Objectives of this assignment:

* to explore time complexity and “real time”
* to “dust off” programming skills

Use This File

* USE THIS FILE AS THE STARTING DOCUMENT YOU WILL TURN IN. **DO NOT DELETE ANYTHING FROM THIS FILE:** JUST **INSERT** EACH ANSWER **RIGHT AFTER ITS QUESTION/PROMPT**.
* IF USING HAND WRITING (STRONGLY DISCOURAGED), **USE THIS FILE** BY CREATING SUFFICIENT SPACE AND WRITE IN YOUR ANSWERS.
* FAILING TO FOLLOW TURN IN DIRECTIONS /GUIDELINES WILL COST **A 30% PENALTY.**

What you need to do:

1. Implement a simple algorithm to compute the sum where where and are a real numbers with .
2. Collect the execution time T(n) of algorithm A as a function of n

* Plot the functions , and on separate graphs. (See below what the functions , ,
* and are.)

1. Refer to the analysis of the time complexity your performed for your Module 1 and discuss it in light of the plots you plotted above.

Objective:

The objective of this programming assignment is to implement in your **preferred[[1]](#footnote-1)\*** language an algorithm A to compute the sum where and are a real numbers (). We are interested in exploring the relationship between the time complexity and the “real time” (wall time). For this exploration, you will collect the execution time of Algorithm A as a function of n and plot , and on different graphs. Finally, discuss your results: use the plots you will build to determine and justify the time complexity of . (**Hint**: analyze ahead the time complexity T(n) of *ComputeSumPowers* to predict the expected shapes of , and )

* Algorithm A
* ComputeSumPowers(a,x,n)
* **inputs:**  is a real number with . is a real number. is an integer()
* **output:** a real number equal to
* sum = 0
* prod = x
* for i = 1 to n
* sum = sum + prod
* prod = prod \* x
* return a\*sum
* Questions
* **IMPORTANT**: The following questions are meant as **hints** to guide you to analyze, predict, determine, and/or justify the shape of time complexities T(n). If you are given the plot T(n) and asked to determine whether T(n) grows like , , , ......., or , your task will in general be hard if you just plot T(n). The shape of the curve T(n) may be misleading as the scales on the x and y vary. A strategy to help determine the asymptotic growth is the following: if we suspect that T(n) grows like a function g(n), then plotting will help you. Indeed, if T(n) grows like a function g(n), the curve of will be look like an unmistakable **horizontal line** for large values of n.
* The following questions should help you understand why plotting is helpful and how to decide whether T(n) grows like g(n).
* For this assignment, we will suspect that T(n) grows as the functions , ,
* or :
* Insert your answers in THIS file after each question
* **a)** Suppose that for very large, where is a constant.
* i) **(1 point)** What would then the values of , and be, respectively? (just replace T(n) with
* and simplify the expression you get).
* .... answer here
* If we replace T(n) with Kg1(n) to which T(n) = K \* n.
* For g1(n):
* For g2(n):
* For g3(n):
* Space is left here for my answer to completely show on the next page of the document.
* ii) **(3 points)** Based on the expressions obtained in the previous question, what would then the **shapes** of the plots, and be, respectively?
* .... answer here
* For : This plot should look like a horizontal line since it is simply equal to the constant “K”. This implies there is simply a linear growth patten of T(n) in respect to n.
* For : The plot for this function should show a decay that approaches 0 as the value of n becomes larger and larger. This will mean that T(n) grows at a slower pace than this function.
* For : The plot for this function should show an upward sloping curve. This would mean that T(n) still grows faster than this function but the rate at which increases slower.
* **b)** Suppose that for very large, where is a constant.
* i) **(1 point)** What would then the values of , and be, respectively? (just plug in T(n)
* and simplify the expression you get).
* .... answer here
* So first we replace T(n) with K \* g2(n). That would give T(n) = K \* 0.5n2.
* For :
* For :
* = = k
* For :
* = = =
* ii) **(3 points)** Based on the expressions obtained in the previous question, what would then the **shapes** of the plots , and be, respectively?
* .... answer here
* For = this shows a linear increase with a slope in relation to 0.5K. The growth would be considered a constant rate of increase. This would show an ascending straight line.
* For = K This plot would show a horizontal line that would indicate the T(n) grows at the same exact rate as g2(n) with it simply being equal to the constant K
* For = The plot for this function would show an upward curve that would be more and more steep as the value of n increases. This shows that T(n) grows at a faster rate than the given function.
* **c)** Suppose that for very large, where is a constant.
* i) **(1 point)** What would then the values of , and be, respectively? (just plug in T(n)
* and simplify the expression you get).
* .... answer here
* First replace T(n) with K \* g3(n). Then simplify each expression.
* For :
* For :
* = =
* For :
* = = K
* ii) **(3 points)** Based on the expressions obtained in the previous question, what would then the **shapes** of the , and be, respectively?
* .... answer here
* For The plot for this would it decreases at 1 divided by the square root of n. This would show a decay that approaches 0 as n increases. This would mean it grows faster than T(n).
* For The plot for this would also be decreasing but at a more rapid steep decay. This shows that T(n) grows even more slower than this function.
* For The plot for this would be a horizontal line, indicating that T(n) grows at the same rate as this function with direct proportions.
* Program to implement (28 points)
* .... answer here
* import java.io.FileWriter;
* import java.io.PrintWriter;
* import java.io.IOException;
* // Jonathan Elder
* // CPSC 3270 Programming Assignment 1
* public class ComputeSumPowersApp {
* public static void main(String[] args) {
* double a = 5.0;
* double x = 0.80;
* int startN = 5000;
* int endN = 12000; // Updated as per your request
* int step = 250;
* try (PrintWriter out = new PrintWriter(new FileWriter("outputData.txt"))) {
* for (int n = startN; n <= endN; n += step) {
* long startTime = System.nanoTime();
* double result = computeSumPowers(a, x, n); // Store result to possibly use later
* long endTime = System.nanoTime();
* long duration = endTime - startTime;
* double function1 = (double) duration / n;
* double function2 = (double) duration / (0.5 \* Math.pow(n, 2));
* double function3 = (double) duration / (5 \* Math.sqrt(n));
* out.println(n + "," + function1 + "," + function2 + "," + function3);
* }
* } catch (IOException e) {
* System.out.println("File could not complete writing.");
* e.printStackTrace();
* }
* }
* public static double computeSumPowers(double a, double x, int n) {
* double sum = 0;
* double prod = x;
* for (int i = 1; i <= n; i++) {
* sum += prod;
* prod \*= x;
* }
* return a \* sum;
* }
* }

Actions to complete:

1) ssh into a **Tux** machine

2) clear the screen on the Tux machine (use the command clear)

3) Display the current date (use the command date)

4) Compile your program

5) Execute your program and interrupt it just to show how the execution starts

6) Take a screenshot of the Tux terminal and insert here (below). Your screenshot should look like this template screenshot (we should see the username, the date, the commands typed, and the results):



* .... (10 points for the screenshot if your program works and produces correct data) Insert here YOUR screenshot with Tux terminal. **Without screenshot with all required information 28 points will be taken off.**
* .... (18 points if **screenshot PLUS** program works and produces correct data) No credit will be awarded if this task is not completed on a Tux machine.

A screenshot of a computer program

Description automatically generated

There are some other files listed here other than just the ones for this project. But you can see my program ComputeSumPowersApp made a file a named outputData.txt. I tried to control + c twice as you can see but it was not needed to stop running the program to produce the file.

collectData()

for n = 5000 to L (with step 250)// L should be as large as your machine

// and your available time allow

Start timing // Note current time start work with nanoseconds if // language allows

ComputeSumPowers(5,0.80,n)

Stop Timing // T(n) = Current Time - **start**

Store the value n and the values T(n)/g1(n), T(n)/g2(n), and T(n)/g3(n) in a file **F** where T(n) is the execution time.

Data Analysis (60 points)

(3\*10 points per plot) Use any plotting software (e.g., Excel) to plot the functions , and in File F as a function of n (on different graphs). File F is the file produced by the program you implemented. Discuss your results based on the plots you obtain (3\*10 points per plot discussion). Think about using the preliminary questions asked above. Do not list here data as tables. Only plots are expected.

* .... insert here the plots and discuss them. We are especially interested in validating the time complexity you determined for the algorithm **ComputeSumPowers(a,x,n).**

A graph with blue lines

Description automatically generated

This plot does end up showing some fluctuations but keeps a level of growth that would suggest the algorithm’s time per unit of input does not dramatically increase as n grows. It would seem there is a linear growth pattern but it is not scalable as you would expect from something growing linearly. This could be due to the input size or the ability to optimize this further. From a practical perspective, an algorithm that shows this kind of ratio is likely to be predictable and manageable in terms of performance scaling. It suggests that doubling the input size n would roughly double the execution time.

A graph with red lines

Description automatically generated

In general, you can see from this plot that there is a decreasing trend as the value of n increases. This shows a fairly quadratic distribution. The execution time of the algorithm increases at a slower rate than n squared though due to the number being cut in half due to the constant 0.5. This would also indicate that the behavior is not acting as such within the test range. The decreasing trend is a positive sign in regard to the algorithm's efficiency, especially for large input sizes. It suggests that the algorithm may handle scaling better than a straightforward quadratic algorithm, which is beneficial for performance. This indicates efficient behavior relative to quadratic complexity. Moreover, the plot suggests that the algorithm could have optimizations or characteristics that reduce the impact of increasing input sizes, leading to less than expected increases in execution time.

A graph with green lines

Description automatically generated

This plot offers insights into how the execution time T(n) of the algorithm compares to a function that will scale with the square root of the input size n, that’s then multiplied by a constant factor of 5. This shows that the execution time T(n) grows at a rate faster than 5 times the square root of n. This behavior might hint at a complexity that could be polynomial but of a lower degree than quadratic or potentially a logarithmic factor combined with linear growth. This shows a more balanced scaling factor while at the same time losing performance over time during larger datasets or performance intensive moments. This faster scaling growth rate of T(n) shows how the algorithm could be better optimized at higher scales. Such optimizations could involve refining data structures, minimizing overhead, or simplifying computational steps, all aimed at aligning the practical performance more closely with the theoretical lower-bound expectations. Identifying and addressing the specific aspects of the algorithm that contribute to this accelerated growth rate could significantly enhance its applicability and performance, particularly in contexts where data volumes are large or computational resources are constrained.

ASK on ***Piazza*** for precisions if you have any doubts, concerns, or issues.

Let us know if you need help to work on Tux machines. (See at the end about how to log on Tux machines)

How to Plot?

I suggest to store the values in File F following the csv format used by Excel. Once the file F is in csv format, you can use Excel to plot.

If you do not know the csv format, google "csv format". Do not hesitate to ask for help if you need any.

Report

* Write a report using this file to insert your answers (Do not delete anything from this original file)
* Good writing is expected.
* Recall that answers must be well written, documented, justified, and presented to get full credit.
* Make sure that the TA has complete instructions/directions to compile and execute your program on Tux machines.

What you need to turn in:

I did not name my source program this and I true hope I am not counted off for that. Please!!

* Electronic copy of your source program (**collectData**)
* Electronic copy of the data (File F in csv format) you produce , and
* Electronic copy of the report (including your answers) (standalone). Submit the file as a Microsoft Word or PDF file.

Grading

* Each question shows the number of points for it

**Login on Engineering Unix Machines**,

Log in remotely on the Engineering Tux machines to implement, compile and execute. To log in remotely, you must use an **ssh** client such as SecureCRT (Windows).

On Windows 10, you may use from the command prompt the following command (if ssh is available):

ssh username@gate.eng.auburn.edu

where username is your Auburn University username (**without** @auburn.edu).

On Mac or any Unix machine (Ubuntu...), use the same command (see above) on a terminal.

1. \* You can use any language as long as it is already installed on Engineering Unix Tux machines. [↑](#footnote-ref-1)