# Analysis & Design of Algorithms
# Assignment 1

## Question 1 :

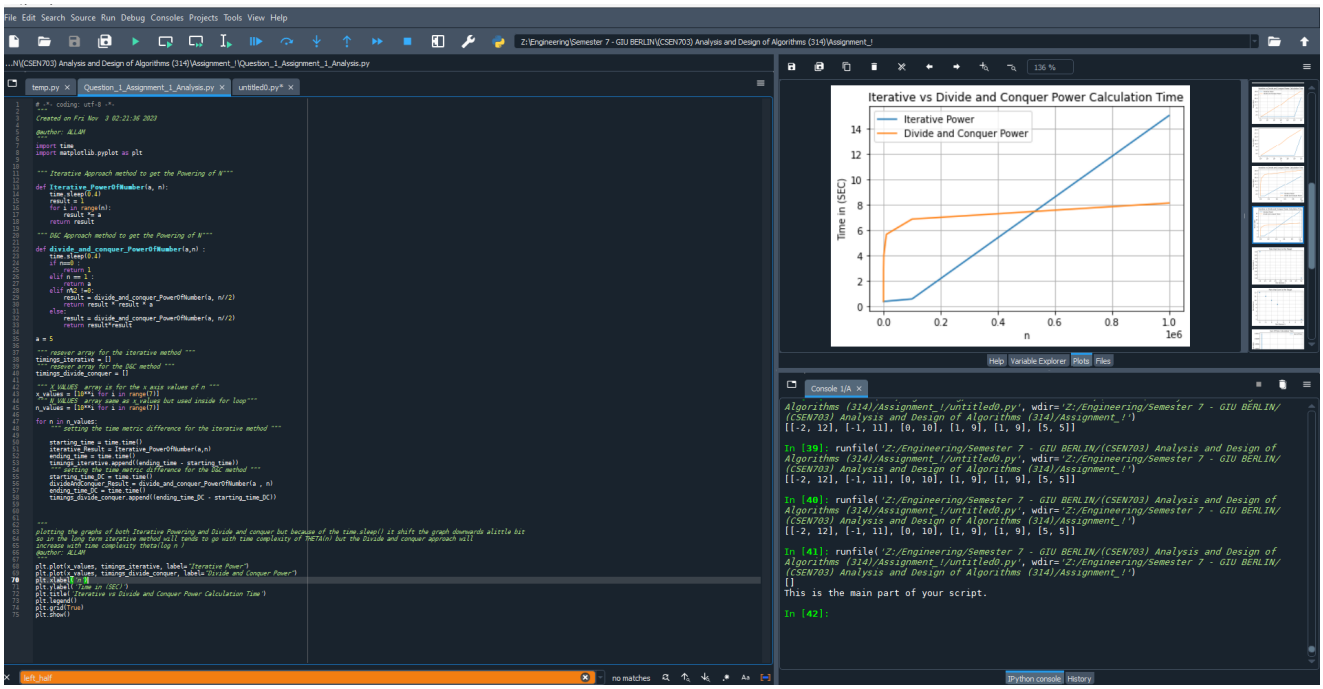
(a)

. Write two program segments to implement the computation of powering a number, an , where n is a natural number, in two different ways:

i. Na¨ıve iterative method through multiplication by n times in a loop.

ii. Repeat the same problem using a divide-and-conquer approach.

**ANSWER**:




(b) Determine each algorithm's asymptotic running time complexity using the big theta nota-
tion. Solve the recurrence for the divide-and-conquer algorithm.

**ANSWER :**

1-Iterative Approach ( Iterative_PowerOfNumer (a,n)) :

The time complexity of this Algorithm is **`θ(n)  Theta of n  as it calculates the power of a number by looping from 1 to n and multiplying the result in each iteration by a`**

2- Divide and Conquer Approach( divide_and_conquer_PowerOfNumber(a,n)) :

By dividing the problem into subproblems

The recurrence relation for this algorithm : Using Master theorem from the lecture
T(n) = aT(n/b) + f(n),
T(n) {

T(n) = **θ(1)** **if n ==1**

**T(n) = T(n/2) + θ(1) + θ(1) otherwise**

}

and substituting using values of a =1 , b=2 , f(n) = **θ(1)**

So by solving this recurrence relation :

**n^log_b(a) = n^log2(1) =1 ,**

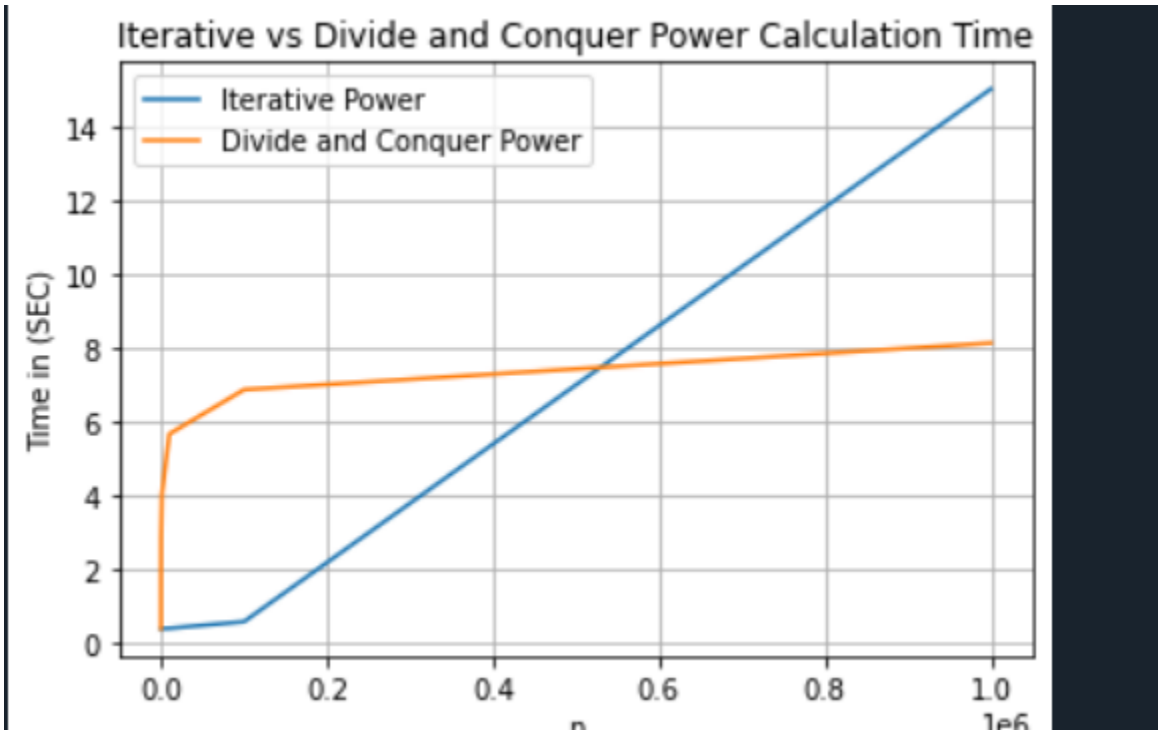
**since f(n) = θ(1),**

**Then**

**T(n) = θ(((n^log_b(a)) * log(n))= θ(((n^log2(1)) * log(n))= θ(((n^0) * log(n)) = θ(log(n))**

Implying that the time complexity for this algorithm will be **θ(log n ).**

(c ) 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




**ANSWER :**

Here we have Iterative Power represented by the blue line and Divide and Conquer power represented in orange color , We can notice that the iterative method has a linear growth rate in time since it depends on the size of input that it receives as it has time complexity **`θ(n),`** but on the other hand we can notice that the Divide and Conquer Method has logarithmic growth rate as it has a time complexity of **`θ(log n ).`**

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

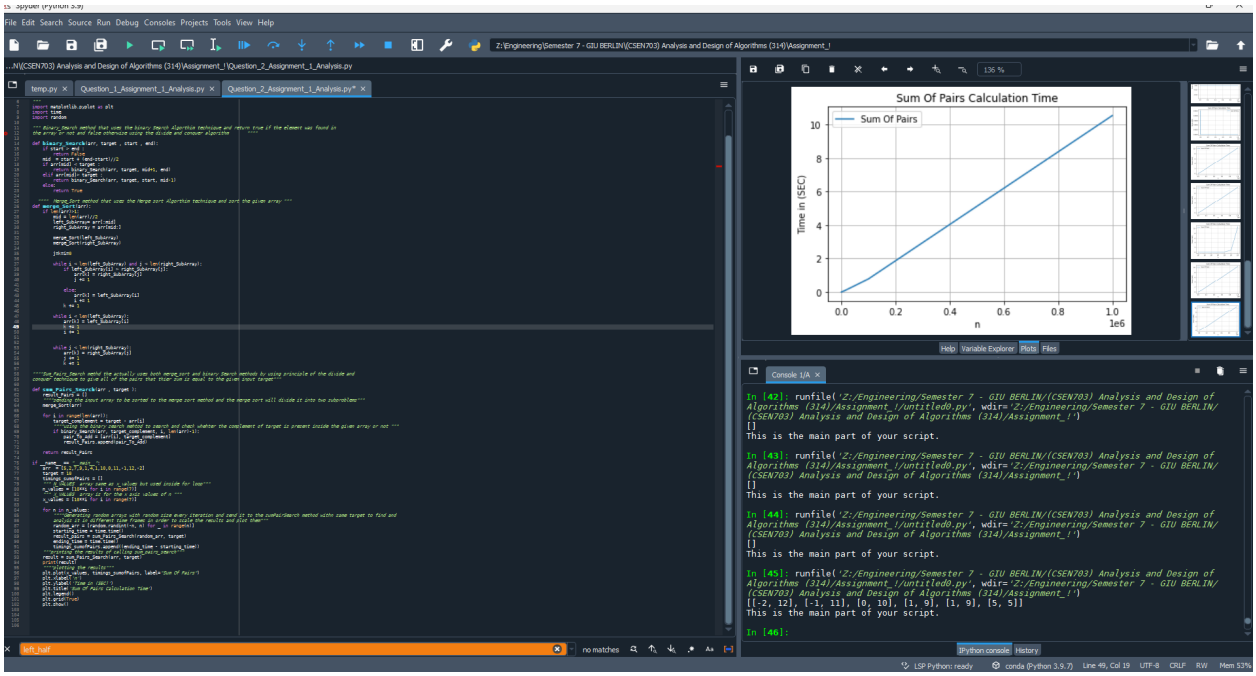
**ANSWER :**

Yes Actually , because the growth rates of the experimental results matches the theoretical analysis results in both methods iterative and divide and conquer as for the Iterative approach the graph shows linear growth and the analysis results shows that it has a time complexity of **`θ(n) , and for the divide and conquer approach the graph shows that it has`** logarithmic growth rate and the analysis shows that it has time complexity of **`θ(log n )which had been proven by the Master Theorem. So it holds for both Algorithms .`**

## Question 2 :

(a)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.

**ANSWER :**



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

**ANSWER:**

Divide and Conquer Approach( sum_Pairs_Search(arr , target )) :

By dividing the problem into subproblems using the merge sort and the binary search algorithms , as first we need to sort the array using the merge sort which divide the problem into two subproblems and after finishing the the sorting , we loop over the resulted sorted array and search for the complemented target using the binary search algorithm and if it is found it return true and save the resulted pair the target and it's compliments in pair and save it in the result

The code hold 3 methods , but the merge sort method affect the code mostly so by analyzing it's recurrence relation and use it as The recurrence relation for this algorithm : Using Master theorem from the lecture T(n) = aT(n/b) + f(n), for merge sort we have

T(n) = **{ T(n) = 0(1) if n ==1**

**T(n) = 2T(n/2) + 0(n) otherwise**

}

and substituting using values of a =2(since we merge sort divide it into 2 subproblems) , b=2 , f(n) = **n**

So by solving this recurrence relation :

**n^log_b(a) = n^log2(2) =n ,**

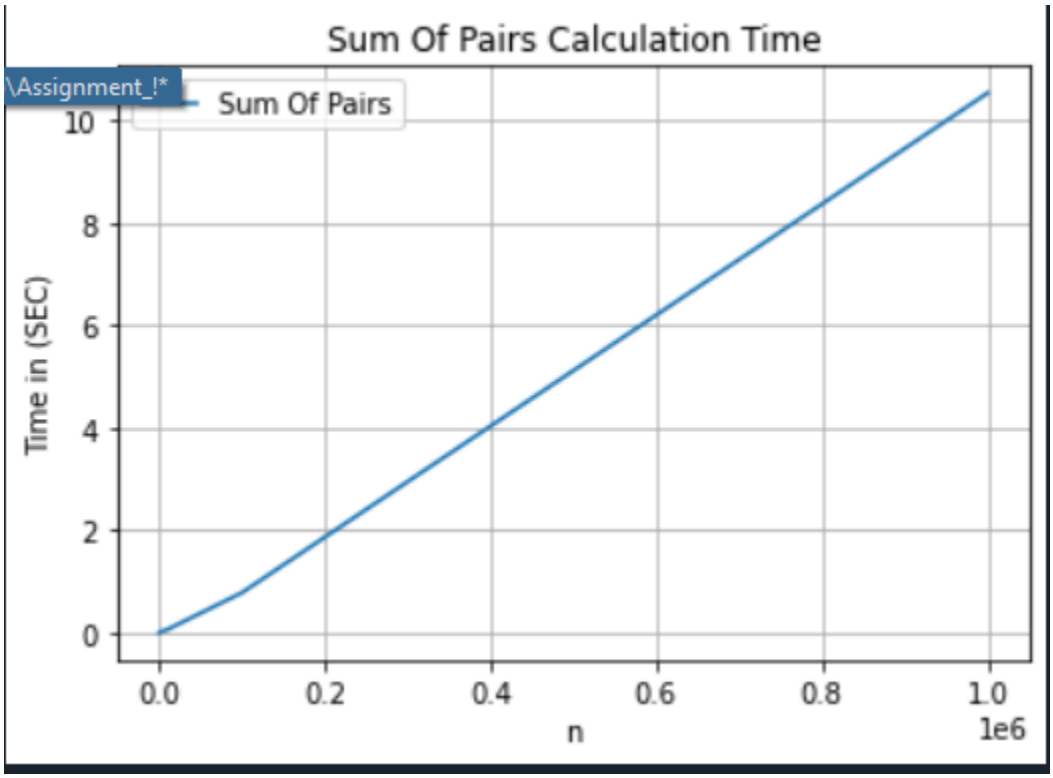
**since f(n) = 0(n) ,**

**Then T(n) = 0(((n^log_b(a)) * log(n))= 0(((n^log_2(2)) * log(n))= 0(((n^1) * log(n))= 0( n log(n))**

Implying that the time complexity for this algorithm will be **0(nlog(n) ).**

(c ) 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.

**ANSWER:**




**But the result may differ because of different process that handled by the processor and it's running time scheduling also i wrote the code in java but converted it totally in python in order to get the running time more accurate**