PS8

A - Set Difference

- For this question, I noticed that every element S_i in the set S, it can have two states when it's in a subset, being either present or absent. Since each item in the set can have 2 states and there are totally n items, so there would be 2^n subsets in total.
- For each subset, since we only care about the maximum item and the minimum item in the set, we can then instead convert the problem into "find the number of subsets with S_i being the maximum element and S_j being the minimum element". Then multiply the number of subsets with the difference between S_i and S_j.
- To find the number of such subsets, we can first sort the array, then if we want S_i to be the maximum element and S_j to be the minimum element, we should only consider the array [S_i : S_j] since everything before S_i would be greater than S_i and everything after S_j would be less than S_j.
- Every element between S_i and S_j can be either absent or present, and there are totally j-(i+1) elements in between, so there would be totally 2^(j-i-1) subarrays in total, we then take mod using the function provided in the lecture slide, multiply this value with S_i-S_j, and run a nested for loop for every i and j for start and ending point, sum up all the values would be the final solution.
- HOWEVER, this algorithm gave me wrong answer and I could not figure out why, so I changed another approach still using similar logic. We can rewrite the sum we compute as:

$$\sum \max(s) - \min(s) = \sum \max(s) - \sum \min(s)$$

- Instead of fixing both endpoints, we can fix one endpoints at a time and compute both sums. We still sort the array in descending order, and find how many arrays are there if we want S_i to be the maximum element, that would simply be 2^N-i-1 as all values after S_i can be absent or present, then we multiply this with S_i, and repeat for every i and sum them up to find the sum of maximum element for all subarrays, then similarly find sum of minimum element for all subarrays, then find the difference.
- HOWEVER, this code still did not work and I still couldn't see how. Hence I wrote a python debug script which compares my program's output with the bruteforce algorithm's output (code ss attatched at the end).
    - I then found out that for the case "1 2 100000000 1000000000" my code would return "-1000000007" instead of "900000000". This is due to I took mod of sum_of_max and sum_of_min separately, and doing this would cause sum_of_max being less than sum_of_min sometimes. Hence I added another modulo value which is 1000000007 to sum_of_max before subtracting, then take mod of the final solution again, which gives me the correct answer.

B – Math

- This question was quite a lot easier than A, as when my friend and I both got stuck on A, he said "I did B first cuz it's so much easier, just easy primes." This sentence gave me some hint which let me think about using prime number as the solution.
- The question requires to output both the min value and the number of steps, I first realized the min value would simply be the product of all prime factors, since I can keep multiplying the value with its prime factors until the frequency of all prime factors are power of 2, then I can keep taking square root to get the minimum value

- The next issue is to find out how many operations are required. I then realized I can multiply the value by an arbitrary value just once, where the arbitrary value would make all the prime factors appear the same number of times which equal to the greatest power of 2 that is greater than the most frequent prime factor.
    - Hence my first thought would be find the power of 2, that would be the number of square root operations I need to apply, then add 1 which would be the multiplication operation.
    - However, that did work out as for the case "1", my code would return "1 1" where the intended solution is "1 0" as no multiplication is required. Multiplication is only required when the frequency of some prime factor is less than the greatest power of 2.
    - Implementation wise, I used map to account for the frequency of each factor.

C – Magic Gems

- This follows exactly the linear recurrence on Maths lecture slide Page 42
- Recurrence being:
    - $f(n) = 1 \times f(n-1) + 0 \times f(n-2) + \cdots + 0 \times f(n-m+1) + 1 \times f(n-m)$
    - $f(i) = 1$ for all $i \in [0, m)$
        - i.e. $f(m) = f(m-1) + f(0) = 2$
- Then substitute values into the matrix then compute for f(n).

D – Two Divisors

- My initial attempt is to use the prime factorization function provided in lecture slide, find all prime factors, and run a double nested loop to check if any pair of factors meet the criteria.
    - However, this approach is flawed as the prime factors are not all factors, it is possible that there exists no such pairs in prime factors but in all factors
    - The case the broke my code was 7817670. The prime factorization is 2x3x3x5x7x12409. There exists no such pair that satisfies the criteria, but such pair does exist for all factors, such as 10 and 9, where gcd(19, 7817670) = 1.
- After knowing that 7817670 is the case that broke my code, while I was trying to figure out what could be a pair of prime numbers a, b that satisfy such criteria which my code failed to capture, I tried out lots of combinations, and realized that "if a+b is coprime with x, then a and b are coprime". This can be proved simply with contradiction
    - assume a+b are coprime with x but a and b are not coprime, this means there exists c such that pc=a and qc=b, where p and q are factors of x, hence a+b = c(p+q), and c is a common divisor of a+b and x, hence contradiction
- This immediately shrinks our search range, we want to search among values a, b which are factors of x and are coprime of each other. Out these pairs of a, b which are coprime, we can impose another condition on them where as if ab=x, then gcd(a+b, x) must be coprime.
- Optimization was the biggest issue I encountered
    - I initially factorized the number into prime factors, then picked the first prime factor, multiple all duplicates of it together, then multiply the remaining factors together, but this approach exceeded the time limit
    - Then I did a linear sweep to find the first prime factor of x, and stop immediately, divide x by this factor while modulo is 0, still not fast enough
    - I then pre-computed all the prime numbers, early stop if x is prime, still too slow
    - Finally, I sweep through all prime numbers to find the first prime factor along with all the strategies I had above.

8A debug script:

```python
import random
import subprocess

def gen_input():
    s = ""
    #T = random.randint(1, 1)
    T = 1
    s += f"{str(T)}\n"
    for _ in range(T):
        n = random.randint(1, 3)
        s += f"{n}\n"
        for _ in range(n):
            s += f"{str(random.randint(1, 1000000000))} "
        s += '\n'
    return s.rstrip()

def brute_force(set_nums):
    i = 0
    inputs = [int(item) for item in set_nums.split()]
    solution = ""

    T = int(inputs[i])
    i += 1
    for _ in range(T):
        s = []
        n = inputs[i]
        i += 1
        for _ in range(n):
            s.append(inputs[i])
            i += 1
        subsets = [[]]
        for element in s:
            temps = []
            for subset in subsets:
                temp = subset.copy()
                temp.append(element)
                temps.append(temp)
            for temp in temps:
                subsets.append(temp)
        subsets.sort()
        res = 0
        for subset in subsets:
            if(subset == []):
                continue
            res += max(subset) - min(subset)
        solution += f"{str(res)}\n"
    return solution.rstrip()

MAX_ITER = 100
EXE = "./A"

if __name__ == "__main__":
    for i in range(MAX_ITER):
        s = gen_input()
        got = subprocess.run(EXE, input=s, capture_output=True, text=True).stdout.strip()
        ans = brute_force(s)
        if got != ans:
            print(f"Failed\n")
            print(f"Input: \n{s}\n")
            print(f"Expected: \n{ans}\n")
            print(f"Got: \n{got}")
            exit(1)
        else:
            print("Passed")
```

Activate
Go to Setti