Analysis assignment 1

Monika Sameer Danial - 52-1490

November 3, 2023

1 Question1

1.1 code

1.2 Determine each algorithm's asymptotic running time complexity using the big theta notation. Solve the recurrence for the divide-and-conquer algorithm.

i. Naive Iterative Method: The naive iterative method involves a loop that multiplies a by itself n times. The time complexity is O(n), which means it grows linearly with n. ii. ii. Divide-and-Conquer Approach: The recurrence relation T(n) = T(n/2) + O(1), the time complexity is O(log(n)).

The recurrence relation for Merge Sort can be expressed as follows:
T(n) = 2 * T(n/2) + O(n)

1.3 One way to check the correctness of the above asymptotic analysis is to code up the program and see if the empirically observed running time matches the running time predicted by the analysis. Determine the scalability of each algorithm experimentally by running it with different power sizes n ranging between 1 and 106, and plot experimental results in a graph.

in figure 1

1.4 Determine whether experimental results confirm the theoretical analysis results in 1.(b)

You can compare the experimental results (the plotted graph) with the theoretical analysis (big theta notation) to check if they match. If the experimental results show a similar trend, it would confirm the theoretical analysis results.

2 Question 2

2.1 Write an efficient program utilizing the divide-and-conquer paradigm and applying the Merge Sort and the Binary Search algorithms to determine, in a given set S of n integers, all pairs of integers whose sum is equal to a given integer.

code

2.2 Determine the asymptotic running time complexity of the proposed algorithm. Solve the recurrence for the divide-and-conquer algorithm.

the time complexity is O(n log n) because merge sort has time complexity of O(n log n) and binary search has a time complexity of O(log n) a = 2 because it divides the problem into two subproblems.

n/b = n/2 because it divides the array into two equal halves. f(n) is the time it takes to merge the two halves, which is O(n). So, the recurrence relation for Merge Sort is: T(n) = 2T(n/2) + O(n) using master theorem :

f f(n) = $O(n^c)$for some constant $c < log_b(a)$, then $T(n) = (n^log_b(a))$. If $f(n) = (n^log_b(a))$, then $T(n) = (n^log_b(a) * logn) =$. If f(n) = $(n^c)$for some constant $c > log_b(a)$, and if $a * f(n/b)k * f(n)$ for some constant $k < 1$ and sufficiently large n, thenT(n) = (f(n)).

$c = log - b(a)$, which falls into case 2 of the master theorem. Therefore, the time complexity of Merge Sort is:

$T(n) = (n^log_2(2) * logn)$, $T(n) = (n * logn)$

So, the time complexity of Merge Sort is (n log n).

2.3 Determine the scalability of the algorithm experimentally by running it with different power sizes n ranging between 1 and 106, and plot experimental results in a graph. Compare the empirically observed running time with the running time predicted by the analysis in 2. b.

in figure 2,

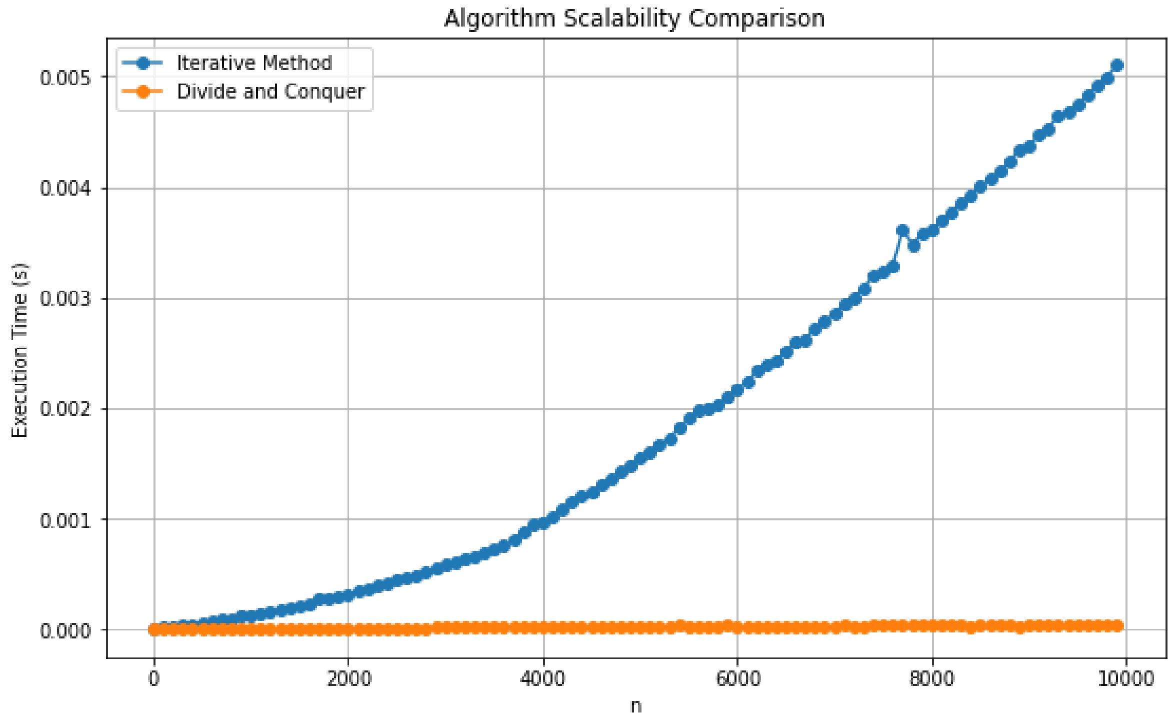

Figure 1: plot showing both Na ıve iterative and divide and conquer in q1

References

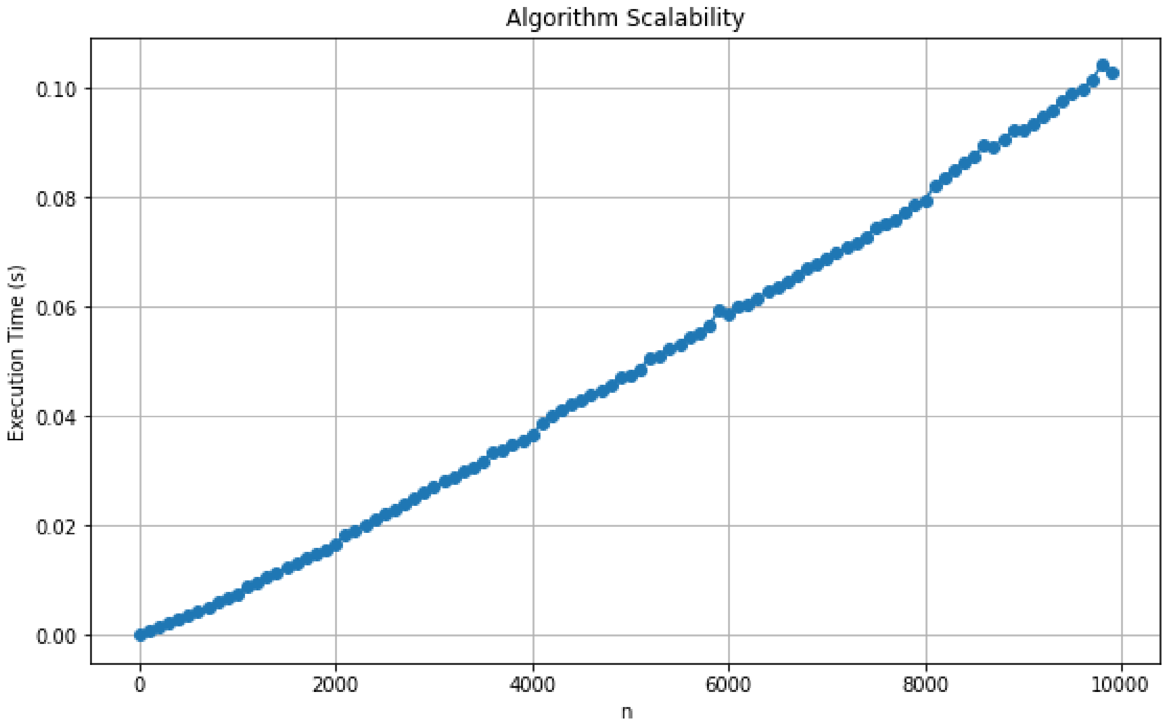

Figure 2: plot showing question 2