Laboratory 2 – Week 2
Design and Implementation of Simple Algorithms

2.1 Introduction

This laboratory involves the creation of a number of Java programs that perform simple tasks. Some of these tasks are numerical in nature.

Note that next week we will be looking into the mathematical foundation for this module in more detail.

Make sure that you save any code you write. Also make sure you save any plots (perhaps using Excel) of run times and performance that you measure. Supplemental material that might be useful for these exercises can be found in Section 2.5.

2.2 Exercise 1: Array Maximum

The first exercise is to create and test the `ArrayMax` algorithm that we discussed in the lectures.

1) Create a project called `Lab2` and then add a class called `ArrayMaxExercise`. Create a method called `ArrayMax`. This method should take as parameter a `double` array and return the maximum in the array as a `double`.
2) Implement the `ArrayMax` algorithm according to the Pseudo-Code.
3) Test the `ArrayMax` algorithm by creating a `main` method, creating an array of a given size and then adding some test data to it. Call the `ArrayMax` algorithm and check that results are as expected.
4) How would you modify the Pseudo-Code and Java code of the algorithm so that it now computes the minimum value?

2.3 Exercise 2: Linear Summation

Details of simple summations and their properties can be found on http://mathworld.wolfram.com/ArithmeticSeries.html.

5) Implement an algorithm `LinearSum(N)` in Java by using a loop to calculate the following linear summation. Test your answer by computing the expected result from the web resource given above.

$$\sum_{i=1}^{N} i = 1 + 2 + 3 + ... + (N-1) + N$$

6) Compare the performance of the algorithm for different values of N in terms of their running time, i.e. the number of executions (steps) it takes for the algorithm to complete the calculation.
7) Build in an automatic testing function that checks that the result is what is expected.

2.4 Exercise 3. Geometric Summation

Details of geometric summations (of series) and their properties can be found on http://mathworld.wolfram.com/GeometricSeries.html.

8) Design an algorithm `GeoSum(a,N)` in Pseudo-Code for calculating the geometric summation:

$$\sum_{i=0}^{N} a^i = 1 + a + a^2 + ... + a^N$$

9) Implement the algorithm in JAVA using a loop. Test the results against the expected answer and then record the following:
    - The source code you produced.
    - Example results/plots generated from your test runs for different values of *a* and *N*.
    - Actual measurements of the running time. Which of the two parameters has the most effect on how long the program runs for?

2.5 Supplemental Material

The following material might be of use for some of the exercises in this worksheet.

2.5.1 Pseudo-Code Etc... for ArrayMax

```
Algorithm 1. ArrayMax(Arr)
Input: A 1-D numerical array Arr of size n
1) Let CurrentMax = $a_0$
2) For i = 1 to n-1
3)   If $a_i$ > CurrentMax Then CurrentMax = $a_i$
4) End For
Output: CurrentMax, the largest value in Arr
```

To create a `double` array of size 10 you could use:

```
double arr[] = new double[10];
```

To find out the size of an array (called `arr`) you could use:

```
int n = arr.length;
```

You can alternatively use the `ArrayList` class. Remember to import it if you do:

```
import java.util.ArrayList;
```

2.5.2 Code Fragment

Here is a sample program fragment to calculate $\sum_{i=1}^{N} i$:

```java
public static int sum(int n)
{
	int partialSum;
	partialSum =0;
	for(int i=1;i<=n;i++)
	{
	  partialSum += i;
	}
	return(partialSum);
}
```

2.5.3 Running Time Measurement in JAVA

The following methods might be useful for measuring running time.

```java
public static void RunAlgorithm()
{
	long StartTime, EndTime, ElapsedTime;

	System.out.println("Testing algorithm …");

	// Save the time before the algorithm run
	StartTime=System.currentTimeMillis();

	// Run the algorithm
	DummyAlgorithm();

	// Save the time after the run
	EndTime=System.currentTimeMillis();

	// Calculate the difference
	ElapsedTime= EndTime- StartTime;

	// Print it out
	System.out.println("The algorithm took " + ElapsedTime + " milliseconds to 
run.");
}
private static void DummyAlgorithm()
{
	// This is a dummy function and should be replaced with your
	// own implementation of the actual algorithm
	// The following is an example algorithm and took 112 ms to run on my
	// computer........
	for(int i = 0;i<1000000;++i)
	{
		double x = Math.sin((double)(i)/1000000.0);
	}
}
```

2.5.4 More Precise Running Time Measurement in JAVA

If the method `System.currentTimeMillis()` always returns 0 for your algorithm with any size of input, you might like to use a more precise time unit. Try the method, `System.nanoTime()`, which returns the current value of the most precise available system timer, in nanoseconds.