CS F211
Data Structures and Algorithms
Editorial - 2

Allowed languages: **C**

February 19, 2021

This document contains a *brief* overview and discussions of solutions of problems that were assigned to you in labsheets. This is by no means a comprehensive or complete set of solutions - most questions will only have an explanation of the approach along with possible time complexities, and (for some) pseudocode or even working C solutions. Please note that this is an **informal** editorial made by the TAs. All these editorials, code snippets and pswudocode can be accessed at this GitHub repository which will be updated periodically. Solutions/editorials for a specific labsheet will be provided **7-10 days after** the last lab for that labsheet ends. For the most part, we are hoping that interested students will contribute their solutions to the repository, to benefit everyone else.

Contributing Solutions

- Fork the GitHub repository and add you "*.c" files in the correct directory. (each assignment will have one directory). The filename must be the problem number in the assignment.
- Please try to use descriptive variable names, and test your code with the sample test cases.
- Commit your code and open a pull request. One of the TAs will review your code for readability and merge the changes. For now, we will not be doing any form of intensive checks before merging code.
- Contact any one of the TAs in case you need help with understanding how to start a pull request.

A: Ghot's Riddle

This is more of an implementation problem. The idea is just to call recursive functions to solve it. We prove the complexity of such a solution below: The given recurrence relation is:

$$f(x) = \begin{cases} f(div_1) + f(div_2) & \text{if } x \text{ has two or more distinct divisors other than 1 and itself} \\ x & \text{if } x \text{ has 1 divisor or less apart from 1 and itself} \end{cases}$$

We note that, for any given number, N, the upper bounds for div_1 and div_2 are $N/2$ and $N/3$ respectively. (there are no bigger divisors possible than these two, other than N, which is excluded in the definition). We can identify the divisors of a number in $O(\sqrt{N})$. Therefore, the recurrence relation for this problem is

$$g(N) = g(N/2) + g(N/3) + \sqrt{N} \tag{1}$$

Simplifying this recurrence relation yields an asymptotic bound of $O(N^{0.788})$. Even less efficient solutions (such as those using an $O(N)$ linear search for identifying divisors) would have an overall time complexity of $O(N)$ for this problem. Since $N \approx 10^5$, even these inefficient solutions would not have exceeded the time limit.

B: Tourism

The idea is to perform a Depth-First Search (DFS) from every island independently and see how many islands you can reach from each island. Then just find the maximum of them. The time complexity of each DFS is $O(n+m)$ where n is the number of nodes and m is the number of edges. So the total time complexity is $O(n(n+m))$, which will work with the given constraints.

We could not think of a more optimal solution to this problem without getting into bridges and stuff so if you have a more efficient solution feel free to let us know.

C: Klutz

Here the idea is that every single person plays exactly $N-1$ games, so to decide which two people are in the answer, we can count the number of times each person appears in the input and see which people play only $N-2$ games. Let the two people in the answer be a and b. Now we have to determine the order of a and b. One easy way to check is if there exists some c whose rating lies in between a and b. This will happen when the two lines (c a) and (b c) are both present in the input or if the lines (a c) and (c b) are present in the input. If these two line are present then the order is b a. If the lines (a c) and (c b) are present then the order is a b. If neither of these cases is true then it is impossible to determine the order. To check which case it is, iterate over all possible values of c and check if the corresponding lines exist. Iterating over values of c takes O(N) and for each c we have to check if there are two corresponding lines or not which O(N^2) (because there are $\frac{N(N-1}{2} - 1$ lines in the input). The total complexity of this is O(N^3).

A more optimal solution would be to make this a directed, fully connected graph of order N where each edge represents a match played between two people, and its direction goes from the winner to the loser of that game. This graph will have exactly one edge missing because there is one missing game. Using the counting logic mentioned above you can find the two nodes between which the edge is missing. Let's call them a and b again.

We now need to add an edge between a and b such that it does not contradict previous information given in the input. What does this mean for our graph? Well a contradiction happens when you have a case like ($x > y, y > z, z > x$) where x,y, and z are ratings of three people. In the case of our graph this is basically a cycle (because you would have edges from x to y, y to z, and z to x). So in our graph representation we can see that having a cycle is not allowed. Now all you have to do is direct the edge one way and check if it creates a cycle or not. If it does, then this is not a valid order. If directing it both ways does not produce a cycle then you cannot determine the order. This can be done using dfs twice so the overall complexity is O($N + \frac{N(N-1)}{2} - 1$) which is equal to O(N^2). This solution is more optimal but for the given constraints, even the first solution will pass.

D: Geass Code

For this problem you can easily find out which multiplier hits which knightmare. It is easy to notice that each multiplier hits a contiguous range of knightmares only. So instead of brute forcing it, (which would be O(N^2)), we can do the following:

Maintain an array that will represent the final answer, initially filled with all 1s. Let us call this array ans. Now iterate through the list of multipliers and find the range of knightmares that it will hit. you can do this in O(1). Let the range be l, r where l is the leftmost point of the range and r is the rightmost. Multiply ans_l with $Mult_i$ and divide ans_{r+1} with $Mult_i$. Do this for all i and then go from the beginning of ans to the end, performing the operation $ans_i = ans + i - 1 * ans_i$. This is very similar to the idea of prefix sums, except with multiplication. It is ease to see why this is the correct approach. Now you have your final array ans. just multiply p_i with ans_i to get the

answer for the i^{th} knightmare. The time complexity of this is O(N). Remember to take modulo throughout all operations. If you are not familiar with modulo under division, search it up online (Note that $10^9 + 7$ is a prime number).

E: Who Will Win Today?

Those of you that ended up looking on the internet would have found it but this question is the same as this one and the same answer will work. You still have to job of understanding the logic and converting C++ code to C or writing it on your own.

F: Koro-sensei and the Powers of Two

A brute force approach to solve this problem is by generating all powers of 2 whose values are less than N and then generating all combinations to represent the sum N - this, however, would have exponential time complexity in N. Since the constraints for N are relatively small (≈ 100), this approach probably would work for most testcases. This article explains a solution to this problem in $O((log(N) + K)^K)$, where $K = log_2(N)$.

G: The COVID Vaccine

This problem is a simplified version of the bin packing problem (the constraint that one large box can fit at most two small boxes makes this easier to solve). To solve this problem, begin by sorting the array of sizes of small boxes (*sizes*) in ascending order - since N is sufficiently small (2×10^3), using even an $O(N^2)$ sort would not result in a TLE. The algorithm is explained in 1.

Algorithm 1: Get Minimum Number of Boxes

```
Inputs: B, N, sizes
boxes = 0
sort(sizes, order=ASCENDING)
i=0, j=N
while i ≤ j do
    if i == j then
        boats += 1;
        i += 1;
        j -= 1;
    else
        if sizes[i] + sizes[j] ¿ B then
            j -= 1;
            boats += 1;
        else
            i += 1;
            j -= 1;
            boats += 1;
        end
    end
end
return boxes;
```

The time complexity of this solution would be $O(NlogN)$ if a merge sort like algorithm was used in the sorting step.

H: Aggregating the Binary Tree

Algorithm 2: Aggregated Sum

```
Function AggSum(N, A, i, l)
1   if i ≥ N then
2       return 0
3   return (A[i] × l) + AggSum(N, A, 2*i + 1, l+1) + AggSum(N, A, 2*i + 2, l+1)
```

This problem can be solved in a recursive approach. Start with the 0^{th} element, and initialise $level = 0$. Then visit the $(2i+1)^{th}$ and $(21+2)^{th}$ element, and so on and so forth recursively. Overview of steps shown in Algorithm 2, where i is the current index and l is the current level. The function needs to be initally called as `AggSum(N, A, 0, 0)`.

I: The Dinosaur Conundrum

Create a mapping between the type of dinosaur (indicated by first character of the dinosaur tag) and the number of dinosaurs with that tag. Since there are 36 possible characters in a dinosaur tag (26 alphabets + 10 digits), this mapping can be represented through an array of frequencies (F) of size 36. This process can be done in $O(N)$. Next, iterate through the 36 membered array, F, and for each element:

- if $F[i] \leq C$, we can split all dinosaurs of this type into separate issues.
- if $F[i] > C$, begin by putting $K_i = \lfloor F[i]/C \rfloor$ dinosaurs in each cage. There will be $F[i]\%C$ dinosaurs remaining - put these remaining dinosaurs into $F[i]\%C$ different cages. Some cages will have K_i dinosaurs, while others will have $(k+1)$ dinosaurs. Calculate the number of fights in each cage by counting the number of combinations.

The second step in the above bullet point can be optimized to be done in $O(1)$, so the overall complexity of this solution is $O(N)$.

J. Air Tickets

This problem describes a directed graph, with all edges having equal weights. A Breadth-First-Search with source vertex S and destination T will identify the cheapest path possible. For a graph with V vertices and E edges, BFS is of the order $O(V+E)$. Since the number of edges is on the order of $O(N^2)$ in this problem, the time complexity of a BFS using an adjacency list is $O(N^2)$.