NZPC 2026 – Editorial

New Zealand Programming Contest

8th August 2026

Table of Contents

You're For The High Jump

You're For The High Jump

Problem Statement
Each student has 3 recorded jumps (a height, `0` for a failed jump, or `NJ` for no jump). List, in input order, every student who cleared the qualifying height QH at least once.

Solution
For each student, treat `NJ` and `0` as height 0, then check whether the maximum of their three values is $\geq QH$. Print the names of qualifying students in the order read. If nobody qualifies, print nothing. A straightforward $O(N)$ scan.

Cubing

Cubing

Problem Statement
Given Claire's and Mai's daily solve times, find Claire's longest run of consecutive wins. Claire wins day i when her time is strictly less than Mai's; a draw or loss ends the streak.

Solution
Sweep the days left to right, keeping a running `current` streak length and the best streak seen:

- If $c_i < m_i$: `current+=1`, update the best.
- Otherwise (draw or loss): reset `current=0`.

Output the best. Runs in $O(n)$.

Germs

Germs

Problem Statement
Given n bacteria counts, find the minimum sum over all contiguous windows of exactly r dishes.

Naive Approach
Summing each of the $n - r + 1$ windows from scratch is $O(n \cdot r)$, which is too slow for large n and r (but may suffice to get AC).

Sliding Window
Compute the sum of the first r dishes, then slide: when moving one step right, add the entering element and subtract the leaving element. Track the minimum window sum along the way.

$$S_{i+1} = S_i + b_{i+r} - b_i$$

This gives an $O(n)$ solution.

Bisques

Bisques

Problem Statement
Two teams of two players each. Compute the bisques awarded to the weaker (higher total handicap) team, equal to half the difference of the team handicap totals.

Solution
Let $t_1 = h_1 + h_2$ and $t_2 = h_3 + h_4$ be the team totals, and $d = |t_1 - t_2|$. The number of bisques is $d/2$, awarded to the team with the larger total.

- If $t_1 = t_2$: print `No bisques are awarded.`
- Half bisques occur when d is odd — print e.g. `2.5` (or `0.5`); avoid a trailing `.0` on whole numbers.
- Use the singular `1 bisque is` when the count is exactly 1.

Best Takeaways in Town

Best Takeaways in Town

Problem Statement
Each visit records 6 grades (`A`/`B`/`C`) for a store. With `A`= 5, `B`= 2, `C`= 0, find the store with the highest *average* score per visit.

Solution
Parse each line by splitting on the comma: the part before it is the store name, the part after is the 6 grades. For every store accumulate a running total of points and a count of visits (a dictionary keyed by name). The winner maximises total/visits.

Details
Store names may contain spaces and up to 4 words — split on the *comma*, not on whitespace. The panel guarantees no ties, and the output line must end in a full stop.

Puppies and Posts

Puppies and Posts

Problem Statement
Posts define a sequence of gaps (widths). A dog walks the gaps in order; if it cannot fit through a gap it simply skips to the next one. Find the dog(s) that fit through the most gaps.

Solution
For each dog of width w, count how many gaps have width $\geq w$: the "skip and try the next" rule makes order irrelevant, so this count is the dog's answer.

- Track the maximum gap-count over all dogs.
- Output every dog achieving it, in **alphabetical** order.
- If no dog fits through *any* gap, print `No dogs fit!`

Alchemist's Secret

Alchemist's Secret

Problem Statement
A message is encrypted with a Caesar shift over a 27-symbol alphabet (a$= 1, \ldots,$z$= 26$, underscore $= 27$). Recover the plaintext, given a keyword guaranteed to appear in it exactly once.

Solution
There are only 27 possible shifts, so brute force them. For each offset s, decrypt by shifting every symbol back by s and test whether the keyword W occurs as a substring. Exactly one offset yields a plaintext containing W; print it.

Cost: $27 \cdot |S|$ — easily fast enough.

More Efficient Solution

A Caesar shift adds the same offset to every symbol, so the *differences between consecutive symbols* are unchanged by encryption. This lets us locate the keyword without trying all shifts.

Map each symbol to its number and take consecutive differences. The keyword `gold` $= [7, 15, 12, 4]$ has difference pattern

$$[15-7,\ 12-15,\ 4-12] = [8,\ -3,\ -8].$$

Compute the same consecutive-difference sequence for the whole ciphertext, then find where the pattern $[8, -3, -8]$ occurs (guaranteed exactly once).

Recovering the shift

At that position the ciphertext symbol lines up with g$=7$. The offset is (ciphertext value there) $-7 \pmod{27}$. Shift every symbol back by that offset to decrypt the message. This is a single pattern match instead of 27 full decryptions.

Order N

Order N

Problem Statement
Draw the letter `N` in ASCII art with a fill character c, height h, and stroke thickness t: two vertical bars of width t joined by a slope-1 diagonal.

Solution
Build the output row by row. On each row i (from 0 to $h-1$):

- The **left bar** occupies the first t columns.
- The **right bar** occupies the last t columns.
- The **diagonal** block of width t starts one column further right on each row (slope 1), linking the left bar's top to the right bar's bottom.

Fill those column ranges with c and the rest with spaces.

Raising the Flag

Raising the Flag

Problem Statement
Create a boolean array of size n such that after flipping the value at every multiple of i for i from 1 to n, there are k ones in the array.

Naive Solution
Start with an array of size n initialized to all zeros. Simulate the process, create an array and iterate through it for each i. Keep track of the number of ones in the final array. Adjust the initial array depending on whether there are too many or too few ones in the final array.

This solution is $O(n \log n)$ which can work for this problem. If you have a bad implementation, it might be $O(n^2)$ which will not run in time.

Key Insight

A position i is flipped for every divisor it has. Most numbers have an even number of divisors, except perfect squares which have an odd number of divisors (because one of the divisors is repeated). Therefore, only the positions that are perfect squares will be flipped at the end.

Optimized Solution

Start with an array of size n initialised to all zeros and set the value at every perfect-square position $i^2 \leq n$ to 1. This chosen initial array yields a *final* array of all zeros. To instead finish with exactly k ones (where $0 \leq k \leq n$), additionally flip the initial value of the first k positions $1, \ldots, k$; each such position ends as a single one. Printing this initial array solves it in $O(\sqrt{n})$.

Segment Fault

Segment Fault

Problem Statement
Each digit of a 7-segment display has **at most one** faulty (never-lit) segment, and the faulty segment may differ from digit to digit. Given the n observed patterns, answer q queries for the k-th largest possible original score.

Key Insight
A fault can only turn a segment *off*. So an observed pattern is some real digit with 0 or 1 lit segments removed. You can brute-force all segments of each digit to find the candidates for each position.

Each position is **independent** so you can do it one by one.

Per-digit Candidates

Solution
For each position i, find the set of candidates candidates_i

- If **any** digit has an empty candidate list, no original score exists $\Rightarrow$ output `ERROR`.
- Otherwise the number of possible scores is $m = \prod_i |\text{candidates}_i|$, and each score picks one candidate per position.

Pinning high digits

The key trick: bound the free positions

$m = \prod_i |\text{candidates}_i|$ can be astronomically large (up to 2^n), but $k \leq 10^6$. Walk the positions from **least** to **most** significant, keeping a running product `max_value` of the candidate counts seen so far. Once `max_value` $> 10^6$, every *more* significant position is **pinned** to its single largest candidate.

Mixed-radix descent
Store each position's candidates in **descending** order and start with every position at its largest candidate (rank 1). Reading positions from the most significant, let S_i be the number of combinations of the positions after i. Choosing the j-th (0-indexed) candidate at position i skips $j \cdot S_i$ scores, so

$$j = \left\lfloor \frac{k - \text{rank}}{S_i} \right\rfloor, \qquad \text{rank} \mathrel{+}= j \cdot S_i.$$

Advance position by position until rank $= k$; the chosen candidates spell out the answer.

1's For All

Problem Statement
Minimise the number of additions, multiplications, and concatenations needed to obtain a number, all applied to the digit 1.

Solution — Dynamic Programming
Let $f(k)$ be the minimum number of 1's to build k. Compute f for all k up to the input by combining smaller results:

- **Concatenation:** $f(k) = \min f(a) + f(b)$ where k is a followed by the digits of b.
- **Multiplication:** $f(k) = \min f(d) + f(k/d)$ over divisors d.
- **Addition:** $f(k) = \min f(a) + f(k - a)$.

Key Insight

The bottleneck is the addition operation: naively it tries all splits $a + (k - a)$, costing $O(k)$ per value and $O(n^2)$ overall. We can prune it.

Empirically the maximum of f over $k \leq 100000$ is only 18. So for an additive split to be optimal, $f(a) + f(k - a) < 18$, meaning at least one of $f(a)$, $f(k - a)$ is at most 9. Keep a set $S = \{x : f(x) \leq 9\}$ and only try additions with one operand drawn from S. For $k \leq 100000$, $|S|$ is only a couple thousand, making the whole DP fast enough.

Alchemist's Numbers

Alchemist's Numbers

Problem Statement
Given a list of numbers, in one operation you may multiply or divide a single number by a prime. Find the minimum total number of operations needed to make all the numbers equal (to some common target you choose).

Key Insight
The fundamental theorem of arithmetic states that every integer greater than 1 can be uniquely represented as a product of prime numbers.

$$A = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k}$$

Suppose the final target is $T = p_1^{t_1} \cdot p_2^{t_2} \cdot \ldots \cdot p_k^{t_k}$, then the number of operations needed to get from A to T is $|a_1 - t_1| + |a_2 - t_2| + \ldots + |a_k - t_k|$. Therefore, we can find the prime factorization of each number and then find the median of the exponents for each prime. The target T will be the product of the primes raised to their respective median exponents.

Example

Consider the numbers $[12, 24, 58, 99]$, their prime factorizations are:

$$12 = 2^2 \cdot 3^1 \cdot 11^0 \cdot 29^0$$
$$24 = 2^3 \cdot 3^1 \cdot 11^0 \cdot 29^0$$
$$58 = 2^1 \cdot 3^0 \cdot 11^0 \cdot 29^1$$
$$99 = 2^0 \cdot 3^2 \cdot 11^1 \cdot 29^0$$

The median of the exponents for 2 is 1.5, for 3 is 1, for 11 is 0.5, and for 29 is 0.5. It doesn't matter if we round up or down for the median.

Therefore, the target number is $T = 2^1 \cdot 3^1 \cdot 11^0 \cdot 29^0 = 6$. The total number of operations to get to T:

$$
\begin{aligned}
12 \to 6: &\quad |2-1| + |1-1| + |0-0| + |0-0| &&= 1 \\
24 \to 6: &\quad |3-1| + |1-1| + |0-0| + |0-0| &&= 2 \\
58 \to 6: &\quad |1-1| + |0-1| + |0-0| + |1-0| &&= 2 \\
99 \to 6: &\quad |0-1| + |2-1| + |1-0| + |0-0| &&= 3
\end{aligned}
$$

Therefore, the total number of operations needed is $1 + 2 + 2 + 3 = 8$.

Naive Solution

1. Use sieve of Eratosthenes to find all primes up to maximum number in the input.
2. factorize each number using the primes and store the exponents in a matrix
3. Find the median of exponents and find the sum of the absolute deviation from the median.
4. Add the sums together to get the final answer.

This is likely too slow and takes too much memory, since most of the prime will not be used, it is better to store it in a sparse dictionary.

Optimized Solution

1. For each number, find its prime factors and their exponents. Keep it in a dictionary `num`. `num[p][i]` represents how many numbers contain p^i as a factor. Remember to include the case where p^0 is a factor, which means that the number does not contain p as a factor.
2. Find the median by iterating over `num[p]`. An efficient algorithm to find the median is to keep a running count of how many numbers have been accounted for as we iterate through the exponents. Once we reach the point where the count exceeds half of the total numbers, we have found our median exponent for that prime.
3. Compute the cost of converting each number to the target by iterating over `num[p]` again and summing up the absolute differences from the median exponent.

Network Protection

Problem Statement
Malware starts at server a; the analyst's backup is at server b. A server u is **recoverable** if the analyst can travel from u to b along some route, always arriving at every server on that route *strictly before* the malware does. Sum the data sizes c_u over all recoverable u.

Setting up the distances
Let $d(x, y)$ be the shortest travel time between servers x and y.

The key claim

u is recoverable $\iff d(u, b) < d(a, b)$

So the answer is simply: run **one Dijkstra from** b, then keep every server u whose distance to b is less than the distance from a to b.

Why?

There are many ways to prove this, but the easiest way (I think) is that since the analyst must reach b, the optimal strategy for the malware is to get to b as quickly as possible and stay there. You can then trace backwards and see that this is true for every server u.

Formal Proof

The recoverable criteria is that along all nodes x from u to b, the following holds:

$$d(u, x) < d(a, x)$$

Add $d(x, b)$ to both sides:

$$d(u, x) + d(x, b) < d(a, x) + d(x, b)$$

Since x is on the shortest path from u to b, we have $d(u, x) + d(x, b) = d(u, b)$. So:

$$d(u, b) < d(a, x) + d(x, b)$$

Recall the triangular inequality: $d(a, b) \leq d(a, x) + d(x, b)$. So we have:

$$d(u, b) < d(a, b)$$

What if $d(a, b) \leq d(u, b)$? Then the malware can just go to b and wait, and the

Downsizing

Downsizing

Problem Statement
A convex polygon lies outside a circle of radius r centred at O. Each point P is mapped to P' on ray OP with $|OP| \cdot |OP'| = r^2$ (*circular inversion*). Find the area of the transformed shape.

Key Insight
Translate so O is the origin; then $P' = \frac{r^2}{|P|^2} P$. A straight line *not* through O inverts to a **circle through** O, so every polygon edge becomes a **circular arc**. The image is a region bounded by n arcs.

Approach 1 — Numerical Approximation

Subdivide and Shoelace

1. Walk around the polygon and **subdivide** every edge into many small segments, producing a dense list of boundary points.
2. **Transform** each point by $P' = \frac{r^2}{|P|^2} P$.
3. Compute the area of the resulting (curved) region with the **shoelace formula** over the transformed points.

With enough subdivisions the polygonal approximation of the arcs converges well within the required 10^{-6} error. Simple, but needs many points.

Approach 2 — Exact: Edges to Arcs

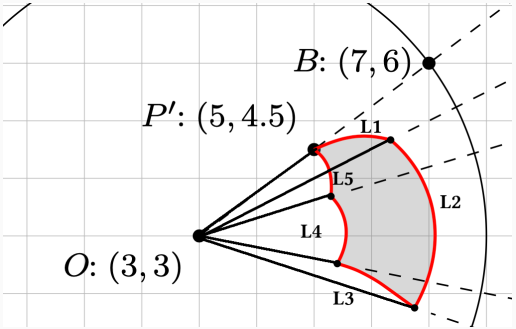
Each edge $A \to B$ lies on a line at distance d from O with unit normal $\hat{n}$. Its image is an arc of the circle

$$\text{centre } C = \frac{r^2}{2d}\,\hat{n}, \qquad R = \frac{r^2}{2d} = |C|$$

(it passes through O). The arc runs between the image endpoints

$$A' = \tfrac{r^2}{|A|^2} A, \qquad B' = \tfrac{r^2}{|B|^2} B.$$

The region is the union of these arcs, one per edge.

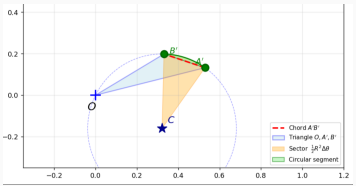


Approach 2 — Area from Sectors & Segments

Sum the signed area swept from O over each arc (Green's theorem). For one directed arc $A' \to B'$ on circle C:

$$\underbrace{\triangle(O, B', A')}_{\text{chord part}} + \underbrace{\left[\,\text{sector}_C(A', B') - \triangle(C, B', A')\,\right]}_{\text{circular segment}}$$

with the sector area $\frac{1}{2}R^2\alpha$ ($\alpha = \angle A'CB'$). Signs follow the CCW orientation automatically.



Result

Add the contribution of all n arcs to get the exact area. (Special case: if an image circle is centred at O, the contribution is just the sector $A' \to B'$.)

Chain Reaction

Chain Reaction

Problem Statement
Given a sequence of n samples, each with a value v, and q queries, each specifying an interval $[l_j, r_j]$, determine how many samples remain after cancellation.

Naive Approach
For each query, simulate the reaction process by maintaining a list of samples in the reaction line. This approach has a time complexity of $O(n^2)$ per query, which is too slow for large inputs as there are many queries. The key challenge is to optimise the brute-force solution.

Optimisations

The first optimization is to precompute the next twin. This can be done in a right to left sweep in $O(n)$. For each query, we can jump to the next twin and skip all the samples in between.

However, given that the maximum value for a is small relative to n, we will get a lot of cancellations by pigeon hole principle. Suppose there are chains of reactions, where a block cancels, immediately exposing the start of another cancellable block, and so on. We can precompute the next twin of the next twin, and so on, using a technique called binary lifting. This allows us to jump over multiple blocks in $O(\log n)$ time.

Create a table $\mathrm{next}[i][j]$ where $\mathrm{next}[i][j]$ is the index reached after jumping to the next twin 2^j times in a row, starting from index i. This can be computed in $O(n \log n)$ time.

Greedy Scan Example

$$A = \{1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2\}$$

i	0	1	2	3	4	5	6	7	8	9	10	11
$A[i]$	1	2	1	2	1	2	1	2	1	2	1	2
$\text{next}[i]$	2	3	4	5	6	7	8	9	10	11	12	12

Binary Lift Example

Define a 2D table

$$\mathrm{nxt}[i][k] \;=\; \text{position reached after exactly } 2^k \text{ reactions starting at } i$$

Recurrence:

$$\mathrm{nxt}[i][0] = \mathrm{nxt}[i], \qquad \mathrm{nxt}[i][k] = \mathrm{nxt}\big[\mathrm{nxt}[i][k-1]+1\big][k-1] \quad (k \geq 1)$$

Reason: after 2^{k-1} reactions we stand at $\mathrm{nxt}[i][k-1]$. The next reaction starts at $\mathrm{nxt}[i][k-1]+1$, then we take 2^{k-1} more reactions from there.

Binary Lift Example (cont.)

Columns are $\mathrm{nxt}[i][0..3]$ for $A = \{1, 2, \ldots\}$:

i	[0]	[1]	[2]	[3]	i	[0]	[1]	[2]	[3]
0	2	5	11	12	6	8	11	12	12
1	3	6	12	12	7	9	12	12	12
2	4	7	12	12	8	10	12	12	12
3	5	8	12	12	9	11	12	12	12
4	6	9	12	12	10	12	12	12	12
5	7	10	12	12	11	12	12	12	12

Terrain Forming

Terrain Forming

Problem Statement
Given a list of increasing elevations, find the minimum possible variance after performing a series of operations.

Key Insight
Each operation only *reorders* the gaps between consecutive values — the multiset of differences is preserved, so we are free to arrange those gaps in any order.

Let the values be $[1, 2, 5, 8, 10]$. The difference array is $[1, 3, 3, 2]$. Suppose we perform the operation on the second number, we get $[1, 4, 5, 8, 10]$. The new difference array is $[3, 1, 3, 2]$ — the same gaps $\{1, 2, 3, 3\}$, just reordered.

Another thing to note is that the value to output is $n^2 D = n \sum_{i=1}^{n} a_i^2 - \left(\sum_{i=1}^{n} a_i\right)^2$

Brute Force?

With the insight, we can brute force all possible permutations of the difference array. However, the number of permutations is $n!$, which is too large for $n = 10^4$. We need to find a better way to find the minimum variance.

The Valley?

Intuitively, the variance is minimized when the values are as close to the mean as possible. This means the difference array at both ends should be large, and becoming near 0 towards the middle, creating a U shape (You can prove this but not necessary).

Therefore, we can sort all differences into non-decreasing order. Because they're sorted, you can build the valley by inserting differences from smallest to largest, choosing at each step whether the current difference goes on the left end or the right end. This greedy insertion order guarantees a valid valley shape.

DP to the rescue

Build the valley with DP. Begin with array [0], tracking the sum s and sum of squares s'. Each step has two operations:

- Left-insert: every element increases by d_i, so $s \mathrel{+}= d_i n$, $s' \mathrel{+}= 2s\, d_i + i\, d_i^2$ (from $(a+b)^2$).
- Right-insert: only the last element increases by d_i, so $s \mathrel{+}= d_i$, $s' \mathrel{+}= pre_i^2$, where pre_i is the prefix sum of d.

Let $dp[i][j]$ be the minimum s' using i elements summing to j. The transitions to $i+1$ elements are:

- $dp[i+1][j+d_i i] = dp[i][j] + 2jd_i + i\, d_i^2$
- $dp[i+1][j+pre_i] = dp[i][j] + pre_i^2$

The final answer is $\min_j(n \cdot dp[n][j] - j^2)$.

Example

Take $a = [1, 2, 4, 6]$, so $d = [1, 2, 2]$ (already sorted) and $pre = [1, 3, 5]$. Each step lists the reached j and the stored value; L = left-insert, R = right-insert.

- Step 1 ($d_1 = 1$), from $dp[1][0] = 0$: L j=1, v=1; R j=1, v=1.
- Step 2 ($d_2 = 2$), from $dp[2][1] = 1$: L j=5, v=13; R j=4, v=10.
- Step 3 ($d_3 = 2$): from $dp[3][4] = 10 \Rightarrow dp[4][10] = 38$, $dp[4][9] = 35$; from $dp[3][5] = 13 \Rightarrow dp[4][11] = 45$, $dp[4][10] = 38$.

(e.g. Step 2 L: $j = 1 + 2 \cdot 2 = 5$, $v = 1 + 2 \cdot 1 \cdot 2 + 2 \cdot 2^2 = 13$.)

$$\text{Answer} = \min(4 \cdot 45 - 11^2,\ 4 \cdot 38 - 10^2,\ 4 \cdot 35 - 9^2) = 52.$$

Hacky Solution

You can also approximate this. Notice the maximum value a_i is relatively small compared to n, meaning the difference array contains a lot of zeros. Therefore, one can use simulated annealing or other randomised methods to generate a candidate arrangement, repeat many times, and output the best one found.