INFSCI 2591: Algorithm Design Project 1

1- The results of the test cases.

The results are saved in the file named:
prj1_ques1_result_ot_test_Cases.html

2- The largest number that the recursive algorithm (Algorithm 1.6) can accept as its argument and compute the answer within 60 seconds: **39** (55.26 seconds), 40th element takes 91.60 seconds

3- The time the iterative algorithm (Algorithm 1.7) takes to compute this answer: to compute the 39th element: 1.120 seconds

4 - Other important observations/results not listed above:

Number	Iteration Time (sec)	Recursion Time (sec)
26	9.2	0.095
27	5.7	0.15
28	7.15	0.24
29	1.001	0.43
30	6.19	0.67
31	6.19	1.08
32	7.8	1.79
33	8.82	2.875
34	9.05	4.6
35	9.05	7.6
36	9.28	12.31
37	1.09	19.97
38	9.28	34.79
39	1.001	55.2
40	1.001	91.60

The table shows different time taken by the two algorithms to compute nth- element. Here, we see that upto a certain number, recursion is faster than iterative function. After a point, iterative takes lesser time and recursive function time grows exponentially.

Time complexity: Recursive – O(n)

Iterative: linear with n,

Space Complexity: Recursive – there will be n recursive calls, so n stacks will be used. So the space complexity is n

Iterative: there is only one variable, which stores the final value, so time complexity is O(1)

On running the 4 programs on my two different laptops, 1 mac and other one Dell, we found out that there was a slight variation in time complexity of the two functions. One of the systems was processing slower than another one. This brings us to the conclusion that there are other factors too, which affect the performance of the algorithm.

The final conclusion: Both the divide-and-conquer algorithms are useful depending on the user requirements. The recursive function takes less number of variables and looks efficient. It is faster up till a certain number. Afterwards, the tree grows substantially, due to which it becomes really slow and uses system resources more than iterative function. Whereas, iterative function uses many variables, looks naïve and takes more time than recursive function initially. It becomes much faster after a certain number and much more efficient than recursive function.

Why there is no overflow problem in Python:

Earlier, python used to have a limit on int values. Later they removed that limit by treating numbers as objects. It allocates 32 bits for the value of the object, and as the value increases, the value can go upto the size of RAM.

The integers can overflow if operations are done in pydata stack (numpy /pandas /scipy), because they use C style fixed-precision integers. This problem can be overcome by internally converting int to long type.