Activity No. <11>	
BASIC ALGORITHM ANALYSIS	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 10/18/25**
Section: CPE21S4	**Date Submitted: 10/18/25**
Name(s): ALCANTARA, JASON P.	**Instructor:** Engr. Jimlord Quejado

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

ILO A:

- **Given the above information, the total number of comparisons in the worst case is: $n * n = n^2$**

- **Graph Analysis explanation (Worse Case):**
  For n = 10 → $10^2 + 10 = 100 + 10 = 110$
  For n = 20 → $20^2 + 20 = 400 + 20 = 420$
  For n = 30 → $30^2 + 30 = 900 + 30 = 930$

  This equation $n^2 + n$, means that the number of operations grows quadratically as n increases.

- **Final Verdict:**
  - The algorithm checks each element of one array against all elements of the other array using a linear search. As a result, in the worst case the runtime expands proportionally to the square of the array size. This means the worst-case time complexity is:
  $T(n) = n^2 + n = O(n^2)$

  As n increases, the algorithm becomes significantly slower. For small inputs this is acceptable but for the large arrays, a more efficient method like sorting or using a hash table would reduce the complexity to O(n log n) or even O(n) in some cases.

ILO B:

Input Size (N)	Execution Speed	Screenshot	Observation(s)
1000	0 microseconds	`Time taken to search = 0 microseconds` `Student (202 882) needs vaccination.`	The search executed instantly. The student needs vaccination.
10000	0 microseconds	`Time taken to search = 0 microseconds` `Student (202 882) needs vaccination.`	Execution time is still 0 microseconds, meaning the program runs very fast.
100000	0 microseconds	`Time taken to search = 0 microseconds` `Student (202 882) needs vaccination.`	Execution time remains 0 microseconds. The time difference is too small to detect.

B. Answers to Supplementary Activity:

Problem A:

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

Theoretical Analysis:

The Unique(A) function is a tool that can identify if there are any element duplicates in the given list. Time complexity for this operation is O(n^2) since all the elements are compared with every other one. Thus if there is an increase in the number of unique elements, the time taken for processing will also increase rapidly. The approach is simple and easily comprehensible, but it is not good for large scale datasets in terms of efficiency. In short, it is a basic and slow method to test for uniqueness.

Experimental Analysis:

The program was subjected to the experiment with various input sizes. With minor data, it was instantaneous, but it became slow with a very large list. This was in accordance with the anticipated O(n^2) growth. The output was unambiguous, that is, the performance decreases as the input size grows. It validates the function to be correct but not appropriate for large inputs.

Analysis and comparison:

Theoretical and experimental findings gave a similar trend. Running time rose rapidly with the increase in data size which was the expected behavior. Real measurements recorded a bit faster time due to the system's efficiency. Nevertheless, the trend followed the quadratic time complexity. Thus, it is confirmed that Unique(A) is effective with small inputs but is not suitable for large ones in terms of efficiency.

Problem B:

```
Algorithm rpower(int x, int n):
1 if n == 0 return 1
2 else return x*rpower(x, n-1)
Algorithm brpower(int x, int n):
1 if n == 0 return 1
2 if n is odd then
3 y = brpower(x, (n-1)/2)
4 return x*y*y
5 if n is even then
6 y = brpower(x, n/2)
7 return y*y
```

Theoretical Analysis:

The rpower function depends on straightforward recursion, which symbolizes the repetitive multiplication of the base. Thus it has a time complexity of O(n). In contrast, brpower, through its faster divide-and-conquer approach, gets O(logn). Consequently, for large powers, brpower would be the one delivering the performance gain. In theory, brpower is already an efficient and practical method for large number crunching.

Experimental Analysis:
In testing, both functions gave correct results. But rpower was slower when the value of the exponent increased. brpower, however, executed in a quicker time and used fewer multiplications. The findings did exactly follow the theoretical predictions. It demonstrated that brpower is the more efficient and quicker function.

Analysis and comparison:
Theory and experiment yielded the same results. The rpower function is functional but is slower to complete with large input values. brpower executed better due to its smaller execution time. The contrast between their findings emphasizes the influence of algorithm design. For the overall, brpower was the more effective option than rpower.

C. Conclusion & Lessons LearnedL:

- **What I learned from this laboratory activity is how the efficiency of an algorithm impacts the speed at which a program operates. It made me realize that certain methods, such as straightforward recursion, can be less efficient than using optimized procedures. By running tests and comparing the outcomes, I could observe how theory aligns with real program behavior. The exercise also enhanced my coding proficiency and provided me with a better understanding of how to benchmark execution time. I understood the significance of using the proper algorithm for various problems. On the whole, I believe I performed adequately, but I would like to get better at writing more effective and structured code.**

D. Assessment Rubric

E. External References