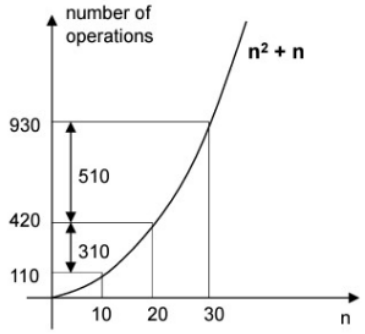
Activity No. 11.1	
Basic Data Algorithms	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 19/10/2025**
Section: CPE21S4	**Date Submitted: 20/10/2025**
Name(s): Crishen Luper S. Pulgado	**Instructor:** Engr. Jimlord Quejado

A. Output(s) and Observation(s)

ILO A

```
bool diff(int *x, int *y){
for(int i = 0; i < y.length; i++){
if(search(x, y[i])!= - 1){
return false;
}
}
return true;
}
```

The total of number of comparisons in the worst case is O(N x M) since in the outer loop, it runs through every element of array y. It runs n times since y.length = n. Inside the for loop, there is a condition statement. It calls the search(x, y[i]). So, for each iteration of the for loop calls the search (x,y [i]), which m comparisons per iterations.



n (size of array)	number of operations
10	110
20	420
30	930

The graph shows the function $n^2 + n$. At n=10 size, the + n has some effect. However, the n^2 dominates and increase number of operations at enormous amount. Hence, making the line steeper. The change in the number of operations is not constant since the gap of each n size becomes bigger than before. So, the worst case is $O(n^2)$ since it grows proportionally to n^2 and becomes impractical for large n.

ILO B

Input Size(N)	Execution Speed	Screen Shot	Observations
1000	0 microseconds	Time taken to search (1000 students): 0 microseconds Student (230, 789) needs vaccination.	For this size, it is much faster to search the student who needs a vaccination. The execution speed takes 0 microsecond, which tells us that it is much faster to search with a 1000 input size.
1000000	1 microsecond	Time taken to search (1000000 students): 1 microseconds Student (230, 789) needs vaccination.	For this size, I observed that it is much more noticeable that is much slower to search the student when it is increased by 1 million sizes. The execution speed takes 1 micro second to find the students

10000000	2 microseconds	Time taken to search (10000000 students): 2 microseconds Student (230, 789) needs vaccination.	For this size, it was increased multiplying by 10 which is 10 million sizes. I observed that it takes much slower to find which the result increased into microsecond when it was 10 million

Analysis: Theoretically, the code itself is O(n log n) due to the sorting vector of students. Since the time complexity of the other algorithms, such as the generating random students O(n) and binary search O(log n), is neglected due to the domination of the time complexity of the sorting algorithm. For the results in the compiler, it behaves accordingly to the theoretical complexity since it takes much more time when the size was increase. The more data it processes, the longer it takes.

B. Answers to Supplementary Activity

Theoretical analysis	Based on the given algorithm, the outer loop runs from i=0 to n-1. For each I, runs from j = I +1 to n-1 in the inner loop. So, when i=0, inner loop runs n-1; and when i=1, inner loop runs n-2. For total comparison $\frac{n(n-1)}{2}$ and the worst-case time complexity is O(n^2). The space complexity is O(1).
Experimental Analysis Input size (n): 1000 → Time taken: 0.0105 ms → Result: Duplicate Found Input size (n): 4000 → Time taken: 0.3311 ms → Result: Duplicate Found Input size (n): 16000 → Time taken: 1.6164 ms → Result: Duplicate Found	The result of the execution speed tells that it increases as the input size was increasing. Therefore, there is directly proportional relationship between the input size and the execution speed.
Analysis and comparison	We can tell that the experimental results support the theoretical analysis. The time grows slower as the input size is increasing.

Problem 1

Theoretical analysis	The first algorithm is the rpower(x,n) which multiplies the x by itself n times. For each recursive call, the n decreases by 1. The recurrence relation is T(n-1) + O(1) and the time complexity is O(n). The space complexity is O(n) due to the recursive call stack. The second algorithm is the brpower(x,n), which is instead of reducing n by 1, it reduces it by half at each recursive step. The recurrence relation is T(n/2) + O(1) and the time complexity is O(log n). The space complexity is O(log n) also.

Experimental analysis:
`Comparing rpower and brpower:`
`n = 1000 | rpower time = 0.017847 ms | brpower time = 0.000162 ms`
`n = 2000 | rpower time = 0.035958 ms | brpower time = 0.000169 ms`
`n = 3000 | rpower time = 0.038683 ms | brpower time = 0.000179 ms`
`n = 4000 | rpower time = 0.049902 ms | brpower time = 0.00011 ms`
`n = 7000 | rpower time = 0.150826 ms | brpower time = 0.000176 ms`
	Based on the result, the rpower algorithm is slower than the brpower algorithm. The rpower has a significant increase of milliseconds when increasing the size. Whereas the brpower stays almost constant in terms of time when increasing the input size.

Analysis and Comparison	The experimental and theoretical analysis supports each other. In theory, the rpower function follows a linear recursion approach. Whereas the brpower follows a divide and conquer like the merge sort. It reduces the exponent by half each recursive step. In the experiment, the execution time of rpower increases as n grows. While the brpower, results an almost a constant execution speed even n increases significantly, which demonstrate a logarithmic efficiency.

C. Conclusion & Lessons Learned

In conclusion, the running time of an algorithm has an effect when the input size grows. There are several factors that affects the running time and it is the hardware use and the software. However, we can focus on the algorithm itself by using a theoretical analysis to measure the possible execution speed and the space complexity. We use a function of n, where n is the size, to analyze the time complexity and space complexity. There are different functions to use in order to interpret the time and space complexity; such as linear, quadratic, logarithmic, exponential, and factorial. We also do an experimental analysis to compare the theoretical analysis. The procedure and the supplementary shows how the input size growth affects the execution speed. All of the activities show how different algorithm used can change the efficiency of the program. I think I had a hard time at analyzing the algorithm at first but I really notice their differences when comparing them. I think I need to improve more in theoretical analysis.

D. Assessment Rubric

E. External References