# TOKI Regular Open Contest #4 Editorial

*(English version is available on page 6.)*

---

## Penulis soal

| | | |
|---|---|---|
| A | Bukan Kelipatan | **Matematika** |
| B | Gunting-Batu-Kertas | **Matematika** |
| C | Tumpukan Kartu | **Klungs & sergiovieri** |
| D | Robot Pemotong Rumput | **athin** |
| E | Dua Titik dan Lingkaran | **athin** |
| F | GCD Path | **Klungs** |

---

## A. Bukan Kelipatan

Selalu terdapat X dari range [2, 100] sehingga X tidak habis dibagi oleh A, B, ataupun C. Sebagai contoh, kita dapat mengambil bilangan prima X yang berbeda dari A, B, dan C, yang menyebabkan X tidak habis dibagi A, B, ataupun C. Ada terdapat lebih dari 3 bilangan prima di range [2, 100], sebagai contoh: 2, 3, 5, 7. Kita dapat simpulkan bahwa jawaban X selalu ada di antara 4 bilangan tersebut.

Salah satu solusi sederhana lainnya adalah melakukan pengulangan dari 2 sampai 100 untuk memeriksa apakah bilangan tersebut tidak habis dibagi oleh A, B, ataupun C.

**Kompleksitas waktu**: O(1)

---

## B. Gunting-Batu-Kertas

Misalkan g1, b1, k1 adalah banyak ‘G’, ‘B’, ‘K’ di barisan 1 dan g2, b2, k2 adalah banyak ‘G’, ‘B’, ‘K’ di barisan 2. Karena kita dapat melakukan permutasi kartu-kartu pada barisan kedua, maka

kita dapat dengan bebas memasangkan kartu-kartu dari barisan pertama dengan barisan kedua.

Perhatikan bahwa, dari tiga kemungkinan pasangan yang memberikan poin, komponen kartu yang digunakan secara keseluruhan berbeda semua. Maka kita dapat secara *greedy* memasangkan sebanyak-banyaknya ‘G’ dari barisan 1 dengan ‘K’ dari barisan 2, membentuk sebanyak minimum(g1, k2) pasangan, dan kartu yang tersisa tidak dapat digunakan untuk pasangan jenis lain yang memberikan poin. Begitupun juga untuk 2 kemungkinan pasangan kartu lain yang memberikan poin. Jawaban akhirnya adalah minimum(g1, k2) + minimum(b1, g2) + minimum(k1, b2).

Kompleksitas algoritma ini adalah O(N) untuk menghitung nilai dari g1, b1, k1, g2, b2, dan k2 dengan mengecek setiap kartu pada kedua baris.

**Kompleksitas waktu**: O(N)

---

# C. Tumpukan Kartu

Didefinisikan f(P) = banyaknya tumpukan yang dihasilkan oleh sebuah permutasi kartu P.

Perhatikan bahwa apabila f(P) = X, maka f(reverse(P)) = N + 1 - X.
Maka, f(P) + f(reverse(P)) = X + N + 1 - X = N + 1 untuk semua permutasi P.

Perhatikan juga bahwa tidak ada dua permutasi berbeda P1 dan P2 sehingga reverse(P1) = reverse(P2). Jadi, $\sum_P f(P) = \sum_P f(reverse(P))$.

Nilai harapan dari banyaknya tumpukan adalah jumlah semua banyaknya tumpukan dari setiap permutasi, dibagi dengan banyaknya permutasi, yaitu N!.

Jumlah semua banyaknya tumpukan dari setiap permutasi =

$$\sum_P f(P) = \frac{\sum_P f(P)+\sum_P f(P)}{2} = \frac{\sum_P f(P)+\sum_P f(reverse(P))}{2} = \frac{\sum_P (f(P)+f(reverse(P)))}{2} = \frac{\sum_P (N+1)}{2} = \frac{(N+1)\times N!}{2}$$

Maka, nilai harapan = $\frac{\sum_P f(P)}{N!} = \frac{(N+1)\times(N!/2)}{N!} = \frac{N+1}{2}$

**Kompleksitas waktu**: O(1)

---

# D. Robot Pemotong Rumput

Kita dapat mencari tahu titik-titik mana robot memulai memotong rumput (melakukan langkah 2). Perhatikan bahwa titik-titik mulai tersebut (relatif terhadap titik awal) adalah:

- Setengah lingkaran (N/2)
- N/2 + K + N/2
- N/2 + K + N/2 + K + N/2
- …

Jarak antar dua titik mulai yang berurutan adalah K + N/2.

Pada suatu saat, titik mulai akan kembali lagi ke titik awal. Apabila seluruh titik mulai di-plot, perhatikan bahwa jarak antar 2 titik bersebelahan pada keliling lingkaran adalah FPB(K + N/2, N).

Setelah mengetahui setiap titik mulai, artinya robot dapat memotong sejauh K meter dari titik-titik tersebut. Sehingga, seluruh rumput terpotong jika dan hanya jika K ≥ FPB(K + N/2, N). FPB dapat dihitung menggunakan Euclid dengan kompleksitas O(log(min(K + N/2, N))) ~ O(log(N)).

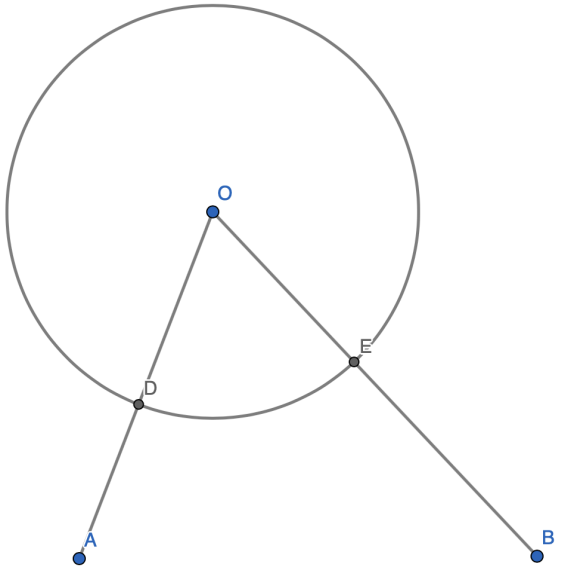
Perhatikan bahwa berhubung N bisa berupa bilangan ganjil, kita dapat mengalikan K dan N masing-masing dengan 2. Hal ini tidak akan mengubah solusi.

**Kompleksitas waktu**: O(T * log(N))

---

# E. Dua Titik dan Lingkaran

Apabila garis dari A ke B melewati lingkaran, maka jawabannya adalah panjang garis tersebut. Apabila tidak, maka perhatikan ilustrasi berikut.




Kita dapat selalu memilih titik O sebagai kandidat titik C. Namun, ada titik-titik lain yang lebih optimal sebagai titik C yakni titik pada juring ODE. Dapat dibuktikan bahwa titik C pasti berada di sisi lingkaran, sehingga titik C berada pada tembereng DE.

Apabila grafik panjang yang dibutuhkan untuk suatu titik C dari D hingga E di-plot, maka grafik tersebut berbentuk konveks cekung bawah. Sehingga, kita dapat melakukan Ternary Search di antara D dan E untuk menentukan titik C yang optimal. Kompleksitas total adalah $\log_{3/2}$ dari suatu konstanta 1.000.000 sehingga O(1).

**Kompleksitas waktu**: O(1)

---

# F - GCD Path

Soal ini dapat diselesaikan dengan pengerjaan secara offline.

Perhatikan bahwa jawaban untuk query ke-i pasti merupakan faktor dari gcd(W[A[i]], W[B[i]]). Misalkan G[X] menyatakan subgraf dari graf masukan yang mana suatu node Y terdapat di G[X] jika dan hanya jika X adalah faktor dari W[Y]. Perhatikan bahwa apabila X merupakan kandidat solusi untuk query ke-i, maka A[i] dan B[i] terhubung di G[X].

Kita dapat membuat G[X] dengan mencatat tiap node Y yang mana W[Y] merupakan kelipatan X (X, 2X, 3X dst). Selanjutnya, tiap edge dapat dicari dengan melakukan looping terhadap tiap tetangga Y yang bobotnya juga habis dibagi X. Perhatikan untuk node ke-i akan dilakukan

looping sebanyak banyak faktor W[i], yang mana untuk batasan di soal, banyak faktor maksimum hanya sekitar 100.

Setelah membuat G[X], cukup periksa untuk setiap query yang mana X adalah faktor dari gcd(W[A[i]], W[B[i]]). Lalu cek, apakah kedua node tersebut terhubung atau tidak. Kita dapat menggunakan Disjoint Set untuk hal tersebut.

Untuk mencari jawaban setiap query, lakukan hal di atas untuk tiap X dari 1 sampai nilai maksimum W[i].

**Kompleksitas waktu**: O((N + M + Q) * MAX_BANYAK_FAKTOR)

# TOKI Regular Open Contest #4 Editorial

---

## Problem authors

| | | |
|---|---|---|
| A | Not a Multiple | **Matematika** |
| B | Rock-Paper-Scissors | **Matematika** |
| C | Stack of Cards | **Klungs & sergiovieri** |
| D | Grass-Cutting Robot | **athin** |
| E | Two Points and Circle | **athin** |
| F | GCD Path | **Klungs** |

---

## A. Not a Multiple

There is always X from range [2, 100] such that X is not divisible by A, B, or C. For example, we can take another prime X different from A, B, and C, thus X is not divisible by A, B, or C. There are more than 3 prime numbers in the range [2, 100], i.e. 2, 3, 5, 7. We can conclude that the answer X always exist. One simple solution is to do a for-loop from 2 to 100 to check whether the particular number is not divisible by A, B, or C.

**Time complexity**: O(1)

---

## B. Rock-Paper-Scissors

Let g1, b1, k1 be the number of ‘G’, ‘B’, ‘K’ in first row respectively and g2, b2, k2 be the number of ‘G’, ‘B’, ‘K” in second row respectively. Since we are allowed to permute the cards in the second row, we are able to freely match cards from the first row to cards from the second row. Observe that, from the three possible pairings which give a point, they altogether use different kind of cards. We can greedily pair as much as possible ‘G’ from the first row and ‘K’ from the second row, creating minimum(g1, k2) pairs, and the remaining cards cannot be use to create any other kind of pairing that give a point. This apply to the other two possible pairing that give a point. The final answer is minimum(g1, k2) + minimum(b1, g2) + minimum(k1, b2). The

complexity of the algorithm is O(N) since we need to scan every cards to compute the value of g1, b1, k1, g2, b2, k2.

**Time complexity**: O(N)

---

# C. Stack of Cards

Define f(P) = number of stacks produced by a card permutation P.

Observe that if f(P) = X, then f(reverse(P)) = N + 1 - X.
Then, f(P) + f(reverse(P)) = X + N + 1 - X = N + 1 untuk semua permutasi P.

Also observe that there are no two different permutation P1 and P2 such that reverse(P1) = reverse(P2). Hence, $\sum_P f(P) = \sum_P f(reverse(P))$.
The expected value of number of stacks is the sum of number of stacks produced by each permutation, divided by the number of permutation, which is N!.

The sum of number of stacks produced by each permutation =

$$\sum_P f(P) = \frac{\sum_P f(P) + \sum_P f(reverse(P))}{2} = \frac{\sum_P (f(P) + f(reverse(P)))}{2} = \frac{\sum_P (N+1)}{2} = \frac{(N+1) \times N!}{2}$$

Then, the expected value = $\frac{\sum_P f(P)}{N!} = \frac{(N+1) \times (N!/2)}{N!} = \frac{N+1}{2}$

**Time complexity**: O(1)

---

# D. Grass-Cutting Robot

We can find out the points from which the robot starts to cut the grass (the 2nd command). Observe that the points (relative to the initial point) are:

- Semicircle (N/2)
- N/2 + K + N/2
- N/2 + K + N/2 + K + N/2
- …

The distance between two consecutive starting point are K + N/2.

At one point of time, the starting point will be the initial point again. If we plot every starting points, the distance between two consecutive starting points on the circle is GCD(K + N/2, N) where GCD is the greatest common divisor.

After knowing every starting points, we can see that the robot cut grass for K meter from every starting points. Therefore, all the grass are cut if and only if K >= GCD(K + N/2, N). The value of the GCD can be computed using Euclid algorithm with complexity O(log(min(K + N/2, N))) ~ O(log(N)).

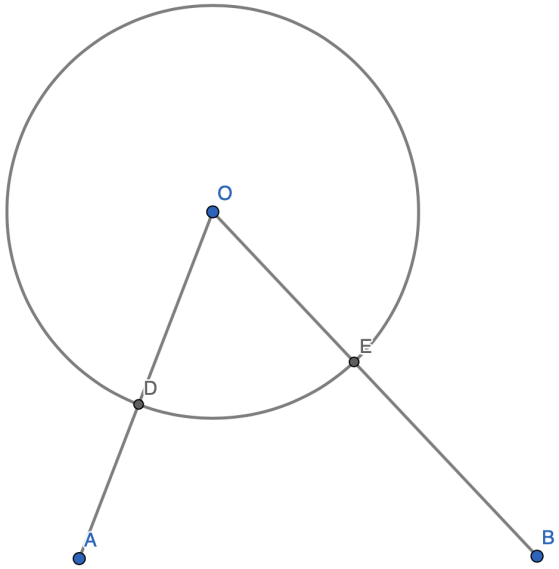
Observe that since N can be and odd integer, we can multiply both K and N with 2. This won't change the solution.

**Time complexity**: O(T * log(N))

---

## E. Two Points and Circle

If the segment from A to B cross the circle, then the answer is the length of that segment. If not, see the illustration below.




We can always pick O as a candidate for C. However, there are other points which is more optimal for C, which is the points on the arc DE. It can be proven that the optimal C must be on the arc DE.

If we plot the graph of the length needed for a point C in the arc DE, then the graph will be a convex graph with minimum optima. Hence, we can use Ternary Search in arc DE to find the optimal C. The complexity will be $\log_{3/2}$ from a constant 1.000.000, hence O(1).

**Time complexity**: O(1)

---

# F - GCD Path

This problem can be solved offline.

Observe that the answer for the i-th query must be a factor of gcd(W[A[i]], W[B[i]]). Let G[X] be a subgraph from the input graph where a node Y is in G[X] iff X is a divisor of W[Y]. Note that if X is a candidate solution for i-th query, then A[i] and B[i] will be connected in G[X].

We can construct G[X] by storing each node Y where W[Y] is a multiple of X (X, 2X, 3X, and so on). After that, each edge can be found by looping each neighbor of Y which weight is also divisible by X. Observe that for the i-th node we will do looping for each divisors of W[i], where for the given constraint, the maximum number of different divisors will be around 100.

After constructing G[X], just check for each query where X is a divisor of gcd(W[A[i]], W[B[i]]) whether they are connected or not. We can use Disjoint Set for that purpose.

To find the answer for each query, do the method described above for each X from 1 to the maximum value of W[i].

**Time complexity**: O((N + M + Q) * MAX_NUMBER_OF_DIVISORS)