OperateString

Consider the characters written around a circle, in the clockwise order. Consider an arrow A, initially pointing to the first character. The first operation moves A forward for X times. The second operation moves A backward (counter-clockwise) -X times.

So, keep the position of A and at last, print the string starting from A and going in clockwise order.

```
class OperateString:
	def operate(self, s, moves):
		print( len(s), len(moves), min(moves), max(moves), sum(moves) %
len(s) )
		shift = sum(moves) % len(s)
		return s[shift:] + s[:shift]
```

OneGcd

Consider Y is the power of two. It's easy to calculate the numbers in the range [X, X + Y) which are coprime with Y. Consider Y is in the form of 2^a * 3^b. Now the number of the numbers in the range [1, Y] which are coprime with Y is Y - Y / 2 - Y / 3 + Y / 6 (Take floor when dividing, just like C++). So the problem seems easy now. Consider P as the set of primes dividing Y (|P| is at most 4), the answer for the range [1, Y] is: $\sum_{S \subseteq P} (-1)^{|S|} \left\lfloor \frac{Y}{\prod_{x \in S} x} \right\rfloor$. As the range given is [X, X + Y - 1], the answer is: $\sum_{S \subseteq P} (-1)^{|S|} \left\lfloor \frac{X+Y-1}{\prod_{x \in S} x} \right\rfloor - \sum_{S \subseteq P} (-1)^{|S|} \left\lfloor \frac{X}{\prod_{x \in S} x} \right\rfloor$.

Overall time complexity is O(2^4 * N).

Bones: Y can be any number, not just divisible by primes less than 10. The solution to that would be using Phi(Y[i])

```
def phi(Y):
	answer = Y
	for p in [2, 3, 5, 7]:
		if Y % p == 0:
			answer = (answer * (p-1)) // p
	return answer

class OneGcd:
	def solve(self, X, Y):
		return [ phi(y) for y in Y ]
```

ArraySorting

It's easy to prove that the answer is at most N - 1. So, there is at least an element that doesn't change. Let dp[i] be the number of changes needed in the range [0, i) to make [0, i] sorted while the i-th element itself will not change.
While calculating dp[i], we need to find the last element like j in the range [0, i) such that j will not change.

$$dp[i] = \min(\min_{j<i, a[j] \le a[i]} i - j - 1 + dp[j], i)$$

Now, the answer could be calculated easily by fixing the last element which will not change. If we consider this element to be i, the answer will be n - i - 1 + dp[i].
The overall complexity is O(n^2). It's possible to solve the problem in O(n log n), find LIS and change the other elements.

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 2048;
int N;
int A[MAX];
int dp[MAX][MAX], dp2[MAX][MAX];

struct ArraySorting {
    vector<int> arraySort(vector<int> _A) {
        N = _A.size();
        int MAXVAL = *max_element( _A.begin(), _A.end() );
        for (int n=0; n<N; ++n) A[n] = _A[n];
        for (int current_index=N-1; current_index>=0; --current_index) {
            for (int current_value=MAXVAL; current_value>=0;
--current_value) {
                int this_edit = (A[current_index] == current_value ? 0 : 1);
                if (current_index == N-1) {
                    dp[current_index][current_value] =
dp2[current_index][current_value] = this_edit;
                    if (current_value < MAXVAL)
dp2[current_index][current_value] = min( dp2[current_index][current_value],
dp2[current_index][current_value+1] );
                } else {
                    dp[current_index][current_value] =
dp2[current_index][current_value] = this_edit +
dp2[current_index+1][current_value];
                    if (current_value < MAXVAL)
dp2[current_index][current_value] = min( dp2[current_index][current_value],
dp2[current_index][current_value+1] );
                }
```

```
            }
        }

      cout << "N = " << N << " approx edits " << dp[0][0] << endl;

        vector<int> B;
        int previous_value = 1;
        for (int n=0; n<N; ++n) {
            int best_edits = N+47, best_value = -1;
            for (int current_value=previous_value; current_value<=MAXVAL;
++current_value) {
                int current_edits = dp[n][current_value];
                if (current_edits < best_edits) { best_edits =
current_edits; best_value = current_value; }
            }
            B.push_back( best_value );
            previous_value = best_value;
        }
        return B;
    }
};
```

SubstringQueries

The first fact is for every Q(L), 0 <= Q(L) < |S| is satisfied.

Let str(i) = s[i], s[i + 1], …, s[n - 1], s[0], s[1], …, s[i - 1].

Sort str's in the lexicographically order (it's possible in O(|S| ^ 2 * log(|S|) easily, while using suffix array, O(|S| log(|S|) is achievable).

For now, consider k is infinitly large. Let's answer Q(L) when 1 <= L <= |S|.

Consider the first strings in the sorted order that their first L characters are equal. Find the minimum index between them, Q(L) is found now.

Consider L = |S|. If several strings in the sorted order are equal, say str(i), str(j), for each L, starting from i going L characters forward makes an equal string with starting from j.

So, for L > |S|, answer is same as answer for L = |S|.

Now, let's involve k. For each L, Q(L) + L <= |S| * k must be satisfied. As we have Q(L) < |S|, for L <= k * (|S| - 1), there is no problem. After that, we should delete inappropriate indices from our list. For example, when L = k * (|S| - 1) + 1, Q(L) is not |S| - 1 for sure (because we can't have string with such length starting at |S| - 1). So, delete |S| - 1 from the list. Go on and find the answer for the last |S| values of Q.

Using suffix array, O(|S| log |S|) is possible. But with the naive method, the overall complexity is now O(|S| ^ 2 log |S|).

```
#include<bits/stdc++.h>

#define pb push_back
using namespace std;

class SubstringQueries
{
    public:
        static const int maxN = 5002;
        static const long long mo1 = 1e9 + 7 ;
        static const long long mo2 = 1e9 + 9 ;
        static const long long b1 = 31;
        static const long long b2 = 71;

        int best[maxN];
        int bestL[maxN];
        int n;
        long long hes1[maxN], hes2[maxN], d1[maxN], d2[maxN];

        int check(int x, int y, string &t)
        {
             for (int i=0;i<n;i++)
                if (t[x+i]<t[y+i]) return x; else
                if (t[x+i] > t[y+i]) return y;

             return x;
        }

        int check_lex(int f, int s, int len, string &t)
        {
            int l = 0;
            int r = len + 1;

            while(l<r-1)
            {
                int mid = l+r>>1;

                long long val1f = (hes1[f+mid-1] - hes1[f-1])*d1[s-f]%mo1;
                if (val1f<0) val1f+=mo1;
                long long val1s = (hes1[s+mid-1] - hes1[s-1])%mo1;
                if (val1s<0) val1s+=mo1;
                if (val1f == val1s) l = mid; else r = mid;
```

```
        }

        if (r == len+1) return 0;
        return t[s + r -1]<t[f + r - 1];
    }

    void solve_smaller(string &t)
    {
        int m = t.size();

        d1[0] = 1;
        d2[0] = 1;

        for (int i =1;i<maxN;i++)
        {
            d1[i] = d1[i-1]*b1%mo1;
            d2[i] = d2[i-1]*b2%mo2;
        }

        for (int i=1;i<m;i++)
        {
            hes1[i] = (hes1[i-1] + (t[i]-'a')*d1[i])%mo1;
            hes2[i] = (hes2[i-1] + (t[i]-'a')*d2[i])%mo2;
        }

        for (int i=1;i<=n;i++){

         int cur = 1;
         for (int j=2;j<=min(m-i, n + 1);j++)
            if (check_lex(cur,j, i, t)) cur = j;

            bestL[i] = cur;
        }
    }

    vector<long long> solve(string s, long long k, vector<long long>
queries)
    {
         n = s.size();

         long long len = k*n;
```

```
            string t ="";
            if (k==1) t = "#" + s; else
               t = "#" + s + s;


            if (k>1){

            best[1] = 1;

            for (int i = 2;i<=n;i++)
               best[i] = check(best[i-1], i, t);
            }

            solve_smaller(t);

            vector<long long> res;

            for (int i:queries)
            {
                if (i<=n){
                   res.pb(bestL[i] - 1);
                   continue;
                }
                long long  poz = min(len - i +1, 1ll*n);
                res.pb(best[poz] - 1);
            }

           return res;
        }

};
```

ParadePlanner

Let the path be v1, v2, v3, v4, v5, v6. Let's fix the middle street, v3-v4. What we need now?

For v3, we need to count the number of possible pairs of (v1, v2), the other side, say (v5, v6) can be found similar.

Let P(v) be number of paths of length 2 starting from v. P(v) = sum for each u in adjacent list v, number of adjacents of u - 1.

Number of pairs (v1, v2) is P(v3) - (number of adjacents of v3 - 1).

So, we finished? No. We are counting each triangle 3 times, also squares, two times. There is another bad case. Consider a, b, c, d, and edges a-b, a-c, b-c, b-d. Subtract these values from the answer.

Complexity: O(n ^ 3).

```
    public long count(int N, int seed, int threshold, int[] toggle) {

        boolean[][] G = new boolean[N][N];

        for (int a=0; a<N; ++a) for (int b=0; b<N; ++b) G[a][b] = false;

        long state = seed;
        for (int a=0; a<N; ++a) for (int b=a+1; b<N; ++b) {
            state = (state * 1103515245 + 12345) % (1L << 31);
            if (state < threshold) G[a][b] = G[b][a] = true;
            if (N <= 10 && state < threshold) System.out.println("adding
edge "+a+" "+b);
        }

        for (int i=0; 2*i<toggle.length; ++i) {
            int a = toggle[2*i], b = toggle[2*i+1];
            G[a][b] = !G[a][b];
            G[b][a] = !G[b][a];
        }

        int[][] E = new int[N][];
        for (int a=0; a<N; ++a) {
            int deg = 0;
            for (int b=0; b<N; ++b) if (G[a][b]) ++deg;
            E[a] = new int[deg];
            for (int b=N-1; b>=0; --b) if (G[a][b]) E[a][--deg] = b;
        }

        long[] P = new long[N];
        for (int a=0; a<N; ++a) P[a] = 0;

        long triangles = 0;
        long triangleEdges = 0;
        long squares = 0;

        for (int a=0; a<N; ++a) for (int b=a+1; b<N; ++b) {
            int common = 0;
            for (int c=0; c<N; ++c) if (G[a][c] && G[b][c]) ++common;
            if (G[a][b]) {
```

```
                P[a] += E[b].length - 1;
                P[b] += E[a].length - 1;
            }
            squares += common * (common-1) / 2;
            for (int c=b+1; c<N; ++c) {
                if (G[a][b] && G[a][c] && G[b][c]) {
                    ++triangles;
                    triangleEdges += E[a].length + E[b].length + E[c].length
- 6;
                }
            }
        }

        long answer = 0;
        for (int a=0; a<N; ++a) for (int b=a+1; b<N; ++b) if (G[a][b]) {
            long c = P[a] - E[b].length + 1;
            long d = P[b] - E[a].length + 1;
            answer += c * d;
        }
        return 2*(answer - 3*triangles - 3*triangleEdges - 2*squares);
    }
```