# Problem A. Advanced Monopoly

## 1 Optimal Strategy

From the profit formula, we observe that each node's contribution is independent. The optimal strategy for both players is to select nodes in descending order of:

$$\text{Priority}(x) = \text{size}[x] - \text{depth}[x] - w_x$$

where:

- size$[x]$: Number of nodes in the subtree rooted at $x$
- depth$[x]$: Depth of node $x$ in the tree (the root has depth 0)

## 2 Solution Approach

1. Precompute for each node $x$:
   - Subtree size size$[x]$
   - Depth depth$[x]$
   - Priority value Priority$(x)$
2. Sort all nodes by Priority$(x)$ in descending order
3. Alternate selecting nodes between players according to this sorted order
4. Calculate the first player's total profit by summing:

$$\sum_{x \in \text{First player's nodes}} (\text{size}[x] - \text{depth}[x] - w_x)$$

# Problem B. Big Staircase

## 3 Determining whether restoration is possible

For the staircase to be restorable, the following conditions must hold:

- The fixed $K$ steps must form an increasing sequence.
- The height difference between two consecutive fixed steps must be greater than or equal to the distance between them: $h_{k_{i+1}} - h_{k_i} \geq k_{i+1} - k_i$. If this inequality does not hold, it is impossible to create an increasing sequence between the fixed steps.
- The height of the first fixed step must be at least its position minus one. This is necessary so that the segment from the beginning to the first fixed step can be built without making some step have negative height.

If any of the above conditions is not satisfied, the staircase cannot be restored.

# 4 Obtaining the minimum number of steps to restore

For a step to be left unchanged, its height must lie between the heights of the two fixed steps to its left and right. However, this condition is not sufficient: in addition, its height must allow the heights of the intermediate steps between it and the fixed steps surrounding it to be *adjusted*.

This observation allows us to keep the steps that are suitable to remain unrestored and discard all the others.

After executing this code, all unsuitable steps will have height $-1$, while suitable ones will retain their initial height.

Note that if we want to choose some of the suitable steps, then the chosen steps must themselves form an increasing sequence. Thus, after the previous observation, the problem reduces to finding the longest increasing subsequence among the new heights.

There are non-optimal ways to choose the longest increasing subsequence; the subtask descriptions explain in detail methods that correctly choose the longest sequence.

# 5 Subtask 1, $N \leq 20$ (23 points)

The constraints allow us to try all possible choices of restoring or not restoring each step using a 20-bit mask.

# 6 Subtask 2, $N \leq 1000$ (37 points)

In subtask 2, the constraints allow computing the longest increasing subsequence using a quadratic algorithm.

One way is to store, for each suitable step, the longest increasing chain that can be formed ending at it. If this length is stored in $maxlis[i]$, then one way to compute it is:

- For each suitable step $i$...
- $maxlis[i] = \max(maxlis[j]) + 1$ for all $j < i$ that are suitable steps and for which the heights and distances allow $i$ to follow $j$ in an increasing sequence.

To make reconstruction of the answer possible, it is also necessary to store a pointer to the previous step.

# 7 Subtask 3 (40 points)

To solve subtask 3, a method for computing the longest increasing subsequence in logarithmic time is required. There are several algorithms for this operation, but they do not take into account the additional constraint of this problem: besides being increasing, the height difference between two steps must be greater than or equal to the distance between them.

One way to eliminate the distance variable between steps is to *normalize* the heights.

From the observation, it was concluded that two steps $j$ and $i$ with $j < i$ can belong to the same staircase if $a[i] - a[j] \geq i - j$. Transforming the inequality, we get:

$a[i] - a[j] \geq i - j \implies a[i] - a[j] - (i - j) = a[i] - (a[j] + (i - j)) \geq 0$

That is, if we add $i - j$ to each $a[j]$, it is enough to check that the height difference between $a[j]$ and $a[i]$ is non-negative. This can be done simply by adding $n - i$ to the height of each suitable step.

Having normalized heights, we can search for the longest sequence of equal or increasing numbers to obtain the solution. As in the previous case, it is necessary to store a pointer to the previous element in order to reconstruct the final staircase.

# Problem C. Clockwork Concert in D

## 8 Problem Statement

Given $n$ numbers $a_1, a_2, \ldots, a_n$, for each pair $(i, j)$ where $1 \leq i < j \leq n$, we:

- Remove $a_i$ and $a_j$
- Add $a_i + a_j$ to the sequence
- Compute the LCM of the resulting sequence

Find the sum of all such LCMs modulo 998 244 353.

Constraints: $n \leq 5 \times 10^5$, $1 \leq a_i \leq 10^6$.

## 9 Definitions

For prime $p$ and positive integer $A$:

- $v_p(A)$: Exponent of $p$ in $A$'s prime factorization ($p^{v_p(A)} \mid A$ but $p^{v_p(A)+1} \nmid A$)
- $M(p) = \max\{v_p(a_1), \cdots, v_p(a_n)\}$
- $m(p)$: Second largest value in $\{v_p(a_i)\}$ (equal to $M(p)$ if maximum appears multiple times)

The original LCM is $\prod_{p \in \text{Prime}} p^{M(p)}$.

## 10 Key Theorem

When replacing $a_i$ and $a_j$ with $a_i + a_j$, the change in $p$-exponent for the LCM is:

$$\Delta M_p(a_i, a_j) = [M(p) = v_p(a_i)](m(p) - M(p)) + [M(p) = v_p(a_j)](m(p) - M(p)) + \max(v_p(a_i + a_j) - M(p), 0)$$

## 11 Proof Sketch

Define indicators:

- $f_i = [M(p) = v_p(a_i)]$
- $f_j = [M(p) = v_p(a_j)]$
- $f_k = [v_p(a_i + a_j) \geq M(p)]$
- $f_l = [m(p) = M(p)]$

Case analysis shows the formula holds for all combinations of $(f_i, f_j)$.

## 12 Solution

Define functions:

$$f(i) = \prod_{p \in \text{Prime}} p^{[M(p)=v_p(i)](m(p)-M(p))}$$

$$g(j) = \prod_{p \in \text{Prime}} p^{\max(v_p(j)-M(p),0)}$$

The answer is:

$$\sum_{1 \le i < j \le n} f(a_i) f(a_j) g(a_i + a_j)$$

## 13 Implementation

Using generating functions:

- Let $F(x) = \sum_{i=1}^{n} f(a_i) x^{a_i}$
- Compute $F^2(x) \cdot G(x)$ where $G(x) = \sum_{i \ge 0} g(i) x^i$ (here $\cdot$ is a dot product)
- Remove diagonal terms where $i = j$

Complexity: $O((n + \max a_i) \log(\max a_i))$ using efficient prime factorization.

# Problem D. Draft Lottery

Convert all numbers to integers (by multiplying by 10). The sum then becomes 1000, and it is now obvious how to manage with 1000 balls. Then find the greatest common divisor (the "cost" of one ball) and divide all numbers by it.

The problem can also be solved without knowing the Euclidean algorithm, simply by noticing that the common divisor is a product of powers of two and five. Therefore, check whether the statement «all numbers are divisible by 2» is true; while it is true, divide all numbers in the current set by 2. Then do the same for 5. The sum of the numbers after the process is finished will be the answer.

# Problem E. Easy Game For Two Novices

## 14 Problem Statement

Given a card game with:

- $n$ card types, with exactly 2 cards per type (total $2n$ cards)
- Two players (First and Second) each start with $n$ cards
- Players alternate turns playing cards onto a stack
- When two identical cards appear in the stack:
  - Both cards and all cards between them are moved to the current player's hand
- Player loses when they have no cards to play

Additional constraints:

- Second player follows a fixed queue strategy (plays front card, adds won cards to back)
- First player can play any card optimally
- Find the minimal number of moves for First to guarantee victory
- $n \leq 3 \cdot 10^5$

# 15 Key Analysis

## 15.1 Lower Bound

The minimal move count is at least $n$, requiring that the Second player never gains cards.

## 15.2 Critical Observations

- If Second holds the pair of the bottom stack card, they will eventually claim it
- If First holds the pair, they control the stack by clearing it at will
- The "trump card" (stack pair) should be used as late as possible

## 15.3 Strategy with Pairs

When First has at least one pair initially:

1. Play one card from the pair initially
2. Clear the stack when Second could gain cards
3. Otherwise play any non-bottom card

This ensures:

- First maintains stack control after clearing
- First always has non-bottom cards to play

## 15.4 No Initial Pairs Case

When First has all $n$ distinct cards:

- Minimum moves: $n + 2$
- Example strategy against queue $1, 2, \ldots, n$:
  - Play sequence: $n, 1, 2, \ldots, n-1$
  - Leaves Second with only a pair of $n$ cards

### 16 Complexity Analysis

The optimal strategy requires:

- $O(n)$ time for pair checking
- $O(1)$ per move decision
- Overall $O(n)$ complexity

# Problem F. Find The Number of Stacks

### 17 Subtask 2 ($N \leq 100$) $O(N^2)$

In this problem, the most important thing is to notice that a greedy strategy gives the correct answer. But not the first approach that may come to mind: taking the ships from the strongest to the weakest and building a new tower every time a ship cannot be placed on top. Instead, we should take the ships from the weakest to the strongest, placing each ship at the base of the tower with the smallest number of ships.

Let us note that this gives the correct solution. Suppose we have placed the first $m$ ships (first, note that any arrangement can be reordered so that each tower is ordered from the strongest ship to the weakest). Then when adding the $(m+1)$-th ship, there are two possibilities: either it can be placed at the base of one of the existing towers (which does not affect the number of towers), or, if this is impossible, we create a new tower. Note that this is necessary, since these $m+1$ ships must be placed somehow, and any ship that we add at the end must end up in a new tower, because the strongest of them could not be placed using the current number of towers.

### 18 Subtask 3 ($N \leq 100000$) $O(N)$

To optimize the previous solution, a bucket sort technique is used for storing the data. Then sorting the ships and processing them takes $O(N)$. To choose which tower to place a ship into, a technique similar to bucketing is used: in the array `torres[i]` the number of towers of size $i$ is stored, the size of the smallest tower is maintained at every moment, and when adding a ship, `torres[i]-; torres[i+1]++;` is performed to update its size. This solution performs one operation per ship.

# Problem G. Governor Scroogius

To obtain a partition with the maximum sum, it is sufficient to simply break the string into blocks of one digit. The reduction in sum can only be achieved through "inversions" like IV, XL, and so on. Each inversion takes at least two digits. Depending on the first digit in such a notation, an inversion can yield a "gain" of 2, 20, or 200. Thus, if $n$ is odd, there is no solution.

If $n$ is even, then $n/2$ must be represented as a sum of the smallest number of 1s (represented by the pair `IV`, since $V < X$ lexicographically), 10s (represented by the pair `XC`, since $C < L$ lexicographically), and 100s (represented by the pair `CD`, since $D < M$ lexicographically). Since if we have more than 10 units, they can be replaced by one 10, and if we have more than 10 tens, they can be replaced by one hundred, the number of pairs `IV` will be $n/2 \bmod 10$, the number of pairs `XC` will be $[n/20] \bmod 10$, and the number of pairs `CD` will be $[n/200]$. The remaining task is to arrange them in lexicographical order, that is, first all `CD`, then all `IV`, and finally all `XC`.

# Problem H. Hard? NP-Hard!

# 19 Problem Description

Given:

- $n$ types of items where type $i$ has volume $v_i$ and value $c_i$
- $q$ queries, each specifying a knapsack capacity $x$
- Select items (with unlimited copies) such that:
  - Total volume exactly equals $x$
  - Total value is maximized

Constraints:

- $1 \leq n \leq 50$
- $1 \leq v_i \leq 10^5$, $1 \leq c_i \leq 10^6$
- $1 \leq q \leq 10^5$
- $10^{11} \leq x \leq 10^{12}$

# 20 Solution Approach

## 20.1 Key Insight

For very large $x$, the optimal solution consists of:

- Mostly selecting the item with maximum $\frac{c_i}{v_i}$ ratio (denoted as $(c_0, v_0)$)
- Using other items only for the residual volume ($x \bmod v_0$)

## 20.2 Algorithm

1. Compute the complete knapsack DP array $f_i$ for non-optimal items
2. Define:
$$g_i = \max_{k \geq 0} f_{i+kv_0} - kc_0$$
where $g_i$ represents the optimal residual solution for volume $i \bmod v_0$
3. The DP transition when adding item $(c_i, v_i)$:
$$g_{(j+v_i) \bmod v_0} \leftarrow \max\left(g_{(j+v_i) \bmod v_0}, g_j + c_i - \left\lfloor \frac{j+v_i}{v_0} \right\rfloor c_0\right)$$
4. Handle cyclic transitions by processing each cycle twice

## 20.3 Complexity

- Preprocessing: $O(n \max v_i)$
- Query: $O(1)$ per query
- Total: $O(q + n \max v_i)$

## 21 Correctness Proof

- Each residual solution uses $\leq v_0$ items
- Maximum residual volume: $\max v_i^2 = 10^{10} \ll V_{min} = 10^{11}$
- Ensures the residual solution always fits within $x$

# Problem I. Invisible Jewels and Judge J

## 22 Problem Description

Two players A and B simultaneously select integers:

- Player A chooses $x \in \{0, 1, \ldots, n\}$
- Player B chooses $y \in \{0, 1, \ldots, n\}$

Payoff matrix for Player A:

$$\text{Payoff}(x, y) = \begin{cases} x & \text{if } x > y \\ 0 & \text{if } x = y \\ y/2 & \text{if } x < y \end{cases}$$

## 23 Solution Approach

### 23.1 Game Decomposition

The game can be decomposed into $n$ $2 \times 2$ zero-sum subgames, each deciding whether to select the maximum value $n$.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0.5 | 1 | 1.5 |
| 1 | 1 | 0 | 1 | 1.5 |
| 2 | 2 | 2 | 0 | 1.5 |
| 3 | 3 | 3 | 3 | 0 |

### 23.2 Nash Equilibrium

For each $2 \times 2$ subgame with strategies:

| | Option 1 | Option 2 |
|---|---|---|
| Player A | $x$ | $y$ |
| Player B | $z$ | $w$ |

Let $(p, q = 1 - p)$ be Player A's mixed strategy probabilities. By Nash equilibrium:

$$px + qz = py + qw$$

Solving this equation gives the optimal mixed strategy for both players.

# 24 General Solution

- The complete solution can be built either bottom-up or top-down
- Each $2 \times 2$ subgame has a unique solution
- Optimal strategies form a probability distribution over $\{0, 1, \ldots, n\}$

## Details

### 1) The $2 \times 2$ subgame at level $j$

At level $j$ each player either:

- **H (High)**: play $j$, or
- **L (Low)**: "fall back" and play optimally in $\{0, \ldots, j-1\}$.

Let $V := V_{j-1}$ be Player A's value in the smaller game.
The $2 \times 2$ payoff matrix to Player A is:

| | B:L | B:H |
|---|---|---|
| A:L | $V$ | $\frac{j}{2}$ |
| A:H | $j$ | $0$ |

Let $p = \Pr(\text{A plays H})$ and $q = \Pr(\text{B plays H})$.

### 2) Indifference conditions

**A indifferent between L and H:** Payoff if A plays H:

$$j(1-q).$$

Payoff if A plays L:

$$V(1-q) + \frac{j}{2}q.$$

Setting equal:

$$j(1-q) = V(1-q) + \frac{j}{2}q \quad \Rightarrow \quad q = \frac{j-V}{\frac{3j}{2}-V}. \tag{$\star$}$$

**B indifferent between columns L and H:** Column L payoff to A:

$$pj + (1-p)V.$$

Column H payoff to A:

$$(1-p)\frac{j}{2}.$$

Setting equal:

$$pj + (1-p)V = (1-p)\frac{j}{2} \quad \Rightarrow \quad p = \frac{\frac{j}{2}-V}{\frac{3j}{2}-V}. \tag{$\star\star$}$$

Thus:

$$p_j = \frac{\frac{j}{2}-V_{j-1}}{\frac{3j}{2}-V_{j-1}}, \qquad q_j = \frac{j-V_{j-1}}{\frac{3j}{2}-V_{j-1}}.$$

### 3) Value recursion

At equilibrium, A's expected payoff equals either action's payoff when indifferent. Using "A plays H":

$$V_j = j(1-q) = j\left(1 - \frac{j - V_{j-1}}{\frac{3j}{2} - V_{j-1}}\right) = \frac{\frac{j^2}{2}}{\frac{3j}{2} - V_{j-1}} = \frac{j^2}{3j - 2V_{j-1}}.$$

With $V_0 = 0$.

### 4) Solving the recursion

**Claim:**

$$V_j = \frac{j(j+1)}{2(j+2)}.$$

**Proof by induction:** Base $j = 0$: $V_0 = 0$ holds.
Assume $V_{j-1} = \frac{(j-1)j}{2(j+1)}$. Then:

$$V_j = \frac{j^2}{3j - 2V_{j-1}} = \frac{j^2}{3j - \frac{j(j-1)}{j+1}} = \frac{j}{\frac{3j - \frac{j(j-1)}{j+1}}{j}} = \frac{j}{\frac{3 - \frac{j-1}{j+1}}{1}} = \frac{j(j+1)}{2(j+2)}.$$

Thus the formula holds for all $j$.

### 5) Simplifying $p_j$ and $q_j$

With $V_{j-1} = \frac{(j-1)j}{2(j+1)}$, compute the common denominator:

$$\frac{3j}{2} - V_{j-1} = \frac{3j}{2} - \frac{j(j-1)}{2(j+1)} = \frac{j(j+2)}{j+1}.$$

Numerators:

$$\frac{j}{2} - V_{j-1} = \frac{j}{2} - \frac{j(j-1)}{2(j+1)} = \frac{j}{j+1},$$

$$j - V_{j-1} = j - \frac{j(j-1)}{2(j+1)} = \frac{j(j+3)}{2(j+1)}.$$

Therefore:

$$p_j = \frac{\frac{j}{j+1}}{\frac{j(j+2)}{j+1}} = \frac{1}{j+2},$$

$$q_j = \frac{\frac{j(j+3)}{2(j+1)}}{\frac{j(j+2)}{j+1}} = \frac{j+3}{2(j+2)}.$$

**Conclusion:** Once $V_{j-1}$ is in closed form, the mixing probabilities $p_j$ and $q_j$ simplify neatly to:

$$\boxed{p_j = \frac{1}{j+2}, \quad q_j = \frac{j+3}{2(j+2)}}.$$

# Problem J. Juggling With Sequences

# 25 Problem Description

Given a sequence specification where each integer $i$ ($1 \leq i \leq n$) appears exactly $a_i$ times, construct a permutation of this sequence that maximizes the sum of prefix majority elements.

Constraints: $1 \leq a_i, n \leq 10^5$.

# 26 Solution Approach

## 26.1 Greedy Intuition

The optimal strategy prioritizes:

- Maximizing the number of prefixes where $n$ is the majority
- Then maximizing prefixes where $n-1$ is majority, and so on

## 26.2 Algorithm

1. First place all $a_n$ copies of $n$
2. For other numbers $i$, place $\min(a_n, a_i)$ copies
3. Next place all $a_{n-1}$ copies of $n-1$
4. For other numbers $i$, place $\min(\max(a_n, a_{n-1}), a_i)$ copies
5. Repeat this process for decreasing values

## 26.3 Optimization

The naive approach is $O(n^2)$. Instead:

- Insert numbers in rounds: first all 1st occurrences, then 2nd occurrences, etc.
- For each round $k$, the contribution is max(remaining counts) $\times$ $|\{i : a_i \geq k\}|$
- Maintain counts using a set for $O(\max a_i + n)$ complexity

# 27 Correctness Proof

The greedy approach achieves the theoretical maximum because:

- For any $i$, the number of prefixes with majority $\geq i$ is maximized
- The round-based construction matches this upper bound

# Problem K. Kinky Chase Festival

# 28 Problem Description

Players can repeatedly challenge a tower, aiming to achieve exactly $i$ points ($1 \leq i \leq M$) through multiple attempts. Each challenge involves:

- Possible outcomes per floor: failure, success, or success with high honors
- Option to exit after completing a floor
- Penalty mechanism:
  - Failed exit: accumulated points multiplied by penalty coefficient
  - Voluntary exit or full completion: direct accumulation of all points

Constraints: $1 \leq n \leq 6$ (max floors), $1 \leq m \leq 10000$ (target points).

# 29 Solution Approach

## 29.1 Dynamic Programming Formulation

Define $f(i, S)$ as the probability of achieving exactly $i$ points from state $S$:

$$f(i,S) = \begin{cases} f(i - \text{Fail}(S), S_0) & \text{if failure occurs} \\ f(i - \text{Success}(S), S_0) & \text{if voluntary/successful exit} \\ f(i, S') & \text{if continuing challenge} \end{cases}$$

where $\text{Fail}(S)$ and $\text{Success}(S)$ represent points earned in each case.

## 29.2 Computational Challenges

- Self-loops occur when $\text{Fail}(S) = 0$
- Dependency structure allows sequential computation by $i$

## 29.3 Numerical Solution

- **Binary Search Approach**:
  - For each $f(i, S_0)$, perform binary search on probability values
  - Terminate when reaching precision $\epsilon = 10^{-18}$ (long double recommended)
- Complexity: $O(m2^n \log 1/\epsilon)$
- Alternative iterative methods also possible

# 30 Implementation Notes

- Sample 2 contains critical self-loop cases for testing
- Precision below $10^{-6}$ may fail due to strengthened test data
- Runtime: $\leq 0.5$s for proper implementation (author-tested)

## Acknowledgement

The problem background was adapted after the original game's service termination announcement during problem preparation.