1b) The recurrence relation is T(n)= T(n)/2=O(1)

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

T(n/2) = T(n/4) + O(1)

...

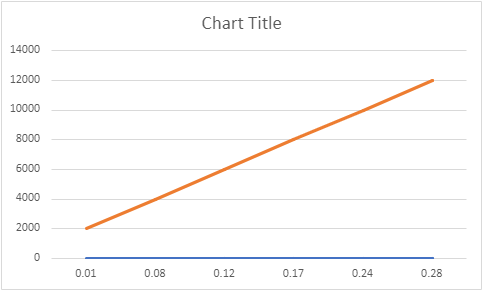
T(n, 1) = O(1)

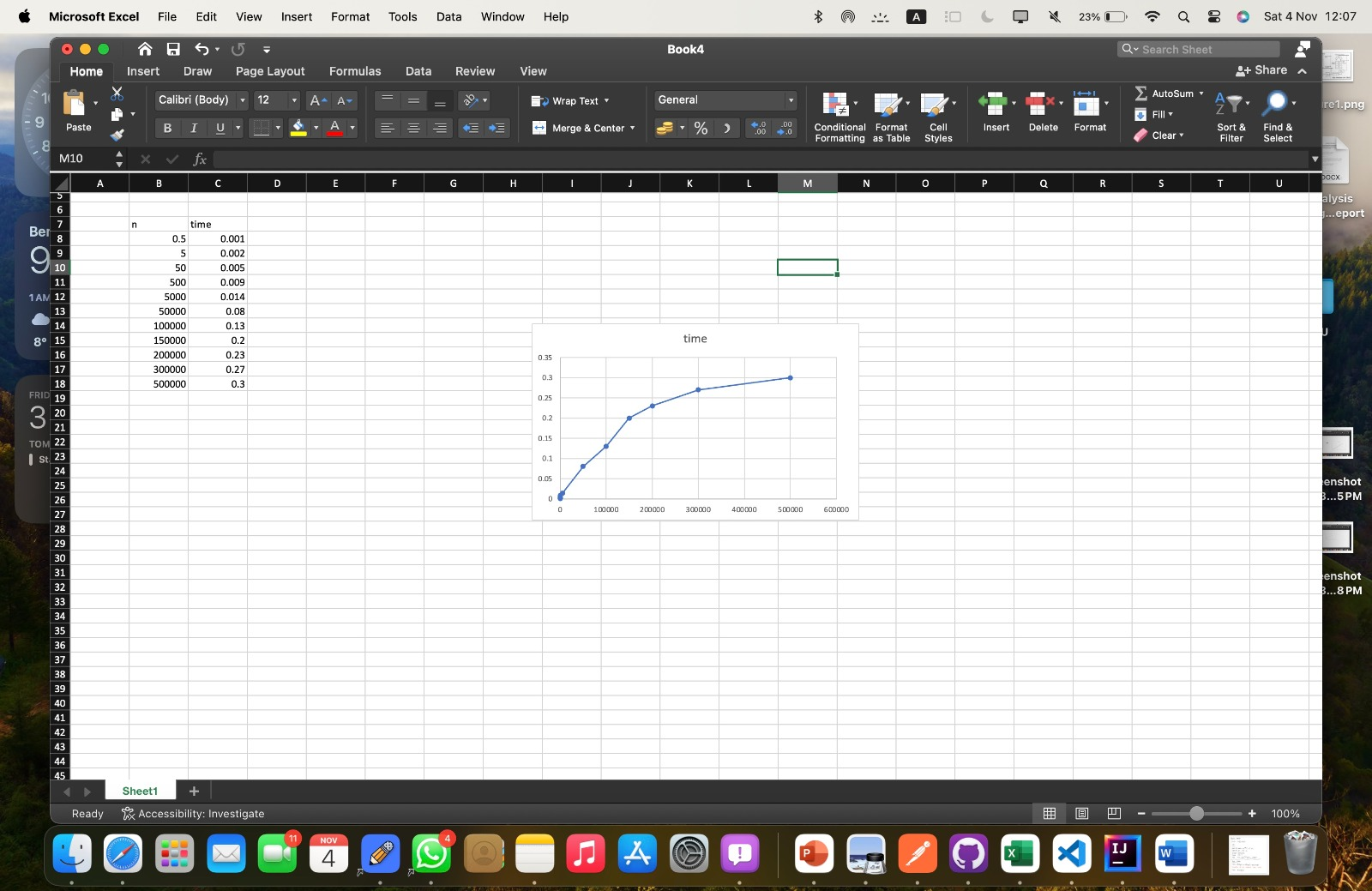
Summing up all these equations, we get:

T(n) = O(1) + O(1) + O(1) + ... + O(1) (log2(n) times)

T(n) = O(log2(n))

So, the asymptotic running time complexity of the `DivideAndConquer` algorithm is Θ(log(n)).

1c)



1d) results from the graph follows the prediction.

2b)1. mergeSort(int[] n):

a = 2, b = 2, and f(n) = O(n). Master Theorem

* Compare f(n) with n^log\_b(a):
* f(n) = O(n)
* n^log\_b(a) = n^log\_2(2) = n
* Compare the two functions:
* Since f(n) is O(n), it is Case 2 of the Master Theorem.

The Master Theorem Case 2 states:

If f(n) is Θ(n^log\_b(a)), where log\_b(a) < 1, then T(n) = Θ(n^log\_b(a) \* log(n))

2. binarySearch(int[] n, int x):

a = 1, b = 2, and f(n) = O(1), then T(n) = Θ(log(n)).

a = 1, b = 2, and f(n) = O(1), so it falls directly into Case 2.

the solution for the Binary Search recurrence relation is:

T\_binary(n) = Θ(log(n))

The Binary Search algorithm has a time complexity of Θ(log(n))

3. findPairs(int[] n, int Sum):

It will be the greater of the 2 recurrence relations which will be o(nlogn) as it is calling the other methods

2c)

