PS1 (Personal difficulty ranking: C > E >> D >>>>>>>>>>>>> B > A)

A – Baby Bites

- This question is quite simple as we only need to make sure the ith element either has the value i or is the word “mumble”, hence linear sweep.

B – Alternating Subsequence

- Since we are trying to find the max valued longest alternating sequence, so we can keep searching for the max value until the sign change and store that value. Every time the sign changes we add the previously stored value to the sum.

C – Jumbled String

- Firstly, I noticed that b and c account for the time that 01 and 10 appears respectively and their patterns are not quite straight forward. However, a and d records how many 00 and 11 are there, where I can deduct the number of 0s and 1s present in the string. Assuming there are n of 0 present. To count the total number of 00 without duplicate, I can let the 1^{st} 0 in the 00 to be the 1^{st} in the total n 0s, then the 2^{nd} 0 in the 00 can be any of the remaining n-1 0s; then let the 2^{nd} 0 to be the 1^{st} 0, I then have n-2 choices for the 2^{nd} 0. This means with n 0s, there would be totally 0 + 1 + 2 + … + n-2 + n-1 pairs of 00. Hence, I can check if a is a triangle number. If yes, then the number of integers that adds up to this triangle number is the number of 0s in the string. Otherwise, such string does not exist. Same logic applies for d.
    - i.e. 10 = 0 + 1 + 2 + 3 + 4, which can be decomposed into 5 integers (including a 0). Hence if there are 10 pairs of 00, then the entire string has 5 of 0s. On the other hand, if a=11 which is saying the entire string has 11 pairs of 00, then it is invalid as 11 is not a triangle number.
- Secondly, another pattern I noticed is that every 0 would either form a 01 or 10 with all the 1s, and it has to form one of the two pairs. Hence if every 0 would form either a pair of 01 or 10 with every 1, then the product of the number of 0s and the number of 1s should be the sum of the number of 01s and the number of 10s, which is b+c.
    - i.e. a=10, b=18, c=7, d=10 would be valid since from the previous part, we know that a=10 and d=10 means there are 5 of 0s and 5 of 1s in the string. 5 * 5 = 25 = b + c. Hence it is valid.
- The two conditions above can determine if the given a, b, c, d can form a valid string. The way I compute the actual string is inspired by the previous pattern I found. I later realized that If I place all the 0s at the left then place all the 1s at the right would result in 0 pairs of 10 and b+c pairs of 01. Then if I flip a pair of adjacent 01, the number of 01 would decrease by 1 and the number of 10 would increase by 1. Or in other words, I can describe this operation as shifting the leftmost 1 that is not more left than all the 0s towards the left by 1 digit. Now I simply needs to carry out this operation c times to have the desired string.
    - Example using a=10, b=18, c=7, d=10:
        - 0000011111 has 25 pairs of 01 and 0 pairs of 10
            - Shift left most 1 towards the left
        - 0000101111 has 24 pairs of 01 and 1 pair of 10
        - 0001001111 has 23 pairs of 01 and 2 pairs of 10
        - 0010001111 has 22 pairs of 01 and 3 pairs of 10
        - 0100001111 has 21 pairs of 01 and 4 pairs of 10
        - 1000001111 has 20 pairs of 01 and 5 pairs of 10

                - Now there is no more 0 on the leftmost 1, so we start shifting the 2nd leftmost 1
            - 1000010111 has 19 pairs of 01 and 6 pairs of 10
            - 1000100111 has 18 pairs of 01 and 7 pairs of 10
                - Which is our solution.
- Now as we can see, iteratively shifting the digits one by one would be extremely slow when the string gets long. An optimisation would be to calculate the position of 0s and 1s beforehand then print the string all at once. In the previous examples, we can see that some 1s are moved to the left-most position of the string. We can calculate the number of 1s that is moved to the left-most position of the string by doing c/number of 0s. The reminder, which can be found using modulo is the number of digits we need to shift the next 1 towards left, which is the number of 0s between the 1s on the right end of the string and the middle 1. Then we can use the total number of 0s to subtract it to find out the number of 0s between the left most 1s and the middle 1.
    - Example using a=10, b=18, c=7, d=10:
        - We know there should be 5 of 0s and 5 of 1s.
        - c/number of 0s = 1, so there the number of 1s on the left-most end of the string is 1
        - c % number of 0s = 2, so there are 2 of 0s between the right most 1s to another 1 in the middle
        - number of 1s = 5, 5 – 1 (left most 1) – 1 (another 1 in the middle) = 3, so there should be 3 of 1s on the right end of the string
        - number of 0s = 5, 5 – 2 (number of 0s between the right most 1s and the 1 in the middle) = 3, so there should be 2 of 1s between the left most 1s and the 1 in the middle
        - Hence the final string is 1000100111, obtained by carrying:
            - Print 1x1
            - Print 3x0
            - Print 1x1
            - Print 2x0
            - Print 3x1
- This process can be further simplified using a while loop.
    - We can first start to print 1 as the head of the string. For every 1 we print, we create some pairs of 10 that is equals to the number of 0 in the string. Hence every time we print a 1, we subtract the number of 0s in the string from c until c (the number of 10 pairs) is less than the number of 0s in the string.
    - Then we keep print 0s until the number of 0s remaining is the same as the number of 10 pairs remaining.
    - Then we print a 1
    - Then we print out all the remaining 0s which each forms a 10 pair with the 1 we printed in the previous step.
    - Then we print out all the remaining 1s.
- (And my code is still taking 300ms to run which idk why ;-;)

D – Riverside Curio

- I initially tried to find a pattern for the total number of lines and the number of lines under the water, then I realized that would be quite impossible as the total number of lines on a certain day relies on potential future input

- I then realized I can first collect all the inputs and put them in the array. After knowing all the input values, I can then start from the very last input and fix the array backwards. Hence I first collect the number of lines above the water level every day and put them into an array, then the number of lines in total would be the higher one between the number of lines in the previous day or the number of lines above water + 1 as the number of lines in total cannot decrease, and I store these values into another array. After collecting these 2 arrays, I work from the last day, where if the difference between the total number of lines on day i and day i-1 is greater than 1, then the number of lines in total on day i-1 would be (lines on day i) – 1 since there can only be 1 more lines than the previous day. Then I subtract the number of lines above water on day i to calculate the number of lines below water.

E – The Fair Nut and Strings

- First I thought about the process of building the string as a tree. Where at every level I can choose the letter 'a' or 'b', hence starting from an empty string, I can choose to add an 'a' or 'b' to be the next character, and recursively builds up a tree.
- Then to choose k strings, I simply need to choose k leaf nodes which gives me the most prefix. However, I was stuck on how to pick the string which gives me the most prefix.
- And Hugh gave me a hint as we were discussing, he told me to think about whether "aaba" and "aaaa" would have more distinct prefixes together or "abaa" and "aaaa" would have more distinct prefixes together. The answer is "abaa" as its first letter of difference with "aaaa" comes before the first letter of difference between "aaba" and "aaaa". So I would choose k strings where their first letter of difference with string s comes as early as possible. However, here comes the second time I struggled on this question, how can I find which string has the earliest letter of difference with string s?
- Then the second person that gave me a hint was Martin, a friend of mine who did the course last year. He saw me doing this question and said he did this questions by "computing the number of prefixes mathematically without actually building the string using maths like I sometimes times by two or whatever."
    - The first half of his sentence gave me a hint that the ultimate goal of this question is to find the total number of distinct prefixes, and my process of building the tree is essentially computing the prefixes. If I want the strings to have their first letter of difference as early as possible, then I want their prefixes also to be different as early as possible, which interpreting using a tree, I want the tree branch to diverge as early as possible. Hence at every level I would try to include as many nodes as possible, if the number of nodes is less than k then I include all the nodes, otherwise I include k nodes. The total number of nodes I include is the total number of prefixes I have.
    - The second half of his sentence gave me the hint that I don't need to actually build the string, but instead just calculate the number of prefixes mathematically. If every prefix can either add an "a" or a "b" as the next letter to create a new string, then the number of nodes at every level would be double the previous level. However, if the ith letter of string s is "b", then the smallest prefix on the previous level cannot append an "a" to its end as that would be smaller than string s. Similarly if the ith letter of string t is "a", then the largest prefix on the previous level cannot append a "b" to its end as that would be greater than string t. Then I initialize the number of prefixes to be 1, and every level double the number of prefixes, subtract 1 if the ith letter of s is b, subtract another 1 if the ith letter of t is a, repeat this process n times.