tree ***T***, delete **7**. Draw the resulting tree below.

Surname:_____ First name:_____ Student #: _____

Part C (**2 marks**): Starting with the tree developed in Part B, insert **7**. Draw the resulting tree below.

Part D (**4 marks**): Starting with the original tree ***T***, delete **5**. Use the deletion strategy presented in class. Draw the resulting tree below.

Part E (**5 marks**): Draw the BST that is created by inserting the following nodes/values in the given order into a BST that is initially empty. 5 3 1 8 7 4 2 6

Question 4 (15 marks) Recursion:

Write a ***recursive function*** that will find the largest triangular number less than or equal to the given target.

The triangular numbers are as follows:

```
 1   =  1
 3   =  1 + 2
 6   =  1 + 2 + 3
10   =  1 + 2 + 3 + 4
15   =  1 + 2 + 3 + 4 + 5
21   =  1 + 2 + 3 + 4 + 5 + 6
etc.
```

The series begins with 1 (the first triangular number). To calculate the n^{th} triangular number, n is added to the previous triangular number. For example, the fourth triangular number is calculated by adding 4 to the third triangular number (which is 6), i.e. 10 = (1 + 2 + 3) + 4.

For example, the following statements would lead to the underlined output:

Example 1:

```
System.out.println( getLargestTN( 1, 0, 1 ));

1               // 1 is the largest triangular number that is less than or equal to 1
```

Example 2:

```
System.out.println( getLargestTN( 1, 0, 7 ));

6               // 6 is the largest triangular number that is less than or equal to 7
```

Example 3:

```
System.out.println( getLargestTN( 1, 0, 25 ));

21              // 21 is the largest triangular number that is less than or equal to 25
```

Please write your function on the following page.
You may use this page for rough work, but anything on this page will not be graded.

Surname:_____ First name:_____ Student #: _____

```
public static int getLargestTN (int n, int sum, int target)
{









}
```

This blank page may be detached and used for rough work.