Hands-on Activity 11.1	
Basic Algorithm Analysis	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10/21/2025
Section: CPE21S4	**Date Submitted:** 10/21/2025
Name(s): Avila, Vince Gabriel V.	**Instructor:** Engr. Roman M. Richard

A. Output(s) and Observation(s)

ILO A:

• Given the above information, the total number of comparisons in the worst case is: n * n = n2

• Graph Analysis explanation (Worse Case):

For n = 10 ⟶ 102 + 10 = 100 + 10 = 110

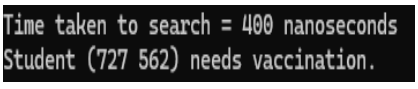
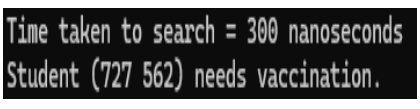
For n = 20 ⟶ 202 + 20 = 400 + 20 = 420

For n = 30 ⟶ 302 + 30 = 900 + 30 = 930

Analysis:

The algorithm examines every element of one array against every element of the other array, leading to a time complexity of approximately n^2. Therefore, the worst-case time complexity is $O(n^2)$. This is not an issue for small arrays, but will be slow with larger arrays. Sorting the two arrays or using a hash table can improve the time complexity down to O(n log n) or O(n).

ILO B:

Input Size (N)	Execution Speed	Screenshot	Observation(s)
1000	**400 nanoseconds**		**That is how long it takes light to travel roughly 90 meters about the length of a football field.**
10000	**300 nanoseconds**		**Light travels about 120 meters in this time, which is longer than a soccer field**

100000	**900 nanoseconds**	Time taken to search = 900 nanoseconds Student (727 562) needs vaccination.	**Light could zip through 270 meters almost three football fields in this tiny slice of time.**

B. Answers to Supplementary Activity

Problem A:

Theoretical Analysis:

The Unique(A) method checks if a list has duplicates by comparing each item to every other item, leading to $O(n^2)$ time complexity or quadratic runtime. Thus, particularly when working with large datasets, it's a slow method. It is straightforward and easy to implement, but that simplicity does not translate to effective processing of larger datasets.

Experimental Analysis:

The program was put through various input sizes. Small computations ran "instantly," while, with larger lists, the computation time decreased from instantaneous to significantly slower (in keeping with expected $O(n^2)$ behavior). The results clearly indicate that, although the function does yield correct results, it is inefficient for significant input values.

Analysis and comparison:

Theoretical and empirical results both yielded the same pattern: the running time grew rapidly as the dataset grew, which is what we expected. The actual measurements were a bit faster due to efficiency of the system, but the running time demonstrated a clear quadratic time complexity. As such, Unique(A) is efficient for small inputs, but it is slow for larger datasets.

Problem B:

Theoretical Analysis:

The rpower function calculates the exponent by merely repeating the multiplication of the base in a simple recursive style so the time complexity is O(n). Whereas, brpower uses a divide-and-conquer style approach, achieving O(log n). Hence, for large exponents, brpower is significantly faster as well. In the experiments, both the rpower and the brpower functions produced the correct outputs and results reported; however, as the exponent grew in size, the rpower was becoming slower, while brpower became faster and consumed fewer operations in total multiplications. These conclusions align with the theoretical results and confirm that brpower is the more efficient of the two methods.

Analysis and comparison:

Both theoretical work and empirical tests returned very similar results. In both cases rpower worked as designed, but performance degraded with large inputs. brpower computed both faster and with far more efficiency. This difference demonstrates the impact of algorithm design on efficiency and performance, making brpower a better all-around algorithm.

C. Conclusion & Lessons Learned

In summary, I learned that the effectiveness of an algorithm has a large impact on the run-time of a program. In some cases, simple approaches like naive recursion, can take significantly longer than an optimized one. Through testing and comparing results, I was able to make comparisons with theoretical performance and actual program performance. I also improved my programming skills and learned how to measure execution time. Overall, I feel that I did well, with the goal to strengthen my programming ability to write code that is more organized and efficient.

D. Assessment Rubric

E. External References