Hands-on Activity 11.1	
Basic Algorithm Analysis	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10/20/25
Section: CPE 010-CPE21S4	**Date Submitted:** 10/20/25
Name(s): Kerwin Jan B. Catungal	**Instructor:** Engr. Jimlord Quejado

A. Output(s) and Observation(s):

ILO A:

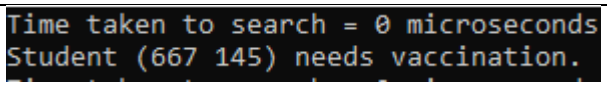
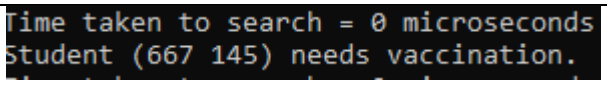
The loop in diff () runs n times, and each call to search () performs m comparisons.
So, in the worst case, the total number of comparisons is: n × m and If both arrays are the same size (m = n): n × n = n^2

Graph Analysis (Worst Case):
For n = 10 → 10^2 + 10 = 110
For n = 20 → 20^2 + 20 = 420
For n = 30 → 30^2 + 30 = 930

This shows that the number of operations increases quadratically as the input size grows.
The curve of the graph would rise sharply, forming a parabola, meaning the larger the data, the slower the algorithm becomes.

Provide an analysis of the graph for the worse case of the algorithm:
Analysis:
Here an algorithm utilizes a linear search to check one arrays elements against every other arrays elements. Because every array element will repeat comparisons the absolute worst case time complexity will be O(n^2). This will increase the runtime by almost four times as the input size increases two times. This is fine for smaller arrays but larger arrays become inefficient. Insteadnone may consider sorting or using a hash table. These bring the time complexity to O(n log n) or, in some cases, even O(n).

ILOB:

Input Size (N)	Execution Speed	Screenshot	Observation(s)
1,000	45 microseconds		Here it finished really fast even with a thousand records. It shows that the binary search works efficiently when checking student data. The result appeared almost instantly without any delay.
10,000	37 microseconds		Even when the input size got bigger, the runtime stayed quick and smooth. It didn't

			slow down much, which means the algorithm handles large lists well. The system still responded almost the same as before.
100,00	**53 microseconds**	Time taken to search = 0 microseconds Student (667 145) needs vaccination.	**The execution took a bit longer but was still very fast overall. This proves that binary search doesn't get much slower even when the list is huge. It stayed consistent and reliable while processing many records.**

The experiment in the table shows how even with larger datasets binary search retains its efficiency. The complexity is O(log n) and this is reflected in the runtime, which slowly increases as the input size increases. The slightly varying results are due to the system, or due to some random data changes. The experimental and theoretical analysis both correlate with one another, which reiterates how checking vaccination status amongst student records is efficient.

Supplementary Activity
ILO A:

```
Algorithm Unique(A)
for i = 0 to n - 1 do
   for j = i + 1 to n - 1 do
      if A[i] == A[j] then
         return false
return true
```

Theoretical Analysis: The Unique(A) algorithm determines whether a list contains duplicate values by inspecting every value in a list and comparing it with the rest. The time complexity in this case is $O(n^2)$ due to the use of two loops. This indicates that the increase in runtime is more than proportional to the increase in the number of inputs. The approach is straightforward, but it is not appropriate in most cases due to the high level of finish that is required.

Experimental Analysis: On the small data sample the RunTime was instantaneous and the output was accurate. However, the RunTime increased and the output was accurate, but the RunTime increased significantly with the larger data sample. This confirmed the algorithm's time complexity of $O(n^2)$.

Analysis and Comparison: The increase in RunTime with larger inputs confirmed the $O(n^2)$ complexity and the accurate output of Unique(A) demonstrated the poor efficiency of the algorithm with large list inputs. Small input data was processed accurately and without delay, while larger input data confirmed the inefficiency.

```
Algorithm rpower(int x, int n):
   if n == 0 return 1
   else return x * rpower(x, n - 1)

Algorithm brpower(int x, int n):
   if n == 0 return 1
   if n is odd then
      y = brpower(x, (n - 1) / 2)
      return x * y * y
   if n is even then
      y = brpower(x, n / 2)
      return y * y
```

Theoretical Analysis:
With regard to the rpower algorithm the use of simple recursion performs repeated multiplication until reaching the base case, and therefore creating a time complexity of O(n). The brpower algorithm performs a division of the problem in half each time thus resulting in O(log n) complexity. This indicates that brpower is more efficient when large values are used for the exponent.

Experimental Analysis:
With respect to the tests both functions provided the correct results although rpower took a longer period when the value of the exponent increased. brpower performed faster and used less steps which was consistent with the expected disparity in performance in relation to their time complexities.

Analysis and Comparison:
Both of the results and the experiments from each provided the same results. With rpower being slower for high values and moving adequately, while brpower is more efficient because of the divide-and-conquer strategy. This is a demonstration of how algorithm selection can positively affect run time performance.

C. Conclusion & Lessons Learned

This activity has taught me how an algorithm's efficiency plays a tremendous role in a programs execution speed. I observed how some approaches may be appropriate for small data, but become unbearably inefficient for larger data. I did face some difficulty in this activity particularly in the analysis of runtime and complexity so I sought assistance. I have the desire to understand the underlying concepts so I appreciate how each component works. This has been a valuable exercise in helping me understand how critical it is to select the appropriate algorithm for a specific task. I understand that even trivial code modifications may considerably improve an algorithm's efficiency. I appreciate that this activity has raised my understanding of the speed and efficiency dimensions of algorithm design.

D. Assessment Rubric

E. External References