Question 1:

(a)

i- Naive Iterative Method (power_i):

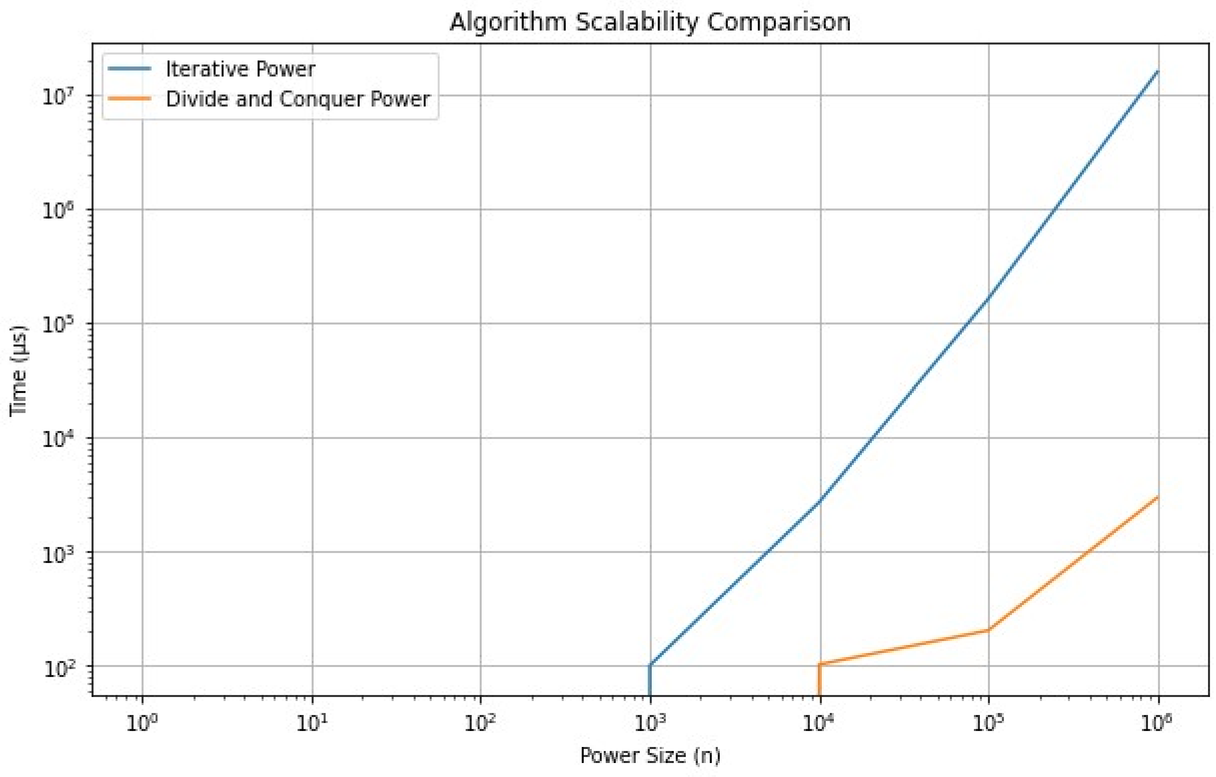
- the time complexity is O(n) in big theta notation. The algorithm's performance grows linearly with n.

ii. Divide and Conquer Approach (power_divid):

- The time complexity is O(log n) in big theta notation. The recurrence relation for this approach is: T(n) = T(n/2) + O(1). Solving this recurrence relation indicates that the time complexity is logarithmic, where the problem size is divided by two each step.

(d) Checking Correctness through Empirical Analysis:

The provided python code iterates through different power sizes (from 1 to 10^6), measures the execution time for each method, and plots the results in a graph. By observing the graph, the scalability of each algorithm can be visually inspected. The plot should demonstrate linear growth for the iterative method and logarithmic growth for the divide and conquer method, validating that the running time matches the predicted running time.



Question 2:

Asymptotic Running Time Complexity:

We can think of the time it takes to solve this problem. The sorting part (Merge Sort) takes longer when we have more numbers to sort. As we double the number of numbers, the time it takes roughly multiplies by two times the logarithm of the number of items.

Then, after sorting, we look through the list. This looking part (Binary Search) is faster, but as the list gets longer, it takes a bit more time, roughly following the logarithm rule.

Combining the sorting and searching steps, it takes longer ((O(n log n) + n times O(log n))) to solve the problem, where (n) is the number of items.

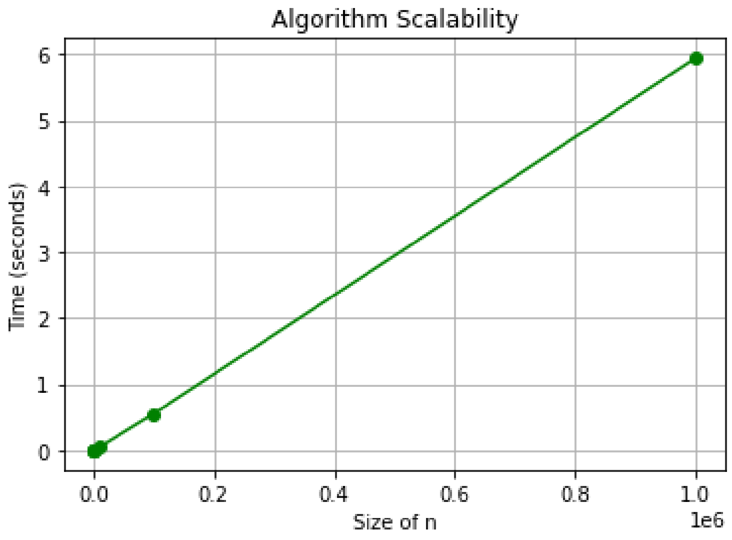
Recurrence for the Divide-and-Conquer Algorithm:

When we break the sorting step into smaller tasks and merge them back together (like dividing a task into smaller tasks and then recombining them), it takes a certain amount of time. With sorting, the time it takes roughly follows a rule where if we double the work, it takes a bit more than double the time.

Comparison:

When we actually do this in practice, sometimes the time taken might not exactly match these rules. For smaller sets of numbers, the actual time might be faster or slower due to how the computer works.

However, as we deal with more and more numbers, we'll notice that the time taken follows the rules we talked about earlier. The time it takes grows in a predictable way as the number of items gets bigger, even though it might not be exact for small groups of numbers


Algorithm Scalability
Time (seconds)
Size of n
1e6
0
1
2
3
4
5
6
0.0
0.2
0.4
0.6
0.8
1.0