Homework 1: Why Growth Rates Matter

Data Structures & Algorithms

Due: *Monday, January 26th 2026 on Canvas*
Estimated time: 30–45 minutes

Goal

The goal of this assignment is to build intuition for algorithmic efficiency by running and interpreting real timing experiments. You will *not* be asked to implement new algorithms from scratch.

Instead, you will:

- Run existing code
- Observe how runtime changes as input size grows
- Answer short conceptual questions
- Make one small, safe code modification

This assignment is about understanding, not advanced C++ programming.

Part 0: Setup (5 minutes)

Before starting, make sure you have:

- A C++ compiler
- `gnuplot` (used for plotting results)

macOS

1. Install Xcode Command Line Tools (includes `g++`):

   ```
   xcode-select --install
   ```

2. Install `gnuplot` using Homebrew:

   ```
   brew install gnuplot
   ```

Windows

1. Install **MinGW-w64** or **MSYS2** to get `g++`.
2. Install `gnuplot` from:

   http://www.gnuplot.info/download.html

3. Make sure both `g++` and `gnuplot` are added to your system `PATH`.

If you have trouble with setup, ask early or come to office hours.

Part 1: Running the Experiments

Clone the course repository and navigate to the timing experiments:

```
git clone https://github.com/amoretti86/data_structures_algorithms.git
cd data_structures_algorithms/running_times
```

Compile and run the matrix multiplication experiment:

```
g++ matrix_plot.cpp -O2 -o matrix_plot
./matrix_plot
```

Then compile and run the algorithm comparison experiment:

```
g++ compare_plot.cpp -O2 -o compare_plot
./compare_plot
```

If `gnuplot` is installed correctly, the programs will generate image files (`.png`). If not, you can still answer all questions using the printed output.

Part 2: Matrix Multiplication ($O(n^3)$)

Q1. Observing growth

Based on the output of `matrix_plot.cpp`:

- Does the runtime grow roughly linearly?
- Roughly quadratically?
- Faster than quadratic?

Answer in 1–2 sentences, explaining your reasoning.

Q2. Doubling intuition

Choose two consecutive matrix sizes from the output (for example, $n = 100$ and $n = 200$).

1. Compute the ratio of the input sizes.
2. Compute the ratio of the running times.
3. Compare this to what you would expect from an $O(n^3)$ algorithm.

Do the observed results roughly match the theory? Why or why not?

Part 3: Comparing Algorithms

The second program compares:

- Linear search ($O(n)$)
- Sorting using `std::sort` ($O(n \log n)$)
- Bubble sort ($O(n^2)$)

Q3. Which algorithm "wins"?

For the largest input size tested:

1. Which algorithm is fastest?
2. Which is slowest?
3. Were any of them close at small input sizes?

Answer in 2–3 sentences.

Q4. Log–log plot intuition

The program generates a log–log plot (log of input size vs. log of runtime).

- What does a straight line in a log–log plot suggest about growth?
- Which algorithm appears to have the steepest slope, and what does that mean?

You do not need formal knowledge of logarithms; answer intuitively.

Part 4: Small Code Modification

Open `compare_plot.cpp` and find the comment:

```
// Linear Search - O(n) - do 1000 searches to make it visible
```

Change the number of repeated searches from **1000 to 5000**.
Recompile and rerun the program.

Q5. What changed?

After increasing the number of linear searches:

1. Did the *shape* of the linear-search curve change?
2. Did its *relative position* compared to sorting change?
3. What does this tell you about constant factors vs. growth rates?

Answer in 2–4 sentences.

Submission

Submit a short PDF or text file containing your answers to Q1–Q5.

- No screenshots required
- No code submission required

Takeaway

By the end of this assignment, you should understand:

- Why Big-O notation ignores constants
- Why "fast for small inputs" can be misleading
- Why growth rates matter more than raw speed

These ideas will be formalized in upcoming lectures.