**CMPSC 413 – Lab-1** (25 points)

**Algorithm Analysis**

**Note:** attach screenshots of your program and results under each programming exercises. Please make sure that the screenshot is readable. Don’t attach a very small screenshot image.

**Exercise-1:**

Code and analyze (time complexity – algorithm analysis) to compute the greatest common divisor (GCD) of two numbers.

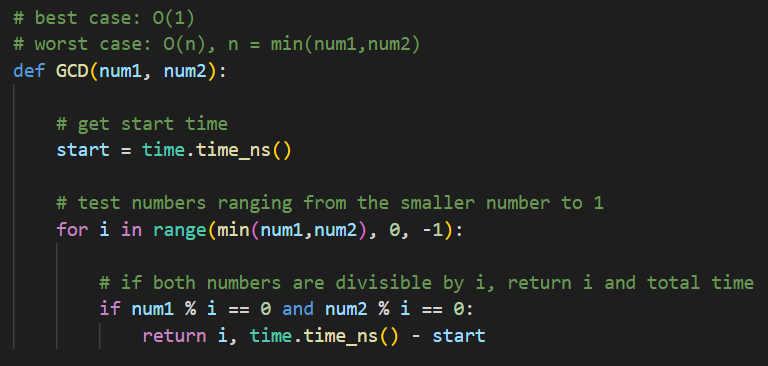
# Attach screenshots of the program and results

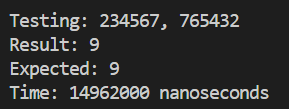
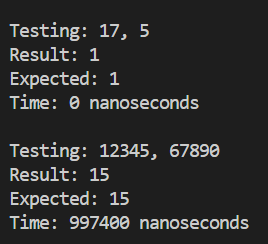
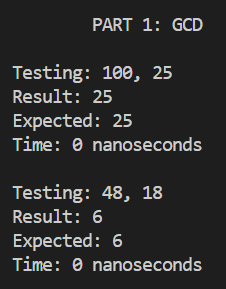
# Explain your understanding about the time complexity

I decided to not use the Euclidean algorithm and create my own. It is a simple for loop ranging from the smaller number to 1 (inclusive). We also use the smaller number as the upper bound of the range because it is able to be the GCD, whereas the larger number cannot be the GCD. The range is descending because we want to know the GREATEST common divisor, so it would be more efficient to check the larger numbers first.

Best case: **O(1)** because the GCD could be the first number.

Worst case: **O(n), n = min(number1, number2)** if it has to iterate through all numbers within range (GCD of 1).





**Exercise-2:**

Code and analyze (time complexity – algorithm analysis) to find maximum and minimum element from a list (or an array). Test it with 1000 random elements and 10,000 random elements. Also, calculate the actual CPU time it took find the minimum and maximum elements.

# Attach screenshots of the program and results

# Explain your understanding about the time complexity for finding minimum element.

# Explain your understanding about the time complexity for finding maximum element.

# CPU time to find minimum in 1000 and 10000 elements.

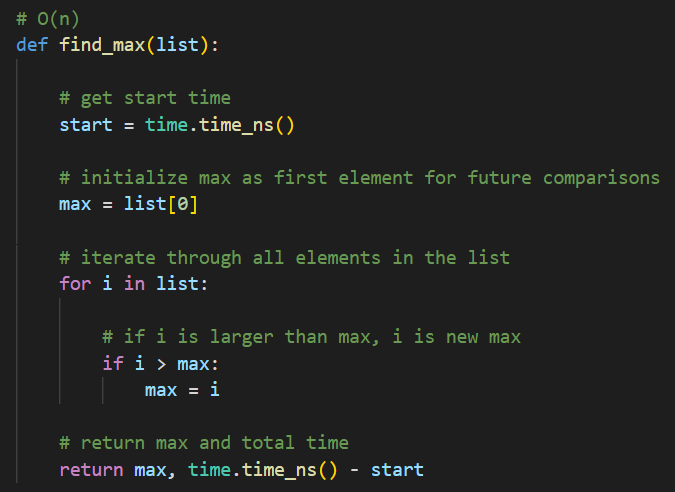
# CPU time to find maximum in 1000 and 10000 elements.

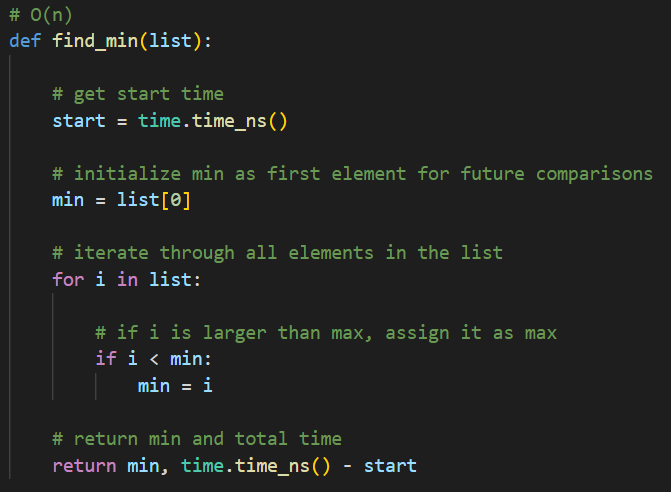
Both functions operate the same way:

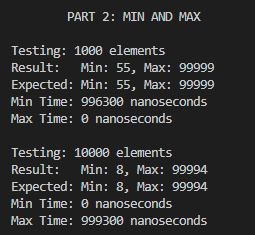
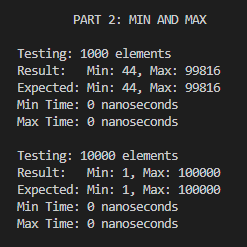
1. Get start time
2. Initialize min or max as the first element
3. Loop through the list and compare all numbers to the current min or max
   1. Min: if the number is less than the current min, it becomes the current min
   2. Max: if the number is greater than the current max, it becomes the current max
4. When all numbers are compared, return the min or max, as well as the total time taken

Since this is a basic for loop and each element is only visited once, the **time complexity will be O(n)**. The amount of work performed is proportional to the input size, n.

For the results, it seems to happen way too quickly, so I am getting 0 nanoseconds. However, sometimes I will get a time, so I know it is actually measuring the time it takes. I included screenshots of two different trials for each. I print the length of the list, the calculated and expected max/min, and the time it takes.







**Exercise-3:**

Write a program to find maximum and minimum element in a list (or an array) using Divide and Conquer strategy and analyze the time complexity. Also, calculate the actual CPU time it took find the minimum and maximum elements.

Steps to perform Divide and Conquer

1. To use divide and conquer as an algorithm design technique, you must divide the problem into two smaller sub problems, solve each of them recursively, and then merge the two partial solutions into one solution to the full problem.

2. Whenever the merging takes less time than solving the two sub problems, we get an efficient algorithm.

Divide-and conquer is a general algorithm design paradigm:

1. Divide: divide the input data S in two or more disjoint subsets S1, S2, …

2. Recurrence: solve the sub problems recursively

3. Conquer: combine the solutions for S1, S2, …, into a solution for S.

# Attach screenshots of the program and results

# Explain your understanding about the time complexity for finding minimum element.

# Explain your understanding about the time complexity for finding maximum element.

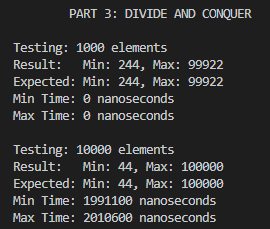
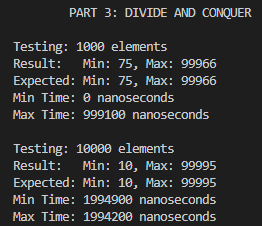
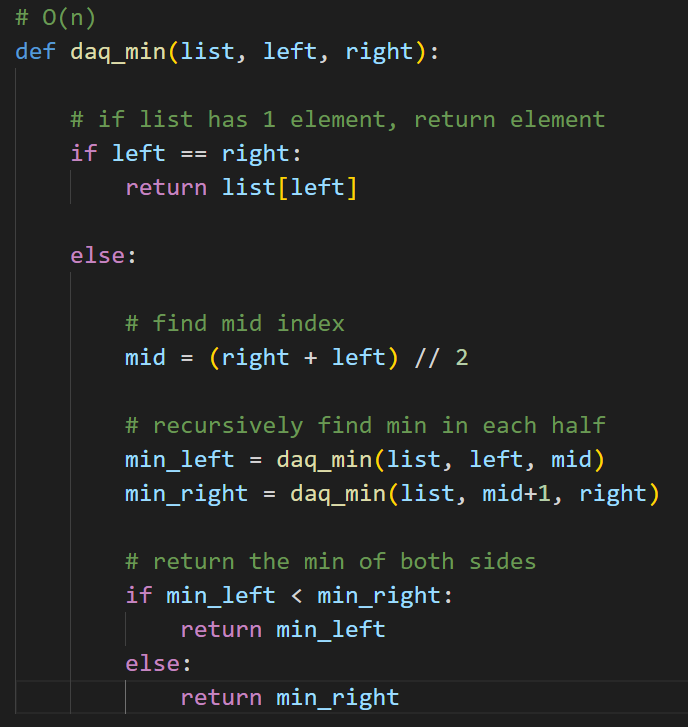
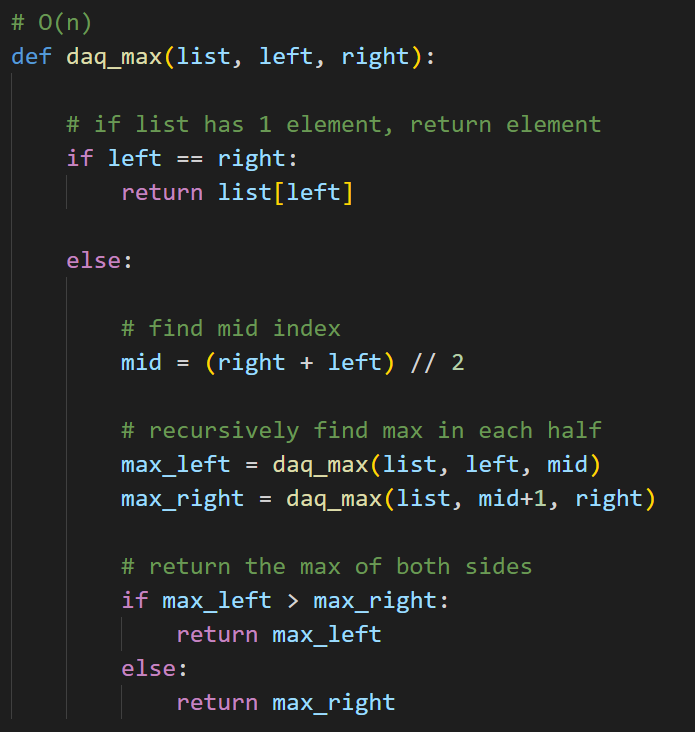
# CPU time to find minimum in 1000 and 10000 elements.

# CPU time to find maximum in 1000 and 10000 elements.

Both are basically the same function, so my explanation applies for both. The time complexity is **O(n)** because you half the number of operations as you divide the lists in half. The divisions are only to structure the comparisons and do not introduce any extra work. While some divide and conquer methods, like mergesort, would be O(nlogn), these functions do not do “n” work “logn” times.

Ex. for a list of 8:

* Layer 1: 1 comparison (1)
* Layer 2: 2 comparisons (n/4)
* Layer 3: 4 comparisons (n/2)  
    
  TOTAL: 7 comparisons or n/2+n/4+...+1 = n



CONCLUSION:  
  
Overall, I have a better understanding of how to identify the time complexity of simpler algorithms because of this assignment. I know my brute force approach for finding GCD is less efficient than the Euclidean Algorithm, but it is still incredibly efficient. I was initially expecting the divide and conquer method to be more efficient for finding the min and max, since it is a little more complex, but my results do not show that. Specifically, the linear approach consistently outperformed the divide and conquer approach for the larger list containing 10,000 elements. It was hard to tell with the linear approach how long it took because it seems to be very efficient. I provided screenshots of it showing a nonzero time just to prove the time was being tracked properly, but it took a couple of tries to get something to show. With that said, for most of the trials, the linear approach was superior when it came to the larger list. I would assume the amount of recursive calls is adding to the decreased performance compared to the linear approach.