Analysis Assignment 1

Ahmad Safwat Elewa – ID: 13001011

Question 1

a) Here I assumed that a = 2 to simplify the calculations.

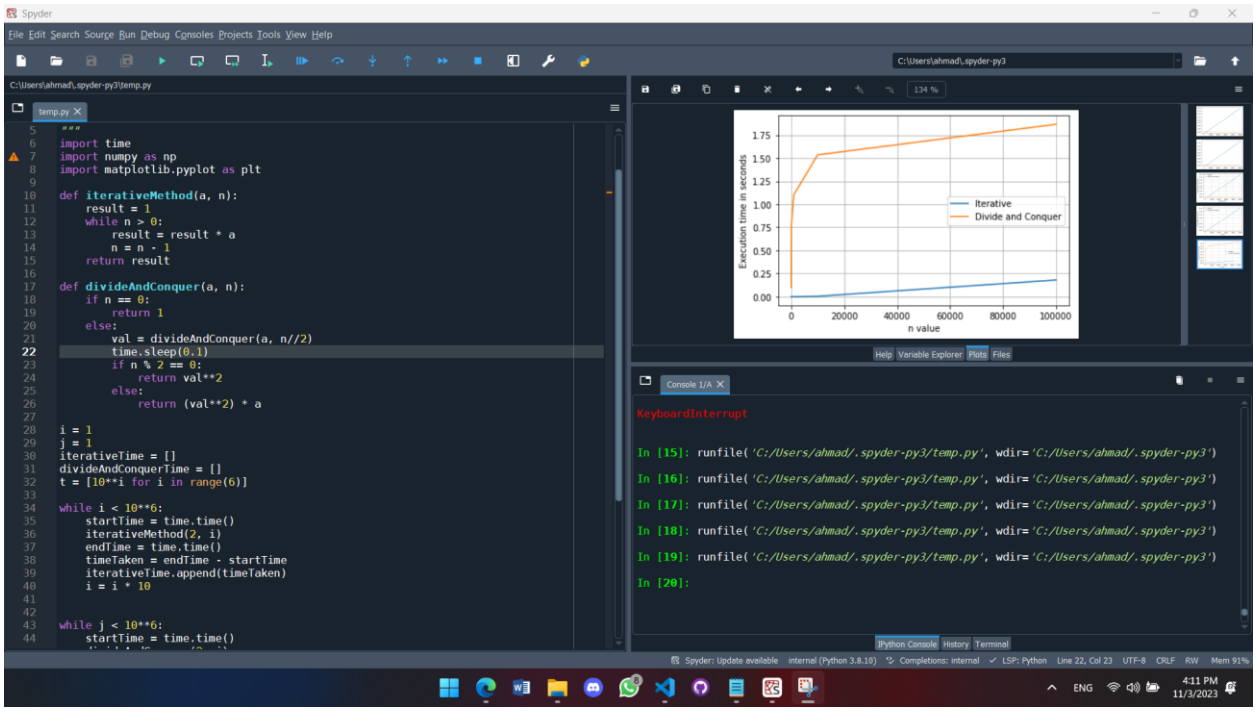
b) T(n) = aT(b) + D(n) + C(n), where 'a' is the number of subproblems per division, 'b' is the size of each subproblem in terms of 'n', D(n) is the divide time and C(n) is the combine time.

In our case, T(n) = T(n/2) + o(1), for all 'n' except 1, where T(1) = o(1).

For this question, we will solve using the master's theorem.
a = 1, b = 2, and f(n) = O(1).
n^(logb(a)) = n ^ (log2(1)) = n^0 = 1
T(n) = O(n^(logb(a)) * log(n)) = O(1 * log(n)) = O(log(n))

While the regular iterative algorithm will have a time complexity of o(n), due to linearly scanning the array once according to its size (n).

c) This graph shows the running time of each algorithm plotted in one graph.

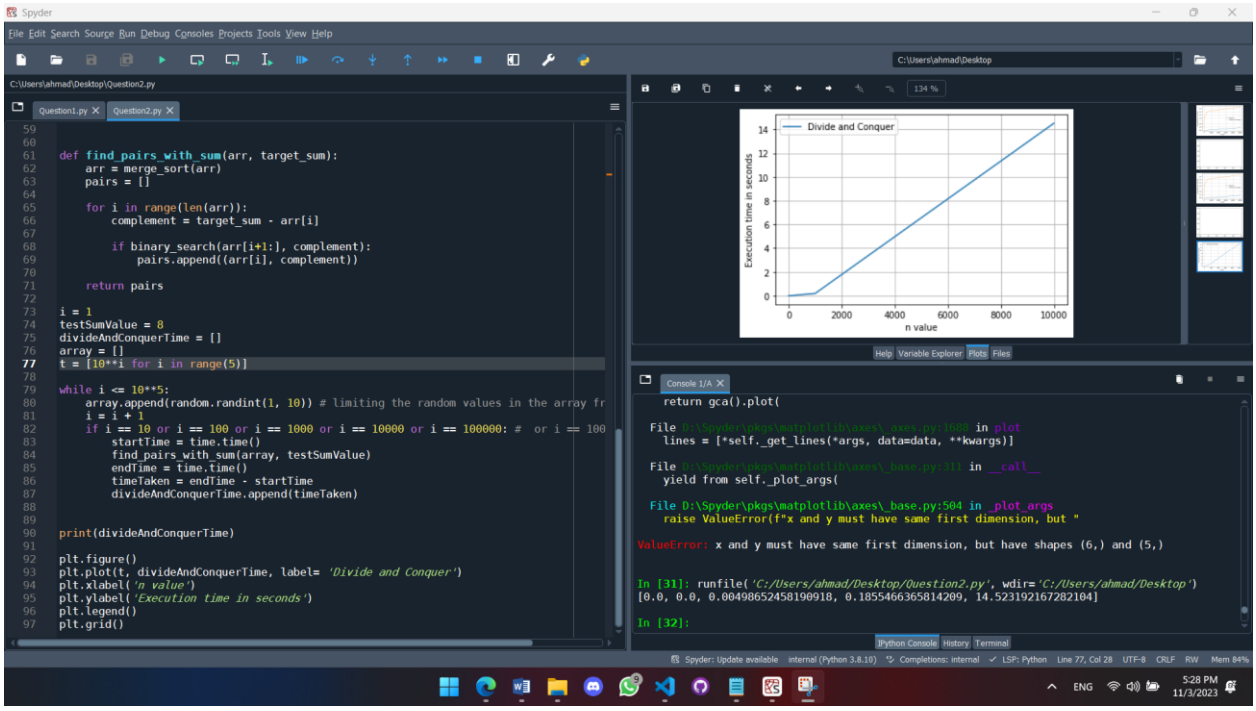


d) Comparing this graph with the recurrence relations we have found in 1b), the results match. (Note that the divide and conquer have an added delay to display better results).

Question 2

a) The code can be found in the .py file.

b) The asymptotic time complexity of my algorithm is 'T(n) = 2T(n/2) + O(n) + O(1)' for all n except 1 (in that case: T(n) = O(1)), since there are 2 splits when the merge sort takes place (a = 2), and there is splitting in the input size in both binary search and merge sort (b = 2), and the combine time is O(n) due to the combine time of the merge sort merging 2 sub-arrays together.
To solve this recurrence relation, we can draw the recurrence tree, which will have a height of 'log(n)', the cost of leaves being 'cn', and the inner part of the tree costing 'cn(logn)', for some constant c.
This means, T(n) = cnLogn + cn = theta(nLogn)



c) This graph shows a graph which resembles nlogn, approving our theoretical asymptotic time complexity.