Report: HW 3, Question 2. Nikhita Sagar

Report:

1. Divide & Conquer: The divide and conquer approach divides the array into its left and right components. It recursively returns the maximum sum out of the left, right and crossover subparts.
It also stores the index's of the values in a tuple and thus returns those to be stored in a file as the start and end indexes of the algorithm.
After opening the file, the program splits the lines into arrays and sends those as parameters to the function, If the start and end ~~value~~ indexes are the same, that ie the array has only 1 element, the function returns that element otherwise it returns the maximum of the maximum sum of the left subarray, maximum sum (sum of crossover), thus covering all possible outcomes

Time complexity:

$$T(1) = O(1)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \quad \text{if } n > 1$$

($O(n)$ ← crossover; 2 ↑ substructures (left / right))

according to master theorem : $O(n \log n)$ time
This is just for each individual try of the algorithm.
For K tries in a file the runtime is $O(kn \log n)$

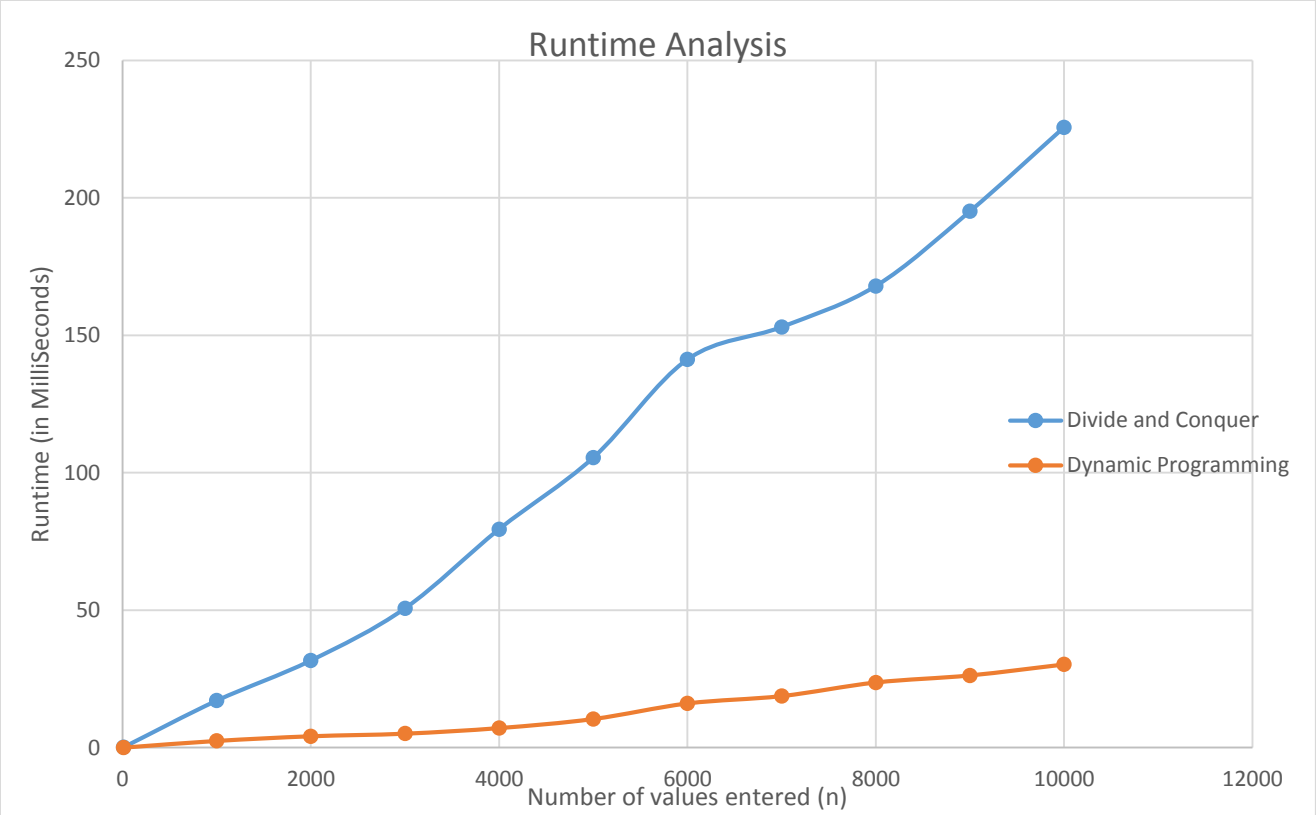
Space Complexity : $O(1)$ since no variable value is stored or memoized.

2. Dynamic Programming: In the dynamic programming bottom up approach, I stored in a new (sum) array the sum of the previous sum and the new element in the array or 0 if the sum was less than 0 then if the maxsum was less than the current sum, it was replaced by the currentsum and the index was stored as the end index.
As before, the file was opened and each line was split as an array.

Time complexity: For the algorithm part of 1 try
$\Rightarrow O(n)$ because there was only 1 for loop iterating over n elements in the array.

Overall program with K tries: $O(k \cdot n)$

Space complexity: $O(n)$ because a sum array with n elements was stored.



As you can see from the graph of the run time averages, the divide and conquer approach takes O(nlogn) time and the Dynamic approach takes O(n) time. All the dynamic solution based tries finished a lot faster than the divide and conquer solutions. This was because the divide and conquer solution splits the array into half and recursively finds the maximum sum in each half. While on the other hand, the dynamic solution stores the sum in a 1D array and iterates only once to find the solution.