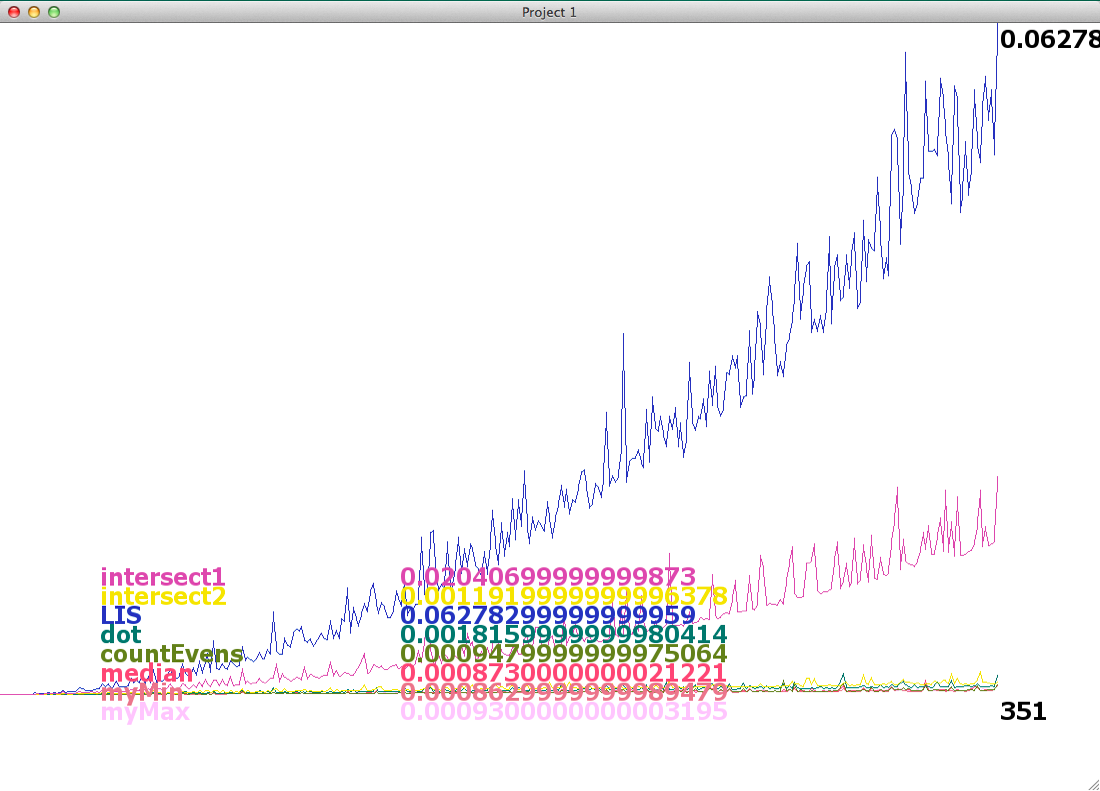
Data Structures Problem Set 1

**Due Date:** *Next* Thursday, September 11th by 11:55pm[[1]](#footnote-0)

**UPDATES:**

* **The due date has been changed!**
* With regards to submission, put everything in a folder titled *LastnameFirstname\_Pset1* and zip it (as opposed to using tar or whatever). Upload the *zipped folder*! Make sure you use your legal names.

In this assignment, you will use Pygame to graph the runtimes of some algorithms as a function of input size. The assignment has three parts. It should look like what we saw in the first lecture. The first part is to implement the algorithms that we will be running in the second part. The second and main part of this assignment is to write a function called *sim* which will do the aforementioned graphing. The third part of the assignment is written and involves (informally) arguing the runtime of the algorithms.



*Part 1/3: sim()*

*sim* will take a list of functions as an argument, run them, and graph their runtimes on increasingly large inputs. You may want to write a function that generates random inputs of a given size (such as lists) for some of the functions to take as an argument.

Example:

functionList = [myMax,

myMin,

median,

countEvens,

dot,

LIS,

intersect2,

intersect1,

fib

]

sim(functionList)

*Part 2/3: Algorithms to implement*

Below are the algorithms that you are to implement. In the cases where Python already has a function that does what is asked, write your own implementation instead. Otherwise, feel free to use any built-in functions you want.

For ease of explanation, many of the functions below are said to take a list or lists as arguments. In your implementation, it will probably be easier for them to take the size of the input and then in the body, the function will create a shuffled list of numbers of the given size.

***countEvens(A)*** *countEvens* finds the number of even numbers in a list, A.

***myMin(A)*** *myMin* finds the minimum element of a list, A.

***myMax(A)*** *myMax* finds the maximum element of a list, A.

***median(A)*** *median* finds the median of a list, A. Recall that in the case where A has an even number of elements (and thus there is no “middle” element) we average the middle two and the average is the median. For example, if A = [7, 17], median(A) = 12.

***secondBiggest(A)*** *secondBiggest* finds the second biggest element of a list. (Feel free to use Python’s max).

***dot(A, B)*** *dot* takes two lists of the same length, A and B, treats them as vectors and takes their dot product. In other words, we multiply the *i*th element of each list and add the products together. For example if A = [7, 17] and B = [2, 3], dot(A, B) = 7\*2 + 17\*3 = 65.

***LIS(A)*** *LIS* gives the length of the longest increasing sublist of a list. In other words, the longest distance between a start and end point in the list where the numbers are strictly increasing. For example, if A = [1, 1, 2, 3, 4, 5, 5], LIS(A) = 5 (starting at the second 1 and ending at the first 5). If A = [3, 2, 1], LIS(A) = 1 (every singleton sublist counts as a strictly increasing sublist).

***instersect1(A, B)*** *intersect1* takes two lists of the same length, A and B, and returns a *set* of elements that are common to both of them. Implement this without sorting the lists, using instead doubly nested for-loops.

***instersect2(A,B)*** *intersect2*does the same thing as *intersect1*. However this time sort both the arrays and return the list of common elements in linear time.

***fib(n)*** *fib(n)* returns the *n*th fibonacci number. Write a recursive implementation.

*Part 3/3: Runtimes*

For this part you have to make an argument for the runtimes of your implementations of the algorithms mentioned above. You don’t have to be too formal about it (no need for mathematical proofs). However, we do want an argument that uses on the code for evidence, as well as an argument based on the graph.

Example:

Consider the following bit of code:

def allPairsShortestPath(A):

for k in range(len(A)):

for i in range(len(A)):

for j in range(len(A)):

A[i][j] = min(A[i][j], A[i][k] + A[k][j]

An acceptable answer would be:

The j for-loop runs len(A) times and each instance of the loop takes constant time, so that loop takes len(A) time. The i for-loop runs len(A) times and each time it does does the j for-loop, so it run len(A)\*len(A) times. Following a similar argument, the k for-loop runs len(A)\*len(A)\*len(A) times. Since the function is just the triply-nested for-loop, the function is a n^3 algorithm, where n = len(A).

Its graph also looks like a cubic equation.

*Hints and Helpful functions*

***random.shuffle()***

To create the lists of length n for these functions, you can use

A = [i for i in range(n)]

random.shuffle(A)

The second line randomly orders the elements of A.

***time.clock()***

With barely any work, this function can be used to measure elapsed time. *time.clock()* takes no arguments and returns the time as a float in seconds.[[2]](#footnote-1)

***\*args***

You can pass a function as an argument! Additionally, you can pass that function’s argument as an argument. Here’s what that means:  
 def call\_the\_function(f, \*args):

return f(\*args)

Here’s some terminal output:

>>> call\_the\_function(min, [1, 2])

1

>>> call\_the\_function(min, 1, 2, 3)

As you can see, \*args can be more than a single argument.

***countEvens(n)***

Below is a summary of some of the above hints in the form of code.

def countOdds(n):

l = randomList(n)

count = 0

for i in l:

if i % 2 == 1:

count += i

return count

Note how countEvens() doesn’t actually take in a list. As discussed above, it merely takes an integer n that defines the length of the list that will be generated.

***Read documentation***

Being able to figure things out by reading documentation is a great skill to develop early on in your computer science careers. [Here](http://www.pygame.org/docs/) is the documentation for Pygame and [here](https://docs.python.org/3.4/index.html) is the documentation for Python. Being able to google an answer is also useful. If you want to know how to do something in some language, googling “[language name] [thing to be done]” often works.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Turning in your homework*

Upload your code and written work to NYU classes. Please do not upload pictures or scans of your homework. Instead, use Microsoft Word, Pages, or LaTeX and upload a PDF.

*Grading and closing remarks*

You may feel like this assignment isn’t too specific about what you need to do. That is intentional. In fact, that is how every assignment in this class will be. We don’t care how you code this assignment. As long as you do everything we ask you to, you are guaranteed a B. In fact, the app shown in the screenshot would have been given a B. There’s nothing wrong with it, but at the same time, there are plenty of things that could be better. You may choose to improve the aesthetics of the app, you may try to make it more informative (that is, how can this app be a better teaching tool?), you may try to make it easier to use, etc. The extra effort and *creativity* you put into this project translates into the extra points that push you towards an A[[3]](#footnote-2). And it is possible to get an A.

1. *What if I send it in 17 seconds after 11:55pm?!* I don’t know because I’m not completely sure what Classes is okay with. [↑](#footnote-ref-0)
2. What exactly those seconds mean is a bit complicated. Click [here](https://docs.python.org/2/library/time.html) to read the Python documentation on time.clock(). [↑](#footnote-ref-1)
3. It also translates into more Python and Pygame experience so the time investment you put into this first assignment can translate into time saved on a future assignment. [↑](#footnote-ref-2)