Contents

1 2D Geometry

1.1 Primitives

```
typedef complex<double> point;
struct circle {
  point c; double r;
  circle(point c, double r):c(c),r(r){}
  circle(){}
};
double cross(const point &a, const point &b) {
  return imag(conj(a)*b);
}
double dot(const point &a, const point &b) {
  return real(conj(a)*b);
}
```

1.2 Intersections

```
// Line - Line
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Line - Segment
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Segment - Segment
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Circle - Line
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Circle - Segment
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Circle - Circle
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Line - Point
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Segment - Point
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

1.3 Circle Generation

```
// From 3 Points
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// From 1 Line 2 Points
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// From 2 Lines 1 Point
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// From 3 Lines
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

1.4 Heron Triangle Area

```
// Formula
// Formula
// Formula
```

1.5 Polygon Centroid

```
for(int i = 1; i < n-1; i++) {
  pt  ai = pts[i] - pts[i-1],
    ib = pts[i+1] - pts[i];
  area += (conj(ai)*ib).imag();
}
```

1.6 Point In Polygon

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

1.7 Convex Hull

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

1.8 Line Segment Set Intersection

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

1.9 Voronoi Diagrams

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

2 3D Geometry

2.1 Primitives

```
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
```

2.2 Convex Hull

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

2.3 Great Circle Distance

```
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
```

3 Combinatorics

3.1 Basics

```
// catalan numbers
long long C(int n) {
  return (C(n-1)*2*n*(2*n-1))/(n*(n+1));
  return NCR(2*n, n) - NCR(2*n, n+1);
  return NCR(2*n, n)/(n+1);
}
// derangements
long long D(int n) {
  return n*D(n-1) + pow(-1, n);
  return (n-1)*(D(n-1) + D(n-2));
}
// iterate over all subsets with < m elements
for (int i = 0; i < (1<<n); i=Integer.bitCount(i) < m ? i
    +1 : (i|(i-1))+1)
// iterate over all the subsets
for (int i=0; i < (1<<n); i++)
  // iterate over all the subsets of the i-th subset
  for(int i2 = i; i2 > 0; i2 = (i2-1) & i)
```

3.2 Permutation (Un)Ranking

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

3.3 Combination (Un)Ranking

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

4 Data Structures

4.1 Palindromic Tree

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

4.2 Treap

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

4.3 Sparse Array

```
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
```

4.4 Skip Lists

```
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
// Code
```

4.5 BIT + Search

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

4.6 Segment Tree + Lazy Propagation

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

4.7 Weighted Union Disjoint Sets

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

5 Game Theory

5.1 Nim Game

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

5.2 Grundy Numbers

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

5.3 General Josephus Problem

6 General Mathematics

6.1 Inclusion-Exclusion Patterns

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

6.2 Determinant

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

6.3 Gaussian Elimination

```
double* GaussianElimination(int N, double **mat) {
  int i, j, k, L; double t;
  for (i = 0; i < N - 1; i++) {
    L = i;
    for (j = i + 1; j < N; j++)
      if (fabs(mat[j][i]) > fabs(mat[L][i]))
        L = j;
    for (k = i; k <= N; k++)
    swap(mat[i][k], mat[L][k]);
    for (j = i + 1; j < N; j++)
      for (k = N; k >= i; k--)
        mat[j][k] -= (mat[i][k] * mat[j][i]) / mat[i][i];
  }
  double *res = new double[N];
  for (j = N - 1; j >= 0; j--) {
    for (t = 0.0, k = j + 1; k < N; k++)
    t += mat[j][k] * res[k];
    res[j] = (mat[j][N] - t) / mat[j][j];
  }
  return res;
}
```

6.4 Fast Fourier-Transform

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

6.5 Misc. Formulas

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7 Graph Theory

7.1 Primitives

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.2 Articulation Points & Bridges

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.3 SCC

```
#define MAXN 100

int N;
vector< vector<int> > adj;

int num[MAXN], low[MAXN];
bool vis[MAXN];
vector<int> S;
vector< vector<int> > SCCs;
int dfsNumber;

void tarjanSCC(int u) {
  low[u] = num[u] = dfsNumber ++;
  S.push_back(u);
  vis[u] = 1;
  for(int i=0; i<adj[u].size(); i++) {
    int v = adj[u][i];
    if(num[v] == -1)
      tarjanSCC(v);
    if(vis[v] == 1)
      low[u] = min(low[u], low[v]);
  }
  if(low[u] == num[u]) {
    vector<int> SCC;
    while(1) {
      int v = S.back();
      S.pop_back();
      vis[v] = 0;
      SCC.push_back(v);
      if(u == v)
        break;
    }
    SCCs.push_back(SCC);
  }
}

void findSCC() {
  dfsNumber = 0;
  memset(vis, 0, sizeof(vis));
  memset(num, -1, sizeof(num));
  memset(low, 0, sizeof(low));
  S.clear();
  SCCs.clear();
  for(int i=0; i<N; i++)
    if(num[i] == -1)
      tarjanSCC(i);
}
```

7.4 2-SAT

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.5 Edmonds-Karp Max Flow

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.6 Dinic's Max Flow

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.7 Min-Cost Max Flow

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.8 Euler Cycles

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.9 Maximum Matching

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.10 HL Decomposition

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.11 Modelling Inequalities

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.12 Max Flow Tricks

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.13 Bellman Ford

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.14 Stable Marriage

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

7.15 Maximum Assignment

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

8 Linear Programming

8.1 Simplex

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

9 Number Theory

9.1 Extended GCD

```
typedef pair<int, int> ii;
long long gcd( long long a, long long b )
{ return( b == 0 ? a : gcd( b, a % b ) ); }
//USED BY: egcd, msolve, inverse, ldioph
template< class Int > struct Triple {
  Int d, x, y;
  Triple(Int q, Int w, Int e):d(q), x(w), y(e){}
};//USED BY: msolve, inverse, ldioph
template< class Int > Triple< Int > egcd( Int a, Int b ) {
  if( !b ) return Triple< Int >( a, Int( 1 ), Int( 0 ) );
  Triple< Int > q = egcd( b, a % b );
  return Triple< Int >( q.d, q.y, q.x - a / b * q.y );
}
```

9.2 Modular Inverse

```
//solves ax = 1 (mod n), O( log(an) )
template< class Int > Int inverse(Int a, Int n){
  Triple< Int > t = egcd( a, n );
  if( t.d > Int( 1 ) ) return Int( 0 );
  Int r = t.x % n;
  return( r < Int( 0 ) ? r + n : r );
}
```

9.3 Modular Linear Equation

```
// solves ax = b (mod n), O( log(an) + gcd( a, n ) )
template<class Int> vector<Int> msolve(Int a,Int b,Int n){
  if( n < 0 ) n = -n;
  Triple< Int > t = egcd( a, n );
  vector< Int > r;
  if( b % t.d ) return r;
  Int x = ( b / t.d * t.x ) % n;
  if( x < Int( 0 ) ) x += n;
  for( Int i = 0; i < t.d; i++ )
    r.push_back( ( x + i * n / t.d ) % n );
  return r;
}
```

9.4 Linear Diophantine Equation

```
/* Solves ax + by = c. If .d == 0 -> no Solutions.
   Otherwise:
            x = t.x + k * b / t.d,
            y = t.y - k * a / t.d;        */
template<class Int> Triple<Int> ldioph(Int a,Int b,Int c){
  Triple< Int > t = egcd( a, b );
  if( c % t.d ) return Triple< Int >( 0, 0, 0 );
  t.x *= c / t.d; t.y *= c / t.d;
  return t;
}
```

9.5 Sieve of Eratosthenes

```
/*(simple, slow version) O( N.log(N) )*/
void sieve(bool prime[], int N){
  memset( prime, -1, N * sizeof( prime[0] ) );
  prime[0] = prime[1] = false;
  int sqrtN = ( int )sqrt( ( double )N );
  for(int i = 2; i <= sqrtN; i++) if(prime[i]){
    for( int j = i * i; j < N; j += i )
    prime[j] = false;
  }
}/*(fast, memory efficient version)
 * gP(n) is non-zero iff n is prime.
 * Requires N / 16 bytes of memory.
 * WARNING! Only works for odd numbers.*/
#define N 51000000
unsigned int prime[N / 64];
#define gP(n) (prime[n>>6]&(1<<((n>>1)&31)))
#define rP(n) (prime[n>>6]&=~(1<<((n>>1)&31)))
void sieve() {
  memset( prime, -1, sizeof( prime ) );
  unsigned int i, i2, j;
  sqrtN = (unsigned int)sqrt((double)N)+1;
  for(i = 3; i < sqrtN; i += 2 ) if(gP(i)) {
    i2 = i + i;
    for(j = i*i;j < N;j+=i2)rP(j);
  }
}
```

9.6 Primality Testing & Factoring

```
vector<ii> factor(long long N) {
  vector<ii> res;
  for(int i = 0, j = 0; primes[i]*primes[i] <= N; i++, j =
       0) {
    while(N % primes[i] == 0)
      j++, N /= primes[i];
    if(j) res.push_back(ii(primes[i],j));
  }
  return res;
}
```

9.7 Euler Phi

```
/* num of +ve ints < than n relatively prime to n. */
int phi(int n){
  vector< ii > p = factor(n);
  for( int i = 0; i < ( int )p.size(); i++ )
    n /= p[i].first, n *= p[i].first - 1;
  return n;
}
```

9.8 Continued Fractions of Rationals

```
/*O( log n ) 1 + 1/x*/
void contFract(int m, int n, vector<int> &ans){
  while( n )
    ans.push_back( m / n ),
    m %= n, m ^= n ^= m ^= n; // swap(m, n)
}
```

9.9 Chinese Remainder

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

9.10 Discerete Logarithm

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

9.11 Tortoise & Hare

```
// mu = start of cycle, lambda = cycle length
ii floyd(int x0) {
 int tortoise = f(x0), hare = f(f(x0));
 while(tortoise != hare)
   tortoise = f(tortoise), hare = f(f(hare));
 int mu = 0; hare = x0;
 while(tortoise != hare)
   tortoise = f(tortoise), hare = f(hare), mu++;
 int lambda = 1; hare = f(tortoise);
 while(tortoise != hare)
   hare = f(hare), lambda++;
 return ii(mu, lambda);
}
```

9.12 Pollard Rho

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

10 Search

10.1 Binary Search

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

10.2 Ternary Search

```
long double min() {
    long double lo = -1e6, hi = 1e6, res = 3e6;
    while(fabs(lo-hi) > EPS) {
        long double left = (hi-lo)/3 + lo, right = (2*(hi-
             lo))/3 + lo;
        long double resL = F(left), resR = F(right);
        if(resL < resR)
            hi = right;
        else
            lo = left;
        res = min(res, min(resL, resR));
    }
    return res;
}
```

11 Strings

11.1 Aho Corasick

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

11.2 Hashing

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

11.3 Z-Algorithm

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

11.4 KMP + Periods

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

11.5 Manacher

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

11.6 Suffix Array

```
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
// Algorithm
```

12 Misc

13 Last Page

Cut this paper out. Use it wisely!

Problem	Tags
01 A	
02 B	
03 C	
04 D	
05 E	
06 F	
07 G	
08 H	
09 I	
10 J	
11 K	
12 L	
13 M	

Time	Meeting Description	Chk
030	All Problems Read. Write Tags.	
060	Ace Decided. Choose Coder.	
090	Decide & Order Solveable Problems	
120	Status Check	
150	Status Check	
180	Status Check	
210	Status Check	
240	Blind Hour. One Problem.	
270	Status Check	
300	Contest Ends	

Solving A Problem

Read the statement carefully.
Break the problem down into pieces.
Plan the solution to each piece.
Think of corner cases to solution.
Calculate Complexity.
Simplify the solution.
Write steps of solution on paper.
Estimate Coding time.

During The Contest

Stay **calm** and **focused** or you and your team mates won't make it
If you don't understand it, it doesn't mean it's hard. Tell your team mate.
READ THE STATEMENT AGAIN. TELL YOUR TEAMMATE!
Write **significant** tags! Think how all the topics might come in use!
Run over the index page. Maybe one of the topic titles will inspire a solution!

Common Bugs

Add eps to double before getting floor or round