Bài 1: SUMN

Dựa vào công thức truy hồi của A, chúng ta có thể chứng minh được dãy A có chu trình tối đa là M.
Gọi độ dài chu trình là L, tổng các phần tử thuộc chu trình đó là S. Đặt $N = q * L + b$ $(b < L)$.
Kết quả là $S * q + K$, với K là tổng của b số đầu tiên trong chu trình.
Độ phức tạp: $O(M)$.

Bài 2: SSUBSEG

Gợi ý: hãy thử "lộn ngược" lại bài toán: thay vì ta xóa đi từng số một trên bảng, ta có N ô trống, ta điền từng chữ số một, và sau khi ta điền mỗi chữ số, hãy tìm dãy liên tiếp có tổng lớn nhất.
Bằng cách này, hãy thử cải tiến bài toán từ $O(N^2) => O(N)$.
Ta dùng kỹ thuật Disjoint Set-Union để cải tiến bài toán này.
Mỗi vị trí là một đỉnh, và các cạnh sẽ nối hai số cạnh nhau. Khi ta thêm một số vào một vị trí, ta sẽ đồng thời thêm cạnh nối nó với vị trí ở bên trái, và vị trí ở bên phải, rồi cộng tổng của các thành phần liên thông vừa ghép vào và tạo thành một thành phần liên thông mới. Ta sẽ cần duy trì biến max để kiểm soát xem thành phần liên thông nào đang có kích thước lớn nhất.
Độ phức tạp: $O(N)$.

Bài 4: NMIN

Ta sẽ dựng một đồ thị, mỗi đỉnh ứng với trạng thái (r, s) – nghĩa là tổng các chữ số là s và phần dư khi chia cho Q là r.
Sau đó, tìm "đường đi ngắn nhất", tính từ trạng thái $(0, 0)$, đến trạng thái $(0, S)$.
Từ đỉnh (x, y) bất kỳ, với mọi d là số nguyên từ 0 đến 9, ta có thể đến được các đỉnh $((x * 10 + d)\ mod\ Q, y + d)$. Ta sẽ cần quan tâm xem đỉnh này đã được đưa vào trong hàng đợi BFS chưa để quyết định xem có đưa vào hay không, và cần quan tâm các thứ tự đưa vào trong hàng đợi. Sau khi đến được đỉnh cuối cùng, ta có thể dùng mảng truy vết về đỉnh $(0, 0)$ để xem số cần tìm là bao nhiêu.
Độ phức tạp: $O(Q * S)$.

Bài 3: DOLL

Bài toán này liên tưởng tới thuật toán tính tổ hợp. Tuy nhiên, thay vì mỗi thùng chỉ có một búp bê, giờ phải tính cho trường hợp mỗi thùng có nhiều búp bê.
Sử dụng phương pháp quy hoạch động như sau:

- Đặt $f[i][j]$ là số cách để chọn ra j búp bê khác thùng, khi xét đến thùng thứ i.
- Công thức truy hồi như sau:
  - $f[i][j] = f[i-1][j-1] * a[i] + f[i-1][j]$
  - Tức là nếu chọn một búp bê ở thùng thứ i, sẽ có thêm $f[i-1][j-1] * a[i]$ cách, ngược lại nếu không chọn thì ta có $f[i-1][j]$ cách.
- Đáp án của bài toán là $f[n][m]$, khi cài đặt cần lưu ý phần $module$ để tránh tràn số.
- Độ $phức\ tạp$: $O(N * M)$

Bài 5: SAB

Ta có thể "thu gọn" động tác chọn một dãy nào đó và sắp xếp lại -> chọn hai số cạnh nhau bất kỳ và đổi chỗ chúng nếu thỏa mãn số đứng trước lớn hơn số đứng sau. Với quan sát này, bài toán sẽ trở nên đơn giản hơn một chút.
Với mỗi vị trí i từ trái qua phải trong dãy B, ta sẽ tìm số đầu tiên bằng với nó xuất hiện trong đoạn từ i đến N ở dãy A. Điều chúng ta cần xử lý ở đây là đánh giá xem liệu trong dãy A, có thể chuyển dần số ở vị trí vừa tìm đó trở về đúng vị trí i không. Khi đó, ta chỉ cần kiểm tra trên đường đi về của nó có số nào bé hơn nó không, vì nếu có, 2 số sẽ không được phép đổi chỗ cho nhau.

Dựa vào quan sát trên, ta có thể dùng thuật $O(N^2)$ và đổi chỗ từng số trở về tương tự như bubble/insertion sort. Còn để kiểm tra tốt hơn, chúng ta sẽ cần sử dụng đến Segment Tree để kiểm tra cho từng thao tác.
Độ phức tạp: $O(Nlog_2N)$

Bài 6: CTOP

Để giải được bài toán này, ta cần hiểu sâu một chút về thuật toán tìm $diameter$ hay đường đi dài nhất trên cây.
Gọi x, y là hai đầu của đường đi dài nhất trên cây ***giữa hai đỉnh đặc biệt.*** Gọi $L[u]$ là đường đi ngắn nhất từ đỉnh x tới đỉnh u, tương tự $R[u]$ là đường đi ngắn nhất từ đỉnh y tới đỉnh u. Ta có thể xây dựng mảng L và R bằng hai lần DFS từ đỉnh x và y .
Lúc này, dễ thấy rằng khoảng cách lớn nhất từ một đỉnh đặc biệt tới đỉnh u chính bằng $\max(L[u], R[u])$.
Do vậy, với mỗi đỉnh u từ $1 \rightarrow n$, ta chỉ cần kiểm tra xem $\max(L[u], R[u]) \leq k$
hay không, nếu có thì đỉnh u chính là đỉnh tốt.
Độ phức tạp: $O(N)$