UNB
EST. 1785
UNIVERSITY OF NEW BRUNSWICK

Example Exam Questions

CS1083 - Introduction to Computer Programming II (in Java)

Andrew McAllister, Ph.D.

The purpose of this document is to help you prepare to write the final exam in this course.

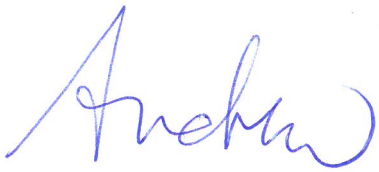
On the following pages you will find a number of example exam questions. The best use of this resource is to answer the example questions as if you were writing a final exam. Do not consult other resources like your course notes or the textbook. Do not Google for help. Do not attempt to compile and run the programs you write. Remember – on the final exam you get graded for the *first* attempt you make at each question, so do the same when you're practicing with these example questions.

There may be more room provided than is required for some answers.

The second half of the document provides sample answers to the example questions. You will get the most value from this document if you can resist the temptation to look at these answers until you have given your best effort to answer the example questions.

These example questions have been taken from actual exams from previous offerings of the course ... and they've been updated slightly to be consistent with the way material is presented in this open access version of the course.

Good luck with your preparation and with the final exam!



Question 1: Recursion with Strings

Write a recursive method called **removeSpaces** that has the following first line:

```
public static String removeSpaces(String s, int index)
```

This method processes the characters in `s` starting from the position given by `index` onwards until the end of `s`. This method returns a string with those characters but without any of the spaces.

For example, if `s` has the value `"abc x y z"`, then calling `removeSpaces` with index of 0 will return `"abcxyz"`.

The only methods you are permitted to use from the String class are charAt() and length(). (Happily, these are the only ones you need.)

Question 2: A mystery() Method

Examine the following Java code:

```
public class Q3
{
   public static void main(String[] args)
   {  int[] array = {3, 5, 1, 2};
      mystery(array, 5);
   }

   public static void mystery(int[] array, int level)
   {  if (level == 0)
         return;

      System.out.print("|");

      for (int i=0; i<array.length; i++)
      {  if (array[i] >= level)
            System.out.print("*");
         else
            System.out.print(" ");
      }

      System.out.println("|");

      mystery(array, level-1);
   }
}
```

What does the above program print?

Question 3: File I/O and Exceptions

Assume you have a text file named “infile.txt” with contents in the following format. You can assume there is one name and one test score on each line.

infile.txt

```
Lynn   8.5
Chris  7.8
John   7.5
Lisa   9.2
Sean   8.9
```

We want a program that will read in these values and produce a new file named “outfile.txt” that indicates the name associated with the highest test score. The contents of the new file should look like this:

outfile.txt

```
Has highest score:  Lisa
```

Do not use `throws IOException` on the first line of the main method. Instead, include all file processing code within an appropriate try-catch block.

Question 4: Time Complexity

Assume you wrote a program that processes an array of N objects. You timed the execution of the program with different values of N (that is, with different numbers of objects in the array). The following are your results – the number of milliseconds it took for the program to run with different array sizes:

N	Milliseconds
1000	25
2000	50
3000	75
4000	100

Based on these results, which of the following algorithm complexities most accurately describes the behaviour of your program?

a) O(N)

b) $O(N^2)$

c) $O(\log_2 N)$

d) $O(N \log_2 N)$

Assume you then ran a similar experiment with a different program and produced the following results:

N	Milliseconds
1000	25
2000	100
3000	225
4000	400

Based on these results, which of the following algorithm complexities most accurately describes the behaviour of this second program?

a) O(N)

b) $O(N^2)$

c) $O(\log_2 N)$

d) $O(N \log_2 N)$

Question 5: Linked Lists

Consider these two Java classes:

```
/** A singly-linked list class. Note there are no 'previous' or
    'size' instance variables. They are not needed.
*/
public class LinkedList
{  private IntNode head;

   // Constructor
   public LinkedList()
   {  head = null;
   }

   public IntNode getHead()
   {  return head;
   }

   //*** The method you write for this question
   //*** will be inserted here

} // end LinkedList class


//********* Beginning of a separate class ********
public class IntNode
{  private int val;
   private IntNode next;

   // Constructor
   public IntNode(int valIn)
   {  val  = valIn;
      next = null;
   }

   public IntNode getNext()
   {  return next;
   }

   public void setNext(IntNode nextIn)
   {  next = nextIn;
   }
} // end IntNode class
```

(This question is continued on the next page)

Write a method called **size()**, to be included in the `LinkedList` class shown on the previous page. Your method returns the size of the linked list (the number of nodes contained in the list). Notice the `LinkedList` class does NOT include a `size` instance variable, so your `size()` method must loop to visit each node in the list, counting them as it goes. Your method begins as follows:

```
public int size()
```

Sample Solutions

The following pages provide sample solutions for the example questions.

If you haven't yet attempted the questions, it is highly recommended that you do so before checking out the sample solutions.

Sample solution for **Question 1: Recursion with Strings**

This question tests whether you can devise a recursive algorithm that moves an index through the positions in a string, plus whether you can turn that into Java code.

Note – this is similar to how a recursive method moves an index through the elements of an array, such as with the recursive algorithm for summing integers discussed in Module 7.

Here is a sample solution. The basic idea is to return the current character concatenated with whatever is returned from a recursive call for the rest of the string. Except if the current character is a blank, then just return what comes back from the recursive call.

```
public static String removeSpaces(String s, int index)
{  char c = s.charAt(index);
   String r = c+"";
   if(c == ' ')
      r = "";
   if (index == s.length()-1)
      return r;
   return r + removeSpaces(s, index+1);
}
```

Solution for **Question 2: A mystery() Method**

Each call to the recursive mystery() method prints one line in the following output. The for loop is used to go through the numbers in the array, and for each number decide if this number is as big as the current level. If so, include an asterisk in this position. The result is a simulated bar chart, like this:

```
| *  |
| *  |
|**  |
|** *|
|****|
```

The array contains the numbers 3, 5, 1, 2. Notice those are the heights of the four 'bars' in the output.

Sample solution for **Question 3: File I/O and Exceptions**

```
import java.io.*;
import java.util.Scanner;

public class ExampleQuestion3
{  public static void main(String[] args)
   {  Scanner     fileReader = null;
      PrintWriter fileWriter = null;
      String name;
      String highestName  = null;
      double score;
      double highestScore = -1.0;
      try
      {  fileReader = new Scanner(new File("infile.txt"));
         fileWriter = new PrintWriter("outfile.txt");
         while (fileReader.hasNext())
         {  name  = fileReader.next();
            score = fileReader.nextDouble();
            if (score > highestScore)
            {  highestScore = score;
               highestName  = name;
            }
         }
         fileWriter.println("Has highest score: " + highestName);
      }
      catch (IOException e)
      {  System.out.println("File processing error.");
      }
      finally
      {  try
         {  System.out.println("Closing output file... ");
            fileWriter.close();
         }
         catch (Exception e)
         {  System.out.println("Error closing file.");
         }
      }
      System.out.println("The end!");
   } // end main
} // end class
```

Solution for **Question 4: Time Complexity**

Assume you wrote a program that processes an array of N objects. You timed the execution of the program with different values of N (that is, with different numbers of objects in the array). The following are your results – the number of milliseconds it took for the program to run with different array sizes:

N	Milliseconds
1000	25
2000	50
3000	75
4000	100

Based on these results, which of the following algorithm complexities most accurately describes the behaviour of your program?

a) O(N) (Notice the times go up in a linear fashion – 25 ms per 1000 objects)

b) $O(N^2)$

c) $O(\log_2 N)$

d) $O(N \log_2 N)$

Assume you then ran a similar experiment with a different program and produced the following results:

N	Milliseconds
1000	25
2000	100
3000	225
4000	400

Based on these results, which of the following algorithm complexities most accurately describes the behaviour of this second program?

a) O(N)

b) $O(N^2)$ (Doubling the amount of data increases the time by a factor of 4, or 2^2)

c) $O(\log_2 N)$

d) $O(N \log_2 N)$

Sample solution for **Question 5: Linked Lists**

```
public int size()
{  int result = 0;
   IntNode current = head;

   while(current != null)
   {  result = result + 1;
      current = current.getNext();
   }

   return result;
}
```