Laboratory 4 – Week 4

**Analysis of Algorithms**

## 4.1 Introduction

**Firstly, the content of this worksheet is (*may be*) part of the Task #1 and Task #2 assessments.**

This worksheet requires you to analyse two algorithms, work out the primitive operation count for both algorithms, derive the Big-Oh computational complexity for both algorithms, implement and then test them both.

## 4.2 Preliminaries

It might be worth creating a sub-directory where you store your CS2004 Java programs. Within these laboratories we will be creating a number of programs, you may edit existing programs or create new ones as you see fit, however it may be better if you create new programs so that you have all of the programs you have created for revision and reference. If you are working from a university machine, make sure you have access to your home directory (we will assume this is the H: drive).

## 4.3 Prefix Averages

This week’s computational problem is the calculation of *prefix averages*: Given an array *X* storing *n* numbers, we want to compute an array *A* such that *A*[*i*] is the average of elements *X*[0], *X*[1], … , *X*[i], for *i*=0,1, …, *n*-1.

For example imagine that we have a ten element array *X* = [7, 3, -1, 2, 9 ,0, 0.8, 52, 2.2, 900].

Then the array *A* would be computed as follows according to Table 4.1:

|  |  |  |
| --- | --- | --- |
| **Array *A*** | **Contents of *A*[*i*]** | **Value** |
| *A*[0] | (*X*[0])/1 | 7.000 |
| *A*[1] | (*X*[0]+*X*[1])/2 | 5.000 |
| *A*[2] | (*X*[0]+*X*[1]+*X*[2])/3 | 3.000 |
| *A*[3] | (*X*[0]+*X*[1]+*X*[2]+*X*[3])/4 | 2.750 |
| *A*[4] | (*X*[0]+*X*[1]+*X*[2]+*X*[3]+*X*[4])/5 | 4.000 |
| *A*[5] | (*X*[0]+*X*[1]+*X*[2]+*X*[3]+*X*[4]+*X*[5])/6 | 3.333 |
| *A*[6] | (*X*[0]+*X*[1]+*X*[2]+*X*[3]+*X*[4]+*X*[5]+*X*[6])/7 | 2.971 |
| *A*[7] | (*X*[0]+*X*[1]+*X*[2]+*X*[3]+*X*[4]+*X*[5]+*X*[6]+*X*[7])/8 | 9.100 |
| *A*[8] | (*X*[0]+*X*[1]+*X*[2]+*X*[3]+*X*[4]+*X*[5]+*X*[6]+*X*[7]+*X*[8])/9 | 8.333 |
| *A*[9] | (*X*[0]+*X*[1]+*X*[2]+*X*[3]+*X*[4]+*X*[5]+*X*[6]+*X*[7]+*X*[8]*+X*[9])/10 | 97.500 |
| Table 4.1: Prefix Average Example | | |

Computing prefix averages has many applications in finance, economics and statistics. For example in the stock market, given the year-on-year variation of a company’s stock price, one can use prefix averages to see the stock’s average variation in stock price over the last year, the last three years, the last ten years, etc.

Let us analyse two algorithms that solve this problem:

Algorithm 4.1. PrefixAverages1(X)

Input: X, a 1-D numerical array of size n

1) Let A = an empty 1-D numerical array of size n

2) For i = 0 to n-1

3) Let s = X[0]

4) For j = 1 to n-1

5) If j ≤ i Then

6) Let s = s + X[j]

7) End If

8) End For

9) Let A[i] = s /(i+1)

10) End For

Output: An n-element array A of numbers such that A[i]

is the average of elements X[0],X[1], … ,X[i]

Algorithm 4.2. PrefixAverages2(X)

Input: X, a 1-D numerical array of size n

1) Let A = an empty 1-D numerical array of size n

2) Let s = 0

3) For i = 0 to n-1

4) Let s = s + X[i]

5) Let A[i] = s / (i+1)

6) End For

Output: An n-element array A of numbers such that A[i]

is the average of elements X[0],X[1], … ,X[i]

## 4.4 Exercise 1: Computational Complexity

1. What is the difference between the two algorithms? Read these two algorithms carefully. If necessary, it might be helpful to set up an array yourself and apply these two algorithms on it by running through them by hand for small size values of *n*.
2. Analyse both algorithms by counting primitive operations and derive *T*(*n*) for both algorithms. What is the time complexity (Big-Oh, O(*n*)) of each algorithm? Which one is the most efficient?

## 4.5 Exercise 2: Implementation

Implement the two algorithms in Java and perform a thorough experimental analysis of their running times. Plot their running times as a function of their input size as scatter plots. Choose representative values of the size *n*, and run at least 5 tests for each size value *n* in your tests.

You may not be able to finish this exercise during your lab, so therefore you should complete the coding of the algorithms on your own in your private study time.