BRONX COMMUNITY COLLEGE
of the City University of New York
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

CSI33 Section E01 **Project 2**
Fall 2017 **September 11, 2017**
Due: Monday, September 25, 2016

A More Efficient Dataset Class for Test Scores

Introduction

This project is the second part of Programming Exercise 7 of Chapter 2, on page 72. You will create a folder called DatasetII which will contain the same files used to begin the previous project, `Dataset.py` and `test_Dataset.py`. You will modify `Dataset.py` to provide a different implementation of the Dataset ADT from the one you provided in project 1. Specifically, the concrete representation will not use a Python `list`, yet it will have attributes which provide enough information to give statistics on any numbers which have been added to the Dataset via the `add` method. No matter how many scores have been added to a Dataset, the size of the object remains constant, and the time to perform any of the statistical queries is constant ($\Theta(1)$) as well. A possible implementation strategy to do this is outlined below. The test program file, `test_Dataset.py`, will be exactly the same as in the first project, since the interface of the Dataset ADT which you are testing is not changing in this project—only the concrete representation.

Implementation Strategy

The idea is to provide a small set of attributes which represent the state of a Dataset, after any number of scores have been added. The values of these attributes provide enough information to calculate statistics (or simply return attribute values) for the scores added so far.

Methods such as `min` or `max` will just return values of private attributes representing the current smallest or greatest score added so far. The real work of deciding these attribute values will be done inside the `add` method. For example, when a new score is added, it will be compared with previous min and max attributes to see if these old values need to change.

The current number of scores added so far is incremented by one each time a score is added.

Similarly, the current total of scores is updated each time a new score is added.

These numbers can be used to calculate the average whenever the average is needed by the `average` method.

Another attribute will be needed, called `sum_of_squares`, to calculate the standard deviation. This will hold the value of the sum of the squares of the scores added so far. The formula for standard deviation in chapter 2, on page 59, can then be used to calculate the returned value for the `std_dev` method. (Chapter 1 uses a different formula, but it requires knowing every individual score in the dataset.