Reflection Essay - Assignment 1

Prasann Viswanathan

Abstract: I thoroughly enjoyed solving this assignment although I have close to no experience with competitive programming or solving problems like these, barring CS101 of course. I look forward to doing more challenging problems like this in the future as well.

Note: This document will be in a Question-to-Question blog post format, as such 'What I learnt' will be divided on the basis of what I learnt from each question, and a final summary of what I learnt overall.

Question 1: Stacking up on Quicksort

This question did not strike me at all. Sir had already taught us an implementation that saved stack-space, why delve further? With that opinion in mind, I simply implemented the partition function with help from https://www.geeksforgeeks.org/quick-sort/ - reference and wrote sir's algorithm verbatim and left it at that. Further research on freeing up stack space in quicksort https://www.geeksforgeeks.org/quicksort-tail-call-optimization-reducing-worst-case-space-log-n/ revealed an answer that was so obvious it was staring me in the face. Go to either the left/right partition based on what is smaller in length! Smaller length implies lesser stack calls, saving precious stack memory. In all honesty I didn't come up with this implementation myself and give myself 0 credit. It was all in the link above.

Learning: First I shall answer the questions posed by Sir in the problem statement.

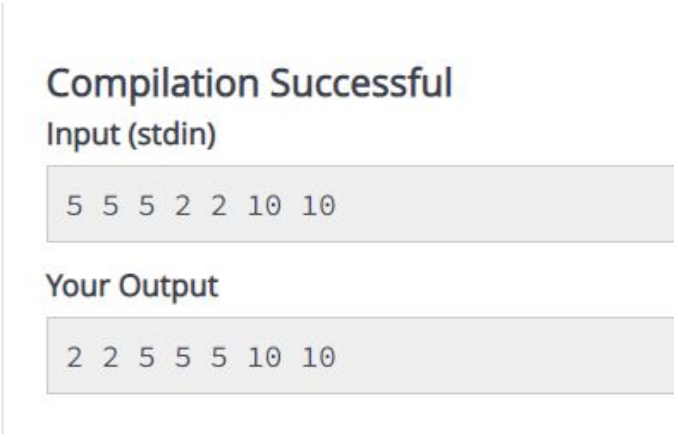
(Why#1) It still consumes stack space as although we got rid of one recursive call, there remains another which in turn consumes stack space.

(Why#2) It consumes stack space proportional to the number of items as when we perform quicksort on the left partition (The one dealt with via stack) the length of the partition is itself proportional to n (It can be upto n-1) as a result the number of times it is called is proportional to n and it consumes stack space proportional to n.

(How much#3) The new implementation (Courtesy geeksforgeeks) consumes stack space of order Log(n) (Because we are choosing the least length partition each time the number of stack calls is at most ceil(log2(n)). (Think of the number of times of halving and choosing the smaller half).

(Why#4) As answered in (How much#3) the worst case stack space is ceil(log2(n)) in this implementation and the explanation has been provided as well.

Output:



Question 2: Go For EASY!!

My approach in this problem was to come up with a recursive formula for F(E, N) which I initially struggled with, as I was heavily reliant on a mathematical expression instead of a programming one. On trying out cases 1, 2, 3, 4 by hand I saw a pattern that I was able to recreate with a simple recursive solution. But unfortunately it wasn't working for test cases >20. I suspected it was because the function calls were piling up exponentially leading to an overflow on the stack. Plus it was ridiculously slow. So I thought of a 'memoization' approach, which wouldn't need exponential stack calls. It would build on previously computed values. Googling the first 7 numbers revealed the fact that they formed a catalan number sequence. On further research I found some handy formulae which would greatly simplify the problem, the main being the one I found on
https://www.geeksforgeeks.org/program-nth-catalan-number/

$$C_0 = 1 \ and \ C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \ for \ n \geq 0;$$

This was very simple to implement, even more so than coming up with the recursive solution and comfortably passed all test cases with some tweaks.

Learning: Recursion can be extremely slow, even if it looks elegant. This problem reminded me of what we did in Fibonacci numbers, how recursion leads to exponential growth of calls to functions.

Output:

Question 3: Iterative Speaking, the Tower of Hanoi

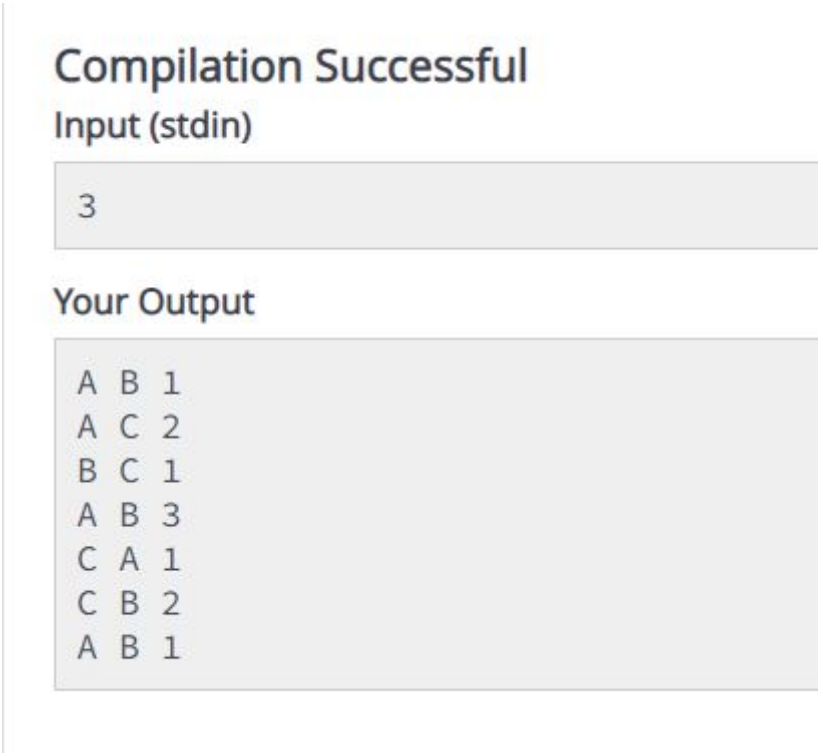
Although sir had essentially given us the solution to this problem in the lectures, I found it a gruelling task to implement due to the sheer number of lines of code needed. Instead I mapped indices of the i-th line to an appropriate pair of alphabets and a number, which I observed from patterns in n : {1, 2, 3, 4, 5, … }.

Every odd index of **i** warranted a 1 to be printed out. Indices 2, 6, 10, ... mapped to 2. 4, 12, … mapped to 3 and so on. This was handled by finding the largest power of 2 to be a common factor to the number and adding 1 to it.

Then observing correctly when the rotation is ‘clockwise’ and ‘anticlockwise’ I handled the cases by implementing a modulus 3 selection process. I ran it and it luckily worked on my first try.

Learning: I did observe readymade solutions on geeks for geeks but didn’t understand them completely, also found them unnecessarily lengthy which is why I didn’t go ahead with them. Although I unnecessarily spent too much time on this question to come up with something ‘elegant’ I learnt that we can think outside the box and don’t need to stick to one way of solving things, even if they’ve been done in class.

Output:



Question 4: Is this Binary Search?

Binary search has been my mortal enemy since CS101.

https://www.geeksforgeeks.org/binary-search/

So I started out with a simple linear search approach by XORing individual elements and proceeding with a simple flag. Enter > Time Limit Exceeded error. Realising I couldn't escape the implementation of Binary Search I implemented a code which simply checks if 'mid' is a local maxima, performing xor operations only when necessary (comparisons). This would save a tonne of time. Also, binary search meant an order of log(n) complexity, no need for sorted list either. Simply search each middle for 'maxima'. The logic was there, the implementation is faulty. With a few hours to go for the deadline I struggle at one test case that refuses to pass no matter what I throw at it.

Update: Needed to add a corner case to deal with edge cases mid=0 and mid=n-1. All test cases passed successfully!

Learning: You have to face your fears, (Binary Search) for me and try to implement them. I am still wary about using Binary Search as almost always it gives some bugs or misses some corner cases, but hey, I guess I am biased. Personally I found this to be the most challenging question amongst the four.

Output: