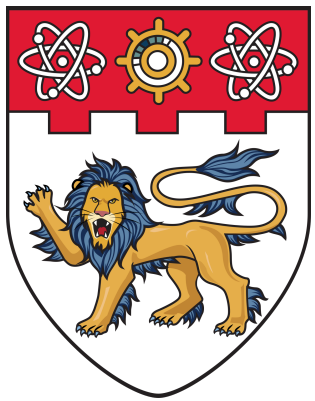
Nanyang Programming Contest 2026

Stage 1

Lee Zong Yu, Pu Fanyi, Zhou Xuhang, Joshua James, Swe Hon

Nanyang Technological University

14 March 2026



- Easy
  - Problem A: Square
  - Problem B: Word
- Medium
  - Problem C: Left is Tall
  - Problem D: Complete Binary Tree
- Hard
  - Problem E: Salt and Honey
  - Problem F: Cursed Note

Problem A: Square

Problem Author: Joshua James
Development: Joshua James
Editorial: Lee Zong Yu

Problem

Abridged Problem Statement

Given L, compare L^2 with L^3.

Solution

$L^2 > L^3$ if and only if $L < 1$.
$L^2 < L^3$ if and only if $L > 1$.
$L^2 = L^3$ if and only if $L = 1$ and the answer of A^{V^L} is 1 because A, V, and L are all 1.

Problem B: Word

Problem Author: Codeforces
Development: Zhou Xuhang
Editorial: Lee Zong Yu

Problem

Abridged Problem Statement

Given a string s, as long as there are two consecutive vowels, delete the second one.

- Subtask 1: $n \leq 1000$
- Subtask 2: $n \leq 10^5$

Subtask 1 ($n \leq 1000$)

Solution

- While the string is not valid, iterates from the beginning of the string until you find two consecutive characters that are vowels. Then, remove the second character.
- The removal of a character at index i in the string of length n can be done by replacing the original string s with $s[1 : i-1]$ concatenates $s[i+1 : n]$.
- The operation of finding two consecutive characters and removal is $O(n)$
- This is repeated for at most n rounds (because each removal reduces the length of string by 1. Hence, the final algorithm is $O(n^2)$

Subtask 2 ($n \leq 10^5$)

Solution

- To reduce the time complexity to $O(N)$, observe that you do not have to always iterate from the front because when you iterate until index i, the operations in previous i iterations have made the substring prefix at index i valid.
- Secondly, removal of a character at an index (i.e. always do the concatenation) is costly. You can maintain an answer string (initially empty). As you iterate, append the non-removal character to the back of the answer string.

  i.e. If you find out that the current character and the previous character are vowels, the current character will be removed. Hence, you do not append it to the answer string; otherwise, you will append it.
- The append operation is efficient $O(1)$. Since you iterate each character at most once, the total time complexity is $O(N)$.

Problem C: Left is Tall

Problem Author: Lee Zong Yu
Development: Lee Zong Yu
Editorial: Lee Zong Yu

Abridged Problem Statement

Given array of n integers, for each index, output the nearest index to the left with element strictly greater than it.

- Subtask 1: $n \leq 10^3$
- Subtask 2: $n \leq 10^6$

Subtask 1

- Since $n \leq 1000$, a $O(n^2)$ solution is accepted.
- Hence, just simulate the process with a nested for loop. The outer for loop is to iterate the index i (find the answer to the index i).
- The inner for loop iterates from i back to 1 to find the first index with element that is greater than it

Subtask 2

To reduce the solution to $O(n)$ or $O(nlogn)$, we figure out there are at least 4 different solutions

Monotonic Stack

- As you iterate through the elements, maintain a stack with the top of the stack being the smallest element.
- The crucial observation here is if $j < i$ and $a[j] \leq a[i]$, the index j is impossible to appear as an answer to any index greater than i. This is because for any index $k > i$, if $a[k] > a[i]$, it follows $a[k] > a[j]$.
- Hence, the stack stores a list of possible answers (in strictly increasing order from the top), and while the current element is greater than the top of the stack, pop the top element (because it wouldn't be seen from any buildings after it). Then place the current element at the top of the stack. Note that: this element is smaller than any element in the stack (Monotonicity is preserved).

Second Solution

Balanced Binary Search Tree

- You may maintain a balanced binary search tree (or set in C++) storing all the index from 0 to $n-1$ (0-indexed).
- Enumerate from the smallest element. In the enumeration, all the elements with the same value are considered together. First, remove all the corresponding index from the balanced binary search tree. Hence, the balanced binary search tree only maintains indexes with elements strictly greater than it. Second, for each of the removed index (after removal of ALL indices), find the largest index in the balanced binary search tree that is smaller than it. This could be done by traversing the binary search tree or performing a lower bound operation in the set (then decrement the pointer to point to the previous element).
- The removal and traversing of element in binary search tree takes $O(\log n)$. Hence, the total time complexity is $O(n \log n)$

Third Solution

Note that the third solution requires the use of a segment tree. If you haven,t learn that, we recommend you learn it before reading the editorial.

Segment tree with coordinate compression

- Iterates the array from the beginning. We maintain a segment tree such that $segtree[val]$ stores the largest index (smaller than the current index) with value val.
- To solve the problem for index i, we perform a range max query from $a[i]+1$ to ∞. This will give us the largest index with value greater than $a[i]$.
- After solving the problem, we update the value at position $a[i]$ to be i in the segment tree.
- Since the value of the elements can be arbitrary large but there are at most n distinct values, we can apply a coordinate compression.
- Time Complexity is $O(n \log n)$

Fourth Solution

Note that the third solution requires the knowledge of range max query. If you haven,t learn that, we recommend you learn it before reading the editorial.

Binary Search with RMQ

- Iterate from the beginning of the array. We denote $RMQ(l, r)$ as the max element from the range l to r.
- As we iterate at index i, if $RMQ(1, i-1) \leq a[i]$, then the answer is -1 (no element before i is greater than it). Otherwise, we do a binary search.
- That is, set $l = 1, r = i-1, m = (l+r)/2$. If $RMQ(m+1, r) > a[i]$, we set $l = m+1$. It is because the answer (index) lies at the right half of the array. Otherwise, we set $r = m$ (answer lies at the left half). Eventually, when $l = r$, we find the answer (first index to the right greater than it).
- RMQ can be efficiently done by a Sparse Table $O(1)$ or by traversing the node in the segment tree. It takes $O(\log n)$ to find the answer to a node and hence $O(n \log n)$ in total.

Problem D: Complete Binary Tree

Problem Author: Zhou Xuhang
Development: Zhou Xuhang
Editorial: Lee Zong Yu

Complete Binary Tree

Abridged Problem Statement

Given a complete binary tree with height n and the vertices is ordered in level-order traversal ordered. Find the LCA of any two vertices.

- Subtask 1: $n \leq 5$ implies there are at most 32 nodes.
- Subtask 2: $n \leq 20$ implies there are at most $2^{20} \approx 10^6$ nodes.
- Subtask 3: $n \leq 30$ implies there are at most $2^{30} \approx 10^9$ nodes.
- Subtask 4: $n \leq 10000$ implies there are at most $2^{10000} \approx 10^{3011}$ nodes.

Subtask 1

- Since n is small, you may build the complete binary tree. Then, you label the vertices with a breath-first search (BFS) using a queue (You may refer to SC1007 Tutorial for it).
- Then, there are a few way to solve it. For example, enumerate all the vertices to treat them as a potential candidates. Then simply check if vertex x and vertex y are in the subtree of the potential candidate by traversing down from the potential candidate. Out of all the potential candidates, choose the one with the largest depth (or the largest vertex id because larger vertex id implies it is "lower").
- Time complexity is $O(2^n \cdot 2^n)$

Subtask 2

Understanding the structure of ancestors

- n is still small, you may build the complete binary tree and label them with BFS.
- However, since there are 10^6 nodes, you cannot enumerate all the nodes and check all nodes in a subtree naively.
- One potential answer is to understand the structure of ancestors. The number of ancestors of a node is at most the depth of the node. That is, a node at depth i, will have at most i ancestors.
- In this way, we can enumerate down the tree to find node x and node y, then we go all the way back to find the list of ancestors.
- Node x and node y each has at most n ancestors. Now the problem becomes finding the largest number that appeared in

Subtask 3

Understanding the structure of level-order traversal

- now n is too large to build the entire binary tree. You will get MLE verdict, if you do so. How to we get the list of ancestors without building the tree?
- We need to understand the structure of such traversal. Observe that the vertex id of parent of vertex with id i is $i/2$. In fact, the left child of vertex with id i is $2 \cdot i$ and the right child of vertex with id i is $2 \cdot i + 1$.
- Hence, we can construct the list of ancestors by dividing the number by 2. Eventually, the vertex id will become 1 and that is the root of the entire tree.
- Then we can find the LCA by the same algorithm.

Time Complexity: $O(n^2)$

Subtask 4

Note that we suggest you use Python to solve the problem because the vertex ID is too large to be stored as integers in C++ or Java (You may store them as strings, but it is annoying as you have to implement the arithmetic operation on strings yourself).

Why $O(N^2)$ doesn,t work for input $N = 10000$

- Under a normal setting, $O(N^2)$ solution should works for $N = 10000$.
- In this case, $O(N^2)$ is not actually a correct time complexity for brute force solution. We ignore the time complexity for comparing two number. In a usual case of other problem, the number is small (i.e. at most 32 bits or 64 bits). In this problem, the number could be as large as 2^n and the number of bits in a number could have n bits. Hence, the time complexity introduced when comparing two large number cannot be ignored anymore.
- Hence, the actual time complexity of previous solution is $O(n^3)$, which should fail $n = 10000$.

Subtask 4

A more efficient way to find LCA

- After getting the list of ancestors (from the highest vertex id to the lowest vertex id), you may observe that the suffixes of two ancestors are the same.
- For example, $[7, 3, 1]$ and $[24, 12, 6, 3, 1]$ have the same suffix $[3, 1]$. And 3 is indeed the LCA of vertex 7 and vertex 24.
- Hence, instead of enumerating from the beginning of the array, you can enumerate from the end of the array until the first index that they different.
- The algorithm is $O(n)$

Time Complexity is $O(n^2)$, including comparing the numbers.

Subtask 4 (alternative)

A more efficient way to find LCA

- Alternatively, you may divide x by 2 if x is larger; otherwise, divide y by 2.
- Eventually, the x equal to y and that is the answer.

A more efficient way to find LCA

- Alternatively, start from root node (vertex id 1) and check if x and y is in the subtree of which child node (left or right).
- To check if a query node is at the left or right subtree of a node k, we need to know the depth of the node k (with root node being depth 0), let say d. Then, if the $d+1$ bit of the query node from the MSB is 0, then it means the query node is at the left subtree; otherwise, right subtree.
- We can understand this as the appending a 0 bit to the number if it is the left subtree.
- Hence, as we traverse down the node, we maintain the depth. Then check if the $d+1$ bit of x and y from MSB is the same or not. Then traverse to the corresponding subtree if they are the same.

Problem E: Salt and Honey

Problem Author: Joshua James
Development: Joshua James
Editorial: Joshua James

Abridged Problem Statement

Abridged Problem Statement

You are given an integer N and the set S of integers $\{1, 2, \ldots, 2N\}$. How many ways are there to group them into pairs such that each number belongs in a pair and the difference of the two elements in each pair is either 1 or N.

N is defined in the problem.

- Subtask 1: $T = 1, 1 \leq N \leq 10$
- Subtask 2: $T = 1, 1 \leq N \leq 1000$
- Subtask 3: $T = 1, 1 \leq N \leq 1000000$
- Subtask 4: $T = 5000, 1 \leq N \leq 1000000$

Hint 1

How can we frame this as a DP problem?

Subtask 1 - $N \leq 10$

- Try all possible pairings.
- Time Complexity: $\mathcal{O}(N!)$

Subtask 2 - $N \leq 1000$

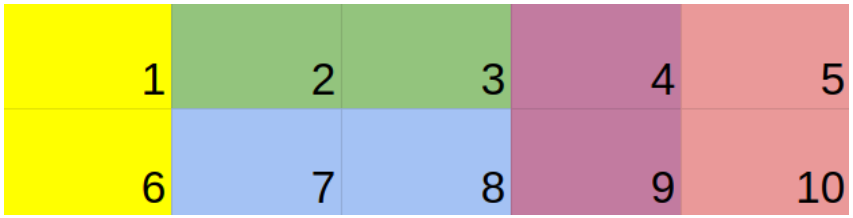
Honestly, no idea. There might be a $\mathcal{O}(N^2)$ solution.

Subtask 3 - $N \leq 10^6$

- Let's only consider the first N numbers $(1..N)$.
- For each number i, we can pair i with $i+1$ or $i+N$.
- Let's define dp_i as the number of pairings for $\{1, 2, \cdots, i, N+1, N+2, \cdots, N+i\}$. Initially $f_0 = 1$ and it is trivial to see that $f_1 = 1$. The answer for this problem is at f_N.
- Suppose we have the values for f_1 to f_{i-1}. We can either pair i with $i+N$ or i with $i-1$ and pair $i+N$ with $i+N-1$.
  - Therefore, $f_i = f_{i-1} + f_{i-2}$.
- For odd N, consider the possibility of having all i paired with $i+1$. The answer for N is $f_N + (N \bmod 2)$.
- Time Complexity: $\mathcal{O}(N)$

Subtask 3 - $N \leq 10^6$

Another way to see it is the number of ways to tile a $2 \times N$ grid with 1×2 or 2×1 dominos which has a very neat Fibonacci solution.



Subtask 4 - $T = 1000, N \leq 10^6$

- Use the same f_i as Subtask 3 but persist the array f for all testcases.
- You can precompute the value of f_i for all $1 \leq i \leq MAXN = 10^6$.
- Time Complexity: $\mathcal{O}(MAX)$

Subtask 5 - $T = 1000, N \leq 10^9$

- This subtask does not exist for NPC.
- Fibonacci has a closed form:

$$f_N = \frac{(\frac{1+\sqrt{5}}{2})^N - (\frac{1-\sqrt{5}}{2})^N}{\sqrt{5}}$$

- Alternatively, you can use matrix exponentiation using the following relation:

$$\begin{bmatrix} f_i \\ f_{i-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_{i-1} \\ f_{i-2} \end{bmatrix}$$

- Therefore,

$$\begin{bmatrix} f_N \\ f_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{N-1} \begin{bmatrix} f_1 \\ f_0 \end{bmatrix}$$

- We can do this using fast exponentiation in $\mathcal{O}(\log N)$.
- Time Complexity: $\mathcal{O}(T \log N)$

Problem F: Cursed Note

Problem Author: Swe Hon
Development: Pu Fanyi
Editorial: Pu Fanyi

Problem

Abridged Problem Statement

On a number line, n Type A points and m Type B points have distinct coordinates. Execute k operations: each operation moves all A points by $+1$ and B by -1, or vice versa. Movement is continuous. If an A point touches or crosses any B point, it is permanently removed. After each operation, output the number of remaining A points.

- Subtask 1 (20 pts): $n, m, k \leq 500$
- Subtask 2 (20 pts): $m = 1$
- Subtask 3 (20 pts): $a_i < b_j$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$
- Subtask 4 (20 pts): $n, m \leq 5\,000$
- Subtask 5 (20 pts): No additional constraints

Subtask 1: $n, m, k \leq 500$

Observation: Relative Motion

All points of the same type move identically $\Rightarrow$ for any pair (A_i, B_j), relative distance changes by exactly ± 2 per operation.

Brute Force Simulation — $O(n \cdot k)$

- For each A point, precompute its nearest left/right B neighbor (sort B, binary search).
- Maintain cumulative displacement d. At each step:
  - Op 1 ($\delta = +1$): check if each alive A collides with its nearest **right** B.
  - Op 2 ($\delta = -1$): check if each alive A collides with its nearest **left** B.

Subtask 2: $m = 1$

Fix A, Move B

Only one B point b_0. Fix all A points in place; only B moves (relative displacement ± 2 per step). Each time B passes through an A point, that A is removed.

Algorithm — $O(n \log n + k + n)$

1. Sort A and store in a **linked list**. Record B's initial position in the list.
2. Each step, B moves by ± 2. Walk along the linked list in that direction, deleting every A point that B passes through.
3. Each A point is deleted at most once $\Rightarrow$ total deletions across all steps is $O(n)$.

Subtask 3: $a_i < b_j$ for all i, j

Only the Nearest B Matters

If A_i can reach b_{j_2} on its right, it must have **already passed through** a closer b_{j_1} first. So each A only needs to check its **nearest** B neighbor.

Collision Condition

Define cumulative displacement: $\text{pre}[t] = \text{pre}[t-1] \pm 1$. Let $g = b_j - a_i$. Collision when $\text{pre}[t] \geq \lceil g/2 \rceil$.

Prefix Extrema are Monotone

$\text{mx}[t] = \max_{0 \leq i \leq t} \text{pre}[i]$ is non-decreasing $\Rightarrow$ **binary search** for earliest t with $\text{mx}[t] \geq \lceil g/2 \rceil$.

Subtask 3: $a_i < b_j$ for all i, j

All A Left of All B — $O(k + n \log m)$

Since all A are left of all B, collisions can only happen to the **right**. No need for mn.

1. Precompute pre$[t]$ and mx$[t]$.
2. Sort B. For each a_i, binary search for nearest right B neighbor b_r.
3. Threshold $z = \lceil (b_r - a_i)/2 \rceil$, binary search on mx.
4. Difference array for answers.

Why this subtask matters

Introduces: **nearest neighbor + prefix extrema binary search + difference array**. But only single-sided (right), making it easier.

Subtask 4: $n, m \leq 5\,000$

Offline Pairwise Checking — $O(nm \log k + k)$

Now B can be on **both sides** of A. Extend with $\text{mn}[t] = \min_{0 \leq i \leq t} \text{pre}[i]$ (non-increasing, also binary-searchable).

1. Precompute $\text{pre}[t]$, $\text{mx}[t]$, $\text{mn}[t]$.
2. For each A point a_i, enumerate **all** B points b_j:
   - $b_j > a_i$: threshold $z = \lceil (b_j - a_i)/2 \rceil$, binary search on mx.
   - $b_j < a_i$: threshold $z = \lceil (a_i - b_j)/2 \rceil$, binary search on mn.
3. Take the minimum collision time across all B as the death time.
4. Use a **difference array** to count deaths per step.

Subtask 5: Full Solution

Complete Algorithm — $O(k + n \log(\max(n, m)))$

Combine nearest-neighbor optimization with both-sided prefix extrema:

1. Compute pre[t], mx[t], mn[t] in $O(k)$.
2. Sort B in $O(m \log m)$.
3. For each A point a_i, find nearest left/right B via binary search:
   - Right neighbor b_r: binary search on mx for earliest t with $\text{mx}[t] \geq \lceil (b_r - a_i)/2 \rceil$.
   - Left neighbor b_l: binary search on mn for earliest t with $\text{mn}[t] \leq -\lceil (a_i - b_l)/2 \rceil$.
   - $T_{\text{death}} = \min(t_r, t_l)$.
4. Difference array + prefix sum for answers.