Assignment 1

For this assignment, you are to work in groups of 2 or 3. This assignment should provide you with experience in the analysis of algorithms.

What to include in your submission:

- **One .zip folder** *named appropriately*[1] containing:
  - A PDF containing your answers to questions #2 - #8.
  - A Python file and a C file. Make sure each program's header (block) comment includes your name.

<u>Complete the following:</u>

1. (2 marks) You are given a Python program, `Assignment1.py`, which will allow you to perform experimental analysis of two algorithms which you will need to code.

    a. Read the starter code to understand what it does.

    b. Complete the program by writing the code for the two functions `prefix_averages1`, and `prefix_averages2` given the pseudocode provided at the end of this document in Appendix A. Run the program on small input sizes to ensure the functions execute as expected. For example, if the input size is 5, and your input array is [30, 43, 3, 5, 9], the prefix averages computed would be: [30, 36.5, 25.33, 20.25, 18]. *Please note: you are free to add any other code to the program to help you during testing.*

2. (2 mark) Once you are sure your program works, you are now ready to perform experimental analysis on **<u>large input sizes</u>**. Decide on an appropriate initial input size, n_1, but keep in mind that the program will also execute on input sizes $n_2 = 2n_1$, $n_3 = 3n_1$, $n_4 = 4n_1$, and $n_5 = 5n_1$. (So if n_1 is 1000000, then n_5 would be 5000000.)

    a. For each function, record the times you obtained in a table like the one shown below: (*Note that the starter code is set up such that for each input size, the algorithm will execute 3 times*.)

Input Size	Run #1	Run #2	Run #3	Average Execution Time (sec)
n_1 =				
n_2 =				
n_3 =				
n_4 =				
n_5 =				

[1] Check the Class Plan for Assignment submission requirements

Please note the following:

- If your timings are erratic (e.g. 0s or there is no clear increase in time with respect to input size), try starting with a larger input size. *The goal is to slow the program down enough to be able to determine how time grows as the input size grows.*
- You are not required to use the same initial input size for both functions.

b. Plot the average running times obtained in part (a) as a function of *n* as scatter plots on a linear scale for each algorithm.

3. (2 marks) For each function - `prefix_averages1` *and* `prefix_averages2` - complete the table given below to help you determine the growth of running time with respect to the size of the input. Clearly **state** the experimental growth rate of the function given your execution times: *does it look more like* $O(n)$ *or* $O(n^2)$?

Input size	**Average Execution time in *ms*** (as determined by experiment)	**Actual Time Increase Ratio:** *Calculate* $\frac{t_i}{t_{i-1}}$	**Time increase factor if O(n):** *Calculate* $\frac{n_i}{n_{i-1}}$	**Time increase factor if O(n^2):** *Calculate* $\frac{{n_i}^2}{{n_{i-1}}^2}$
$n_1 =$	$t_1 =$	N/A	N/A	N/A
$n_2 =$	$t_2 =$			
$n_3 =$	$t_3 =$			
$n_4 =$	$t_4 =$			
$n_5 =$	$t_5 =$			

4. (6 marks) Repeat Questions #1- #3 above using the C file, `Assignment1.c`, as your starting point.

5. (1 mark) Given the experimental analysis you performed for `prefix_averages1` and `prefix_averages2` using Python and C, do the experimental growth rates align with the theoretical growth rates of these two algorithms? If not, please provide possible reasons for the discrepancies.

6. (3 marks) Al Gorythm, who is well versed in Python's built-in functions, claims he has the best implementation to calculate prefix averages. He imports the **mean()** function from the **statistics** module[2], and computes the prefix averages in the following way: [3]

```
# X is the input array
# A is the output array storing the computed prefix averages
for i in range (len(X)):
        A[i]= sum(X[0:i+1])/(i+1)
```

This new version, which we will call `prefix_averages3`, produces the correct output using very few lines of code. However, how does this version compare to `prefix_averages1` and `prefix_averages2` with respect to their *growth rates*? Add a function to the python program that defines `prefix_averages3` and perform experimental analysis as you did for `prefix_averages1` and `prefix_averages2`. Report your findings and explain why or why not Al's algorithm is the best of the three versions.

7. (2 marks) Assume `prefix_averages1` on an input size of n = 500000 (*five hundred thousand*) takes 3 minutes. Given that time, what do you predict the execution time to be when the input size grows to 15n (i.e., 7500000)? Justify your prediction.

8. (2 marks) Assume `prefix_averages2` on an input size of n = 500000000 (*500 million*) takes 3.5 seconds. Given that time, what do you predict the execution time to be when the input size grows to 15n (i.e., 7500000000)? Justify your prediction.

[2] To import the mean function, the following line has been added to the top of the Python program:

```
from statistics import mean
```

[3] **`mean(X[i:j])`** calculates the average of the elements in **`X`** from index **`i`** to index **`(j-1)`** inclusive.

Appendix A: Pseudocode of prefix_averages1 and prefix_averages2

prefix_averages1:

Input: An n-element array X of numbers.

Output: An n-element array A of numbers such that $A[i]$ is the average of elements $X[0], \ldots, X[i]$.

```
Let $A$ be an array of $n$ numbers.
for $i \leftarrow 0$ to $n-1$ do
  $a \leftarrow 0$
  for $j \leftarrow 0$ to $i$ do
    $a \leftarrow a + X[j]$
  $A[i] \leftarrow a/(i+1)$
return array $A$
```

prefix_averages2:

Input: An n-element array X of numbers.

Output: An n-element array A of numbers such that $A[i]$ is the average of elements $X[0], \ldots, X[i]$.

```
Let $A$ be an array of $n$ numbers.
$s \leftarrow 0$
for $i \leftarrow 0$ to $n-1$ do
  $s \leftarrow s + X[i]$
  $A[i] \leftarrow s/(i+1)$
return array $A$
```