Solution 4

Group A

1) Problem 1

   Here we claim: for n people entered the room in order, the probability of the j_{th} people can actually sit on their own seat is $\frac{n-j+1}{n-j+2}$.

   One each proof is from [1]. It is easy to prove by induction that before the j_{th} people entered the room, all the seat from number 2 to number $j-1$ has already been taken. Therefore, there is one more seat being take among the remaining $n-j+2$ available seats. Thus, the j_{th} people sitting in their own seat is $\frac{n-j+1}{n-j+2}$.

   From the above claim, we know the second last people sit in his/her own seat with probability $\frac{2}{3}$ and the last people sit in his/her own seat with probability $\frac{1}{2}$. Since the second last people sit in his/her own seat doesn't provide any additional information about the remaining seats, thus the probability that last two people can take their own seats is $\frac{2}{3} \times \frac{1}{2} = \frac{1}{3}$.

2) Problem 2

   First let 5 girls choose their first date, there can be 5! arrangements. Now assume the i-th girl choose the i-th day. Then the first girl has 4 choices, assume she choose 2, then the second girl has 3 choices, assume she choose 3, then the third girl has 2 choices, assume she chooses 4, then the last two girls have only 1 choices together. Total is $5! * 4 * 3 * 2/2$, we divide by 2 because we calculate every unique result twice. The answer is

   $$1440.$$

3) Problem 3 Consider 7 girls A, B, C, D, E, F, G, starting from A, WLOG, say we have AB and AC:

   - Case I: if we have BC, then D, E, F, G have to form a 4-cycle (e.g. DE, EF, FG, GD). In this case, we have in total $7! \times C_7^3 \times 3 = 529200$.
   - Case II: if we do not have BC, again WLOG, say we have BD, let us consider
     - Subcase (i): if we also have CD, then A, B, C, D form a 4-cycle. This is already covered in Case I.
     - Subcase (ii): if not, again WLOG, say we have CE, then we cannot have DE otherwise there is no arrangement satisfying the requirements. Up to this point, it is easy to see that the only possibility would be A, B, C, D, E, F, G form a 7-cycle.

     In this case, we have in total $7! \times C_6^2 \times 4 \times 3 \times 2 = 1814400$.

   Enventually, we have $529200 + 1814400 = 2343600$ different possibilities.

4) Problem 5 Only N=1 makes W_t^N a martingale. Use Ito's Lemma, when $N \geq 2$, W_t^N has a non zero drift term, not martingale.

5) Problem 6

$$\begin{aligned} E[\frac{1}{S_T}] &= E[\frac{1}{S_0}e^{-(\mu-\frac{\sigma^2}{2})T-\sigma W_T}] \\ &= \frac{1}{S_0}e^{-(\mu T-\frac{\sigma^2}{2}T)}E[e^{-\sigma W_T}] \\ &= \frac{1}{S_0}e^{-(\mu T-\frac{\sigma^2}{2}T)}e^{\frac{\sigma^2}{2}T} \\ &= \frac{1}{S_0}e^{-\mu T+\sigma^2 T} \end{aligned}$$

6) Problem 7

An algorithm for solving a linear evolutionary partial differential equation is stable if the total variation of the numerical solution at a fixed time remains bounded as the step size goes to zero. The Lax equivalence theorem states that an algorithm converges if it is consistent and stable (in this sense). Stability is sometimes achieved by including numerical diffusion. Numerical diffusion is a mathematical term which ensures that roundoff and other errors in the calculation get spread out and do not add up to cause the calculation to "blow up". Von Neumann stability analysis is a commonly used procedure for the stability analysis of finite difference schemes as applied to linear partial differential equations. These results do not hold for nonlinear PDEs, where a general, consistent definition of stability is complicated by many properties absent in linear equations.

7) Problem 8

$$\text{success rate} = \frac{\text{\# of successful shoot}}{\text{\# of total shoot}} = \frac{S}{T}$$

Let $S_0/T_0 < 0.5$ and for some $n \geq 0$, the first time we have $S_{n+1}/T_{n+1} > 0.5$. Consider S_n and T_n, since $S_n/T_n \leq 0.5$, we must have $S_n = S_{n+1} - 1$ and of course $T_n = T_{n+1} - 1$. Having this, we can estimate

$$S_n = S_{n+1} - 1 > \frac{T_{n+1}}{2} - 1 = \frac{T_n}{2} - \frac{1}{2}.$$

Since both S_n and T_n are integers, we know

$$S_n \geq \left\lceil \frac{T_n}{2} \right\rceil \geq \frac{T_n}{2}.$$

So, the only possibility is $S_n/T_n = 0.5$.

8) Problem 9

In the actual trading, the decision of shutting down a strategy may depend on various different metrics, like alpha, beta, sharpe ratio, etc. Here we formulate the problem as, given two time series of performance metrics, testing whether one series is significantly better (not necessarily larger) than the other. The definition of "better" depends on the actual performance metrics we choose. Here we use P& L as an example. Since we have paired P& L time series (i.e., the ith elements from two time series of P& L are from the same day), we should take advantage of the pairing information when evaluating two strategies. We can process in two directions:

First, since we have paired P& L, we take the daily distance between the (scaled if needed) P& L of the two strategies. Here distance can be nave difference, L1 norm, L2 norm, Lp norm or other distance measures. We can then use parametric or nonparametric test to see whether the distance time series is a stationary time series.

Second, we can fit a common model that would reasonably describe each series separately. This might be an ARIMA model or a multiply-trended Regression Model with possible Level Shifts or a composite model integrating both memory (ARIMA) and dummy variables. This common model could be estimated globally and separately for each of the two series and then one could construct an F test to test the hypothesis of a common set of parameters.

Reference: [2] [3]

9) Problem 10

Key Point: Converting to barycentric coordinates

- point (p1.x, p1.y);
- point (p2.x, p2.y);
- point (p3.x, p3.y);

check whether (p.x,p.y) is in the triangle.

- double alpha = ((p2.y - p3.y)*(p.x - p3.x) + (p3.x - p2.x)*(p.y - p3.y)) / ((p2.y - p3.y)*(p1.x - p3.x) + (p3.x - p2.x)*(p1.y - p3.y));
- double beta = ((p3.y - p1.y)*(p.x - p3.x) + (p1.x - p3.x)*(p.y - p3.y)) / ((p2.y - p3.y)*(p1.x - p3.x) + (p3.x - p2.x)*(p1.y - p3.y));
- double gamma = 1.0f - alpha - beta;

All alpha, beta and gamma large than zero, then the point is in the triangle.

10) Problem 11

Here is my solution for LeetCode problem 125:

```
class Solution {
public:
    bool isPalindrome(string s) {
        string temp = "";
        for (int i = 0; i < s.size(); i++) {
            if ((s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <= 'Z')
            || (s[i] >= '0' && s[i] <= '9')) {
                temp += s[i];
            }
        }
        transform(temp.begin(), temp.end(), temp.begin(), ::tolower);
        int pointer1 = 0;
        int pointer2 = temp.size() - 1;
        while (pointer1 < pointer2) {
            if (temp[pointer1] == temp[pointer2]) {
```

```
                pointer1++;
                pointer2--;
            } else {
                return false;
            }
        }
        return true;
    }
};
```

11) Problem 12

```
int lengthOfLDS(vector<int>& nums) {
    vector<int> ladder(1);    //Build a ladder.
    if(nums.empty()) return 0;
    ladder[0]=nums[0];
    for(int i=1; i<nums.size(); ++i){
        int m=int(ladder.size());
        bool foundless=false;
        for(int j=m-1;j>=0;--j){
            if(nums[i] < ladder[j]){
                if(j+1==ladder.size()){
                    ladder.push_back(nums[i]);
                }
                else{
                    ladder[j+1]=min(ladder[j+1],nums[i]);
                }
                foundless=true;
                break;
            }
        }
        if(!foundless) ladder[0]=min(ladder[0],nums[i]);
    }
    return ladder.size();
}
```

12) Problem 13

```
#include <iostream>
#include <vector>
using namespace std;

class Solution{
    bool IsValid(vector<vector<char>>& broad, int x, int y){
        //  Check columes!
        for (int i=0; i<9; i++) {
            if ((i != x) && (broad[i][y]==broad[x][y])) {
                return false;
            }
```

```
        }

        //  Check rows!
        for (int j=0; j<9; j++) {
            if ((j != y) && (broad[x][j]==broad[x][y])) {
                return false;
            }
        }

        //  Check 3x3 box!
        for (int i=3*(x/3); i<3*(x/3+1); i++) {
            for (int j=3*(y/3); j<3*(y/3+1); j++) {
                if ((i != x) && (j != y) && (broad[i][j]==broad[x][y])) {
                    return false;
                }
            }
        }

        return true;
    }
public:
    Solution(){}
    ~ Solution(){}
    bool SudokuSolver(vector<vector<char>>& broad){
        for (int i=0; i<9; i++) {
            for (int j=0; j<9; j++) {
                //  Is the Sudoku still unsolved? Find a vacancy!!!
                if (broad[i][j]=='.') {
                    //  Try to fill in 1-9 into vacancy!!!
                    for (int k=0; k<9; k++) {
                        broad[i][j]='1'+k;
                        //  DFS!!! since we use SudokuSolver()!!!
                        if (IsValid(broad, i, j && SudokuSolver(broad))) {
                            return true;
                        }
                        broad[i][j]='.';
                    }
                }
                //  Cannot solve
                return false;
            }
        }
        //  If NO vacancy found=>already solved!!!
        return true;
    }
};
```

13) Problem 14

This is a typical dynamic programming problem [4].

First of all we have to find a state. The first thing that must be observed is that there are at most 2 ways we can come to a cell from the left (if its not situated on the first column) and from the top (if its not situated on the most upper row). Thus to find the best solution for that cell, we have to have already found the best solutions for all of the cells from which we can arrive to the current cell.

From above, a recurrent relation can be easily obtained: S[i][j]=A[i][j] + max(S[i-1][j], if i¿0 ; S[i][j-1], if j¿0) (where i represents the row and j the column of the table , its left-upper corner having coordinates 0,0 ; and A[i][j] being the number of apples situated in celli,j). S[i][j] must be calculated by going first from left to right in each row and process the rows from top to bottom, or by going first from top to bottom in each column and process the columns from left to right.

Pseudocode:

for i = 0 to N - 1 **do**

for j = 0 to M - 1 **do** S[i][j] = A[i][j] + max(S[i][j-1], if j ¿ 0; S[i-1][j], if i ¿ 0; 0)

end for

end for

Output S[n-1][m-1]

REFERENCES

[1] Yared Nigussie, Finding Your Seat Versus Tossing a Coin, The American Mathematical Monthly, Vol. 121, No. 6 (JuneJuly), pp. 545-546

[2] Harvey, C. R., & Liu, Y. (2014). Evaluating trading strategies. Available at SSRN 2474755.

[3] Hamilton, J. Time series analysis.

[4] https://www.topcoder.com/community/data-science/data-science-tutorials/dynamic-programming-from-novice-to-advanced/