CSCI205 Computer Science III Data Structures and Algorithms
Lab Assignment: Linux/Valgrind and Introduction to Algorithm Analysis

Please thoroughly read the document before asking questions.

Acceptable AI use: AI use is permitted but must be cited as described in the course outline. Submitting AI code without proper citation is dishonest and will be considered cheating.

Task One: Install Linux and Valgrind.

- **Windows Users:** Installing the Linux Subsystem is the easiest way to get up and running. https://docs.microsoft.com/en-us/windows/wsl/about You can also choose to dual boot or deploy Linux in a virtual machine

- **Mac Users with old school Intel Processor:** Although there is a version of Valgrind for Mac OSX it is incredibly problematic and not recommend for reliable use. The best way to get setup to run Linux is to install it on a virtual machine. I have used the following with easy success.

    - **Virtual Box:** https://www.virtualbox.org/
    - **Ubuntu:** https://www.dev2qa.com/how-to-install-ubuntu-on-virtualbox-mac/
    - You certainly don't have to use Ubuntu, but it is the most user friendly installation and interface.

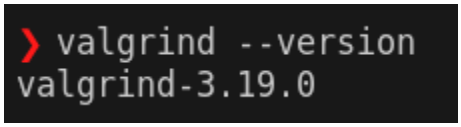
- **Mac Users with new school ARM M*N* processor**: Things may be easier for you folks in 2024. This was from 2022. Use your best judgment.

    - https://mac.getutm.app/ - link for UTM for Mac in the off chance anyone has ARM64 (aarch64) cpu. "Download" is a free download, the "Mac App Store" option is a paid supporter version but there is no difference.
    - https://mac.getutm.app/gallery/ubuntu-20-04 - here you can find a link to latest version of Ubuntu server for ARM iso download as well as setup instructions for getting your Ubuntu VM up and running
    - As a side note - UTM is open source. Their github repo is here: https://github.com/utmapp/UTM

- Once you have ***whatever version of Linux*** up and running, you can install valgrind by opening a terminal and following these steps.

    - **Valgrind on Ubuntu:** https://zoomadmin.com/HowToInstall/UbuntuPackage/valgrind
    - **Note:** This tutorial assumes Ubuntu and the **apt-get** package manager. If you did not install Ubuntu just search for the correct syntax to use with whichever package manager you have

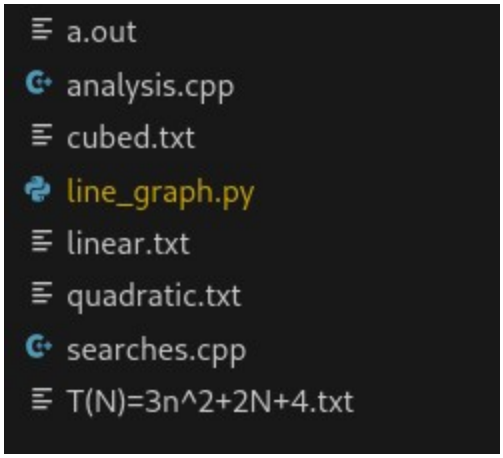
- **Submission:** Take a screen shot of you executing the following command from your Linux terminal: > **valgrind --version** Successful execution of this command will echo the version of Valgrind you installed. All I want to see is successful execution of this command. This screen shot should be saved in your repo directory for the week.

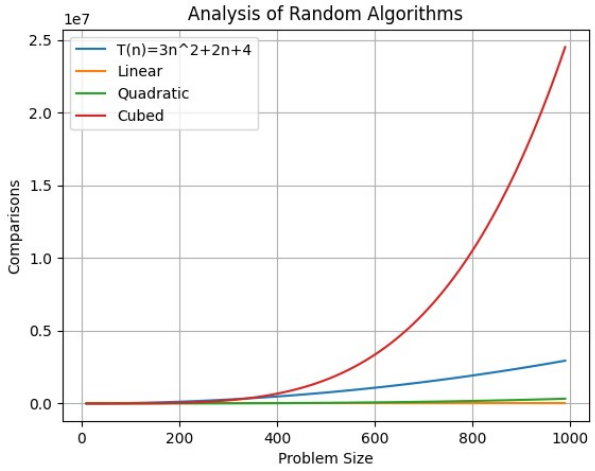


- From here on out, for the rest of the semester, your code **must** compile with **zero warnings** and in addition, run with **zero memory errors.** I will be checking this by doing the following
    - **Build syntax: *g++ -g -Wall -pedantic -o executable_name <list of cpp files>***
        - **-o:** name your *object file*, the new file representing the compiled and linked executable binary
        - **-g:** generate debug symbols, must use this prior to running valgrind or debugging
        - **-Wall:** The -Wall compiler flag is used to enable a broad set of warning messages during code compilation. It stands for "all warnings" and instructs the compiler to report a wide range of potential issues and non-standard code constructs. While it's not as strict as the **-pedantic** flag, it is still a valuable tool for improving code quality and catching potential bugs and problematic code patterns.
        - **-pedantic:** used to enforce strict adherence to the C++ language standard, which includes both the ISO C++ standard and any additional constraints imposed by the compiler itself. When you enable this option, the compiler will issue warnings or errors for code that does not conform to the standard or may have portability issues. Asking for this feedback is a great way to learn.
    - **Valgrind test: *valgrind --tool=memcheck --leak-check=full ./executable_name***
        - **Note:** “memcheck” is the default tool, I just want you to understand the syntax for tool selection.

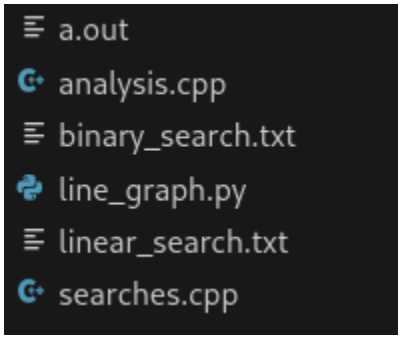
Task Two: Plotting data with Python . . . aka: automating your plotting

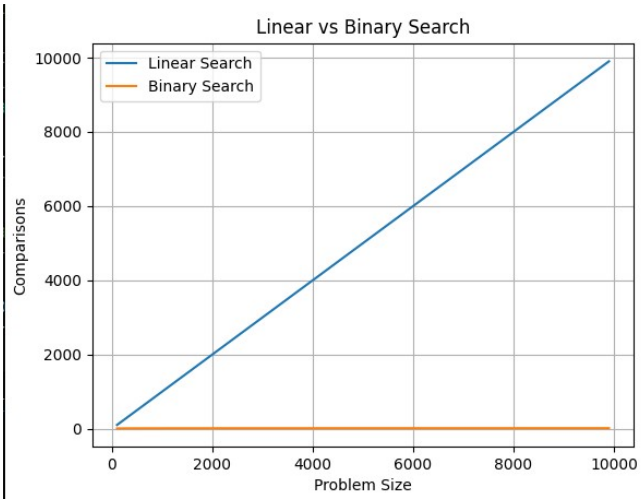
- You have seen how to quantify aspects of algorithms, now automate the plotting
- Research the Python modules ***matlotlib*** and ***pyplot***
    - **https://matplotlib.org/stable/tutorials/introductory/pyplot.html**
- Build a custom Python module that meets the following requirements
    - Will scan working directory for any files with a **.txt** extension. The assumption here is that each **.txt** file will contain plotting data in two columns: ***<problem size> <operation counts>***
    - The name of each **.txt** file will be used to label a plot. Check out the example below. This plot was generated by **line_graph.py** running in a directory containing **4 .txt files**





- The names of the four plots were generated from the names of the text files. The following rules apply
  - Plot labels should begin with a capital letter
  - You can use multiple words as shown in this example





  - Notice that there are two text files that were generated by running ***searches.cpp***
    - ***binary_search.txt*** and ***linear_search.txt***

```
ofstream f1("linear_search.txt");   // record counts of computations into a file
ofstream f2("binary_search.txt");   // record counts of computations into a file
```

    - the file names were processed as following
      - binary_search.txt => “Binary Search” plot label
      - linear_search.txt => “Linear Search” plot label
      - generalize this to process any file name with ***any number of underscores*** into a plot label

    - here_is_a_dumb_file_name.txt => “Here Is A Dumb File Name” plot label

- The name of the graph should be supplied via command line arguments when running the Python utility. The graph above was created and named by running the following command

```
❯ python3 line_graph.py "Linear vs Binary Search"
```

- Have fun with it. Maybe automate the entire process from building the executable, to running the experiments to creating the graph all with a single command.

- You will be submitting this and I will be using it to grade your experiments and plots

    - **LEAVE ME SOME DIRECTIONS** on how to use it. Please do not make me reverse engineer as you don’t want me to be grumpy while grading.

Task Three - Simple Big O Analysis: Design experiments to analyze and test the following algorithms. Where appropriate the functions should be executed multiple times with increasing problem sizes. Use your best judgment on limits, but you need a nice sample to make sense of the graphs. Tallies should be written to files and plotted using the utility described above. For each problem include best case, average case and worst case analysis. Include any other information you find interesting.

Prepare for Submission: Create a word document and do the following for each of these 9 problems

- Include a screenshot of the graph showing the plot of your experiment. The graph should be clearly labeled and easy to read. The plot should also align with the efficiency notation.
- **Include a brief description of your experiment, analysis and how you came to your conclusions, including the final Big O notation.**

Problems: To get the most out of the following problems, solve them yourself WITHOUT AI. By submitting this assignment you are claiming that you fully understand every detail of each of the algorithms. You will be expected to be able to describe these details verbally, without any technology.

If you do use AI for any aspect YOU MUST CITE IT. Failure to do so will be considered a violation of academic integrity.

Include the following functions in a single cpp file. The “main” function should be the first function definition in the file. It should be **very easy** to manipulate this function for me to test things individually. All code must be adequately commented.

Any dynamic memory should be freed appropriately.

1. **Constant Time – Array Middle Element:** Write a C++ function that takes an array of integers and returns the middle element. Analyze the time complexity and discuss why it is O(1). Ensure that the plot aligns with constant time.

2. **Linear Time – Array Range:** Implement a function to find the range of all elements in an array of integers using a loop. Analyze its time complexity and discuss why it is O(n). Ensure that the plot aligns with linear time.

3. **Linear Time - Exponential Function:** Implement an iterative function to calculate the result of raising a number to a power (e.g., x^n). Do not use any library exponentiation, use brute force iterative multiplication. Analyze and discuss its time complexity. Ensure that the plot aligns with your conclusions.

4. **Quadratic Time – Square Matrix Max Element:** Write a function to find the maximum element in a square matrix ***(number of rows == number of columns)*** of integers. The function should accept a reference to the matrix and the size N. N will represent both the number oif columns and number of rows.

   Analyze its time complexity, and discuss in detail why it is O(n^2). Ensure that the plot aligns with quadratic time.

5. **Linear Time - Factorials:** Write an ***iterative*** function to calculate the factorials of a number, not including 1 and itself. It must be O(n). Analyze its time complexity and discuss why it is O(n). Ensure that the plot aligns with linear time.

6. **Linear (or Better?) - Prime Numbers:** Using the concept of "trial division" write a function that implements an iterative solution to determine if any number N is prime. Analyze and discuss the time complexity of this approach. Implement the strategy that cuts trial division off at the square root of N. Analyze and discuss the time complexity of this approach. Graph both.

7. **Quadratic Time – Array has duplicates?:** Write a boolean function that determines if an array contains any duplicate items. Provide a detailed analysis of the time complexity and discuss why it is O(n^2). Ensure that the plot aligns with quadratic time.

Final Submission: Push the following to your repository

- Lack of professional documentation and commenting will be egregiously punished
- Screen shot of your installed valgrind version
- Your Python plotting utility
- Your C++ source file for the problem set
- A **PDF** containing your experiment descriptions, analysis and graph screen shots
- If I forgot anything in this list use your best judgment