Q1)

(b)

i. Iterative Approach:

In the iterative method, we multiply 'a' by itself 'n' times in a loop. The time complexity can be expressed as O(n) because it directly depends on the exponent 'n'. Therefore, the asymptotic running time complexity of the naive iterative method is Θ(n).

ii. Divide-and-Conquer Approach:

The divide-and-conquer approach to computing a^n involves splitting the problem into smaller subproblems and recursively solving them. In each step, we divide 'n' by 2, and the recurrence relation for the running time can be expressed as follows:

$T(n) = T(n/2) + O(1)$

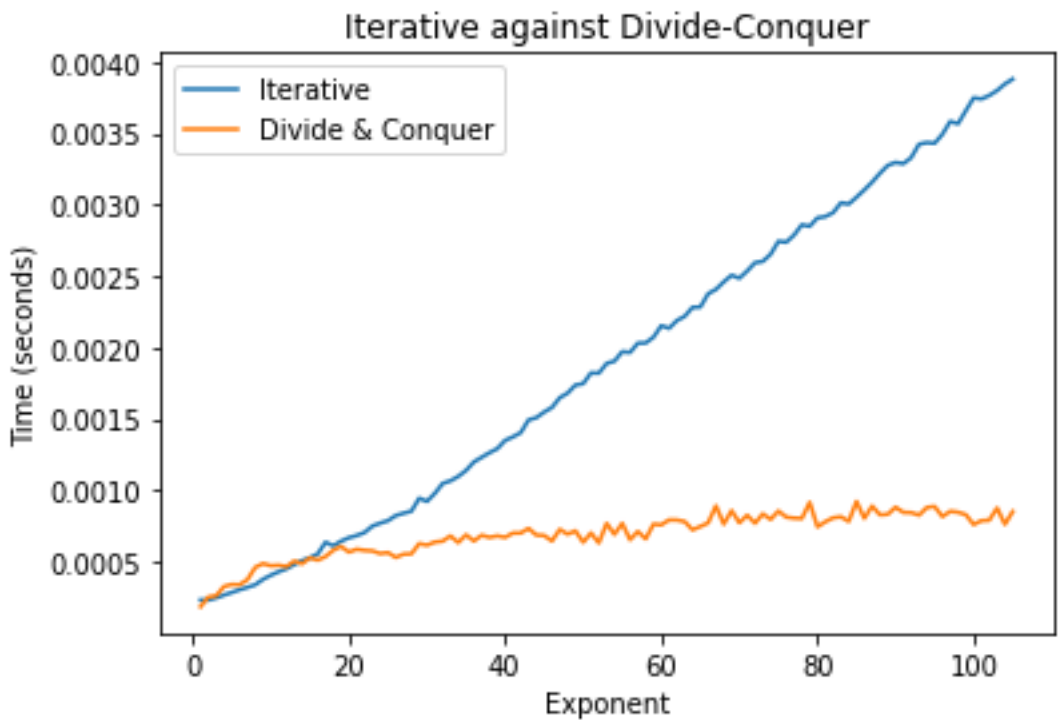
This recurrence represents the time complexity of the divide-and-conquer algorithm. To solve it, we can use the Master Theorem, which has the form:

$T(n) = aT(n/b) + f(n)$

In this case, a = 1 (since we're dividing the problem in half), b = 2, and f(n) = O(1) (the work done outside the recursive calls).

In our case, f(n) = O(1) and a = 1, b = 2. Since O(1) is a constant, it falls under the second case. Therefore, using the Master Theorem, the asymptotic running time complexity of the divide-and-conquer algorithm is Θ(log n).

(c)



(d) Experimental results confirms the theoretical analysis results as the graph shows

Q2)

(b) Merge Sort: The Merge Sort algorithm has a time complexity of O(n log n), where 'n' is the number of elements in the array. This is the dominant part of the algorithm.
Binary Search: The Binary Search algorithm has a time complexity of O(log n) for each search. In the worst case, we perform it for each element in the array.

Therefore, the overall time complexity of the proposed algorithm is O(n log n) for the Merge Sort and O(n log n) for Binary Search in the worst case. Since both components contribute the same order of complexity, the overall time complexity remains O(n log n).

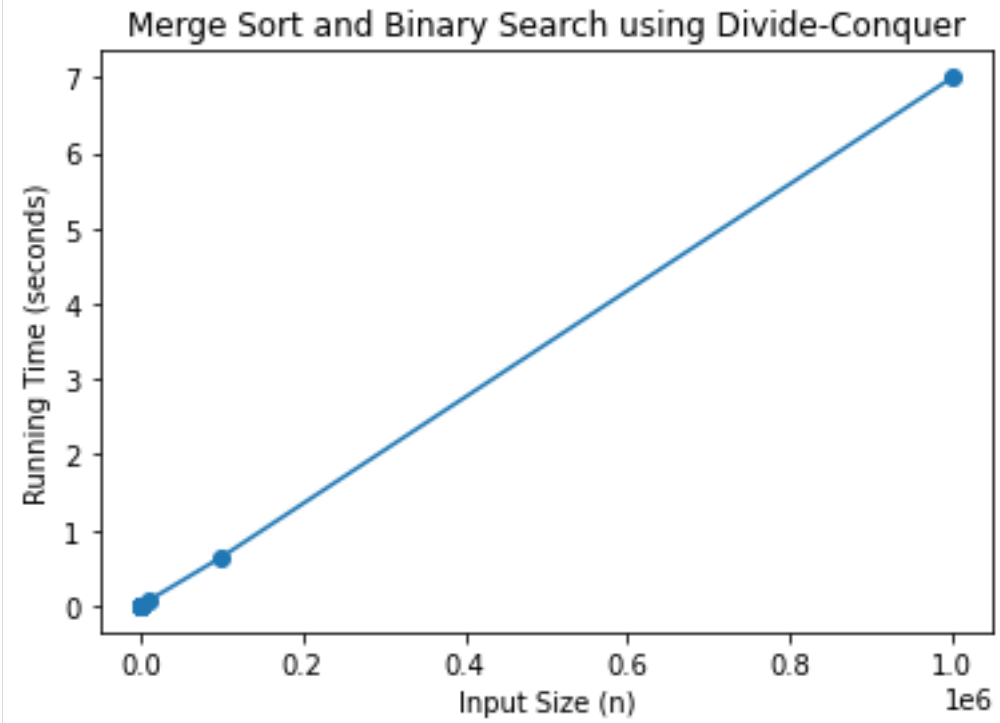
Now, let's consider the divide-and-conquer part, which is the Merge Sort algorithm. The time complexity of Merge Sort can be expressed by the following recurrence relation:

$T(n) = 2T(n/2) + O(n)$

Using the Master Theorem, we can determine the time complexity of Merge Sort. In this case, $a = 2$, $b = 2$, and $f(n) = O(n)$.

In this case, $f(n) = O(n)$, which falls under the second case. Therefore, the time complexity of the Merge Sort component is O(n log n), which matches the overall time complexity of the entire algorithm

(c)


Merge Sort and Binary Search using Divide-Conquer
Running Time (seconds)
0
1
2
3
4
5
6
7
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Input Size (n)