Objectives of this assignment:

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

What you need to do:

1. Implement a silly (naïve and inefficient) 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
3. Plot the functions , , and on separate graphs.
4. 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 .

**Algorithm A**

**ComputeSumPowers(x,n)**

**inputs:**  is a real number with . is a real number. is an integer()

**output:** a real number equal to

sum = 0

for i = 1 to n

prod = 1

for j = 1 to i

prod = prod \* x

sum = sum +prod

return sum

**Answer These Questions**

These questions are meant as hints to analyze, predict, determine, and/or justify the shape of time complexities

**Insert your answers in THIS file after each question**

**a)** Suppose that for very large, where is a constant.

i) **(0.5 point)** What would then the values of , , and be, respectively? (just replace T(n) with

and simplify the expression you get).

.... answer here = K

=

=

ii) **(1.5 points)** Based on the expressions obtained in the previous question, what would then the **shapes** of the plots , , and be, respectively?

.... answer here Would be a horizontal line across the y-value equivalent to the constant K.

Would be a curve that starts at a large y value and approaches 0 as n grows. Graph used for reference attached below.

A screenshot of a computer

Description automatically generated with medium confidence

would be a curve that starts at a high y value, then approaches 0 as n grows. Graph used for reference attached below. I’m ignoring the curve on the left since it’s in the negatives, which wouldn’t make sense for time complexity, and because there’s an asymptote at n=1 that no value can reach.

A screenshot of a computer

Description automatically generated with medium confidence

**b)** Suppose that for very large, where is a constant.

i) **(0.5 point)** What would then the values of , , and be, respectively? (just plug in T(n)

and simplify the expression you get).

.... answer here = Kn3/2

= K

=

ii) **(1.5 points)** Based on the expressions obtained in the previous question, what would then the **shapes** of the plots , , and be, respectively?

.... answer here

Would start out slow, but quickly grow as a curve as n grows. Screenshot used for reference attached below.

A screenshot of a computer

Description automatically generated with medium confidence

= Would be a horizontal line across the y-value equivalent to the constant K.

Would start begin as a very large value, drop quickly, then begin to rise slowly again. Screenshot used for reference attached below. I’m ignoring the curve on the left since it’s in the negatives, which wouldn’t make sense for time complexity, and because there’s an asymptote at n=1 that no value can reach.

A screenshot of a computer

Description automatically generated with medium confidence

**c)** Suppose that for very large, where is a constant.

i) **(0.5 point)** What would then the values of , , and be, respectively? (just plug in T(n)

and simplify the expression you get).

.... answer here = Kln(n)

=

=K

ii) **(1.5 points)** Based on the expressions obtained in the previous question, what would then the **shapes** of the plots , , and be, respectively?

.... answer here

‘s plot would start by dipping in the negative y axis, then would continue it’s growth as if it were a normal logarithmic function as n grow larger. Screenshots attached, first one is to showcase the plot when n <=1 and the second is to showcase how the plot behaves as n grows large. These graphs were attached because they were used for reference while answering the question.

Chart

Description automatically generated with low confidenceA screenshot of a computer

Description automatically generated with medium confidence

Would start as a logarithmic plot that then approaches 0 as n grows. Screenshot used for reference attached below.

A screenshot of a computer

Description automatically generated with medium confidence

Would be a horizontal line across the y-value equivalent to the constant K.

**d)** **(4 points)** Time complexity of Algorithm A:

Report on this table the time complexity you obtained for the silly algorithm in M1:Homework (Refer to your homework)

|  |  |  |
| --- | --- | --- |
| Operations | Total Operations | Grows as |
| Comparisons | ½n^2 + (3/2)n + 2 | n^2 |
| Additions (line 6) | n+1 | n |
| Multiplications | n^2 + n | n^2 |

**Program to implement (28 points)**

.... answer here My program worked, produced data, and stored it in a csv file correctly. This is how you run it on a tux machine:

Navigate to the directory (I was on bmm0066@tux056) The file is just in that home directory.

programming\_1.java should be there

type in “javac programming\_1.java”

After compilation finishes, type in “java programming\_1”

The program should run and it will output to the system how many iterations are done. It runs till 20000.

Once done, there should be a file called “Retry.csv” with all the output data.

**State here whether your implementation worked and produced data.**

**Provide here the instructions to compile and execute your program on a Tux machine.**

collectData()

for n = 100 to L (with step 100)// L should be as large as your machine

// and your available time allow

Start timing // Note current time **start**

ComputeSumPowers(0.25,n)

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

Store the value n and the values T(n)/, T(n)/n2, and T(n)/n.ln(n) in a file **F** where T(n) is the execution time.

// Pay attention do not use n^2. The ^ operator is often not the exponentiation. Rather, it // is the exclusive OR (XOR)

**Data Analysis (42 points)**

(3\*7 points per plot) Use any plotting software (e.g., Excel) to plot the values , , 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\*7 points per plot discussion). Do not list here data as tables. Only plots are expected.

.... answer here

The raw data produced from my program would be the equivalent of T(n). The graph of the raw data appeared to grow as a polynomial function, more specifically it seemed to grow as Kn^2. Screenshot attached below. Based on this information, we can use my answers from **b)** to predict and analyze my plots. This is attached just to show my assumption that the algorithm grew as n^2 held true in testing.

Chart, scatter chart

Description automatically generated

Formula 1 (), also appears to grow as a polynomial function, but as n grows, we realize it doesn’t get to nearly as high values as the raw data did. This supports our claim from **b)** that would be = to Kn^(3/2). The plot is attached below.

Chart, scatter chart

Description automatically generated

Formula 2 (), appears to start at a high value, but it is important to pay attention to the scale for this graph. While the previous graphs reached values about 500000, this graph doesn’t even reach 20. It then appears to remain constant for the rest of its iterations, retaining its value right around one nanosecond. This constant horizontal line that holds true for the vast majority of the 20000 iterations, supports the claim that this graph for be a horizontal line. The plot for this formula is attached below.

Chart, scatter chart

Description automatically generated

Formula 3 (), starts at a low value, then grows consistently as n grows. It doesn’t grow nearly as fast as a polynomial function would, however. Again, it’s important to pay attention to the scale of the y-axis, and notice that this graph doesn’t reach beyond 4000 nanoseconds. This slow but steady growth after starting near 0, fits the description of this function from **b)** perfectly. This supports the claim that would be = to . The graph is attached below.

Chart, scatter chart

Description automatically generated

**Improve Algorithm A**

a) (**12 points**) Propose a more efficient algorithm to compute the sum such that the time complexity **grows as n**. **Use pseudocode to describe it**.

.... answer here

**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

for i = 1 to n

prod = a \* (x^i)

sum = sum + prod

return sum

This algorithm gives the correct answer by doing the main power (^) operation and adding the product to the total sum for every n value. Since there is a loop in this algorithm that has to iterate from 1 to n, the time complexity is dependent on n. Since it’s just one loop that iterates to n once, this algorithm grows as n.

B) (**8 points**) Propose a more efficient algorithm to compute the sum such that the time complexity is **constant** (independent of ). **Use pseudocode to describe it**.

.... answer here

**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

if (x != 1 && x != 0)

sum = a\*((1-x^n)/(1-x))

if (x == 0)

sum = 0;

if (x == 1)

sum = a \* n

return sum

For this question I found this formula for the sum of a geometric series that returns the answer to this problem. The screenshot of the information I found is below.

Text, letter

Description automatically generated

^This is the image from the web, it’s just a white background so it blends in.

Using this formula, I could replace x from our problem with r in the formula, and solve the normal case where 0<x<1 without having to make a loop to iterate to n. There are two cases where this formula doesn’t work however, x=1 and x=0. These two cases are easy to solve with arithmetic instead of a loop as well, since when x = 0, the result will always be 0 since it’s being multiplied in. The last case, x=1, is also easy to solve since it would just be a + itself n times. So a\*n is the solution for that case. Since this algorithm doesn’t loop through anything and doesn’t iterate to n for anything, it does the same operations the same amounts of times, every time it executes, no matter the input values. This causes the time complexity for this algorithm to be constant and independent of n.

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:**

* Electronic copy of your source program (**collectData**)
* Electronic copy of the data 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)