Contents

1 DFS

```java
import java.util.*;
class DFS {
  /**Does a DFS from s and returns the pre-, post- and connected component-values
   * O(m+n)
   * Properties:
   * Vertex v is reachable from start iff post[v] != -1 && post[v] < post[start]
   * Cycle in directed graph iff pre[v] < pre[u] < post[v] for some edge (u, v)
   * Vertices ordered by decreasing post-value gives a topological sort
   * @param G     Graph represented by adjacency list
   * @param pre   [n] matrix filled with each vertex' pre-value (init to -1)
   * @param post  [n] matrix filled with each vertex' post-value (init to -1)
   * @param cc    [n] matrix filled with each vertex' connected component-number (init to -1)
   * @param s     The vertex to search from (-1 will search for all connected components)*/
  static void dfs(ArrayList<ArrayList<Integer>> G, int[] pre, int[] post, int[] cc, int s) {
    int n = G.size();
    boolean allCC = s == -1;
    if(s == -1)
      s = 0;
    // Initialize the next values that can be assigned
    int num = 0;
    int ccnum = 0;
    for(int i = 0; i < n; ++i) {
      num = Math.max(num, post[i]+1);
      ccnum = Math.max(ccnum, cc[i]+1);
    }
    LinkedList<Integer> Q = new LinkedList();
    while(s != -1) {
      Q.push(s);
      while(!Q.isEmpty()) {
        int a = Q.peek();
        cc[a] = ccnum;
        if(pre[a] == -1) {
          pre[a] = num++;
          for(int b : G.get(a))
            if(pre[b] == -1)
              Q.push(b);
        } else {
          if(post[a] == -1)
            post[a] = num++;
          Q.pop();
        }
      }
      s = -1;
      if(allCC) {
        // Find next vertex to start search from
        ++ccnum;
        for(int i = 0; i < n && s == -1; ++i)
          if(post[i] == -1)
            s = i;
      }
    }
  }
}
```

2 SSSP Unweighted - BFS

```java
import java.util.*;
class BFS {
  /**Finds shortest path from s to every other vertex
   * O(m)
   * @param G Graph represented by adjacency list
   * @param P [n] array filled with each vertex' parent
   * @param s The vertex to start the search from
   * @return  [n] array with the distance from s to each vertex*/
  static int[] bfs(ArrayList<ArrayList<Integer>> G, int[] P, int s) {
    // Initialize distance array and parent array
    int[] D = new int[G.size()];
    Arrays.fill(D, Integer.MAX_VALUE);
    Arrays.fill(P, -1);
    D[s] = 0;
    LinkedList<Integer> Q = new LinkedList();
    Q.add(s);
    while(!Q.isEmpty()) {
      int a = Q.poll();
      for(int b : G.get(a))
        if(D[a]+1 < D[b]) {
          D[b] = D[a]+1;
          P[b] = a;
          Q.add(b);
        }
    }
    return D;
  }
}
```

3 SSSP Non-negative - Dijkstra

```java
import java.util.*;
class Dijkstra {
  /**Finds shortest path from s to all vertices in a graph with non-negatively weighted edges
   * Requires Pair
   * O((m+n)*log(n))
   * @param G Graph represented by adjacency list
   * @param P [n] array filled with each vertex' parent
   * @param s The vertex to start the search from
   * @return  [n] array with the distance from s to each vertex*/
  static int[] dijkstra(ArrayList<ArrayList<Pair>> G, int[] P, int s) {
    // Initialize distance array and parent array
    int[] D = new int[G.size()];
    Arrays.fill(D, Integer.MAX_VALUE);
    Arrays.fill(P, -1);
    D[s] = 0;
    PriorityQueue<Pair> Q = new PriorityQueue();
    Q.add(new Pair(s, D[s]));
    while(!Q.isEmpty()) {
      Pair a = Q.poll();
      for(Pair b : G.get(a.id))
        if(D[a.id]+b.v < D[b.id]) {
          D[b.id] = D[a.id]+b.v;
          P[b.id] = a.id;
          Q.add(new Pair(b.id, D[b.id]));
        }
    }
    return D;
  }
}
```

4 SSSP Arbitrary - Bellman-Ford

```java
import java.util.*;
class BellmanFord  {
  /**Finds shortest path from s to all vertices in a graph with arbitrarily weighted edges
   * Only explores paths reachable from s
   * O(n*m)
   * @param G Graph represented by adjacency list
   * @param P [n] array filled with each vertex' parent
   * @param s The vertex to start the search from
   * @return  [n] array with the distance from s to each vertex*/
  static int[] bellmanFord(ArrayList<ArrayList<Pair>> G, int[] P, int s) {
    int n = G.size();
    // Initialize distance array and parent array
    int[] D = new int[n];
    Arrays.fill(D, Integer.MAX_VALUE);
    Arrays.fill(P, -1);
    D[s] = 0;
    for(int i = 0; i < n-1; ++i)
      for(int u = 0; u < n; ++u)
        if(D[u] != Integer.MAX_VALUE)
          for(Pair a : G.get(u))
            if(D[u]+a.v < D[a.id]) {
              D[a.id] = D[u]+a.v;
              P[a.id] = u;
            }
    return D;
  }
  /**Checks whether a graph contains a negative cycle, as calculated by bellmanFord()
   * Checks for negative cycles in subgraph reachable from s, as calculated by bellmanFord()
   * O(m)
   * @param G Graph represented by adjacency list
   * @param D The [n] D matrix filled in by floydWarshall()
   * @return  Whether the graph contains a negative cycle or not*/
  static boolean hasNegativeCycle(ArrayList<ArrayList<Pair>> G, int[] D) {
    for(int u = 0; u < G.size(); ++u)
      if(D[u] != Integer.MAX_VALUE)
        for(Pair a : G.get(u))
          if(D[u]+a.v < D[a.id])
            return true;
    return false;
  }
}
```

5 Minimum Spanning Tree - Prim

```java
import java.util.*;
class Prim {
  /**Searches a graph for its minimum spanning tree
   * Allows for negative weights and disconnected graph
   * Requires Pair
   * O((m+n)*log(n))
   * @param G Graph represented by adjacency list
   * @return  An [n] array with each vertex' parent */
  static int[] prim(ArrayList<ArrayList<Pair>> G) {
    int n = G.size();
    // Initialize visited, parent and cost arrays
    boolean[] V = new boolean[n];
    int[] P = new int[n];
    int[] C = new int[n];
    Arrays.fill(P, -1);
    Arrays.fill(C, Integer.MAX_VALUE);
    PriorityQueue<Pair> Q = new PriorityQueue();
```


```
18      int count = 0;
19      int s = 0;
20      while(s != -1) {
21        Q.add(new Pair(s, 0));
22        while(!Q.isEmpty() && count < n) {
23          Pair a = Q.poll();
24          if(!V[a.id]) {
25            ++count;
26            V[a.id] = true;
27            for(Pair b : G.get(a.id))
28              if(!V[b.id] && C[b.id] > b.v) {
29                C[b.id] = b.v;
30                P[b.id] = a.id;
31                Q.add(new Pair(b.id, b.v));
32              }
33          }
34        }
35        s = -1;
36        if(count < n)
37          // Find next vertex to start search from
38          for(int i = 0; i < n && s == -1; ++i)
39            if(!V[i])
40              s = i;
41      }
42      return P;
43    }
44  }
```

6 Strongly Connected Components

```
1   import java.util.*;
2   class SCC {
3     /**Searches a graph for its strongly connected components
4      * Requires DFS
5      * O(m+n)
6      * @param G Graph represented by adjacency list
7      * @param C Filled with each SCC's vertices
8      * @return  An array with each vertex' SCC-number*/
9     static int[] scc(ArrayList<ArrayList<Integer>> G, ArrayList<ArrayList<Integer>> C) {
10      int n = G.size();
11      // Reverse graph
12      ArrayList<ArrayList<Integer>> R = new ArrayList();
13      for(int i = 0; i < n; ++i)
14        R.add(new ArrayList());
15      for(int a = 0; a < n; ++a)
16        for(int b : G.get(a))
17          R.get(b).add(a);
18      // Initialize arrays for DFS
19      int[] pre = new int[n];
20      int[] post = new int[n];
21      int[] scc = new int[n];
22      Arrays.fill(pre, -1);
23      Arrays.fill(post, -1);
24      Arrays.fill(scc, -1);
25      DFS.dfs(R, pre, post, scc, -1);
26      // Sort vertices by post-value
27      int[] order = new int[2*n];
28      Arrays.fill(order, -1);
29      for(int i = 0; i < n; ++i)
30        order[post[i]] = i;
31      // Initialize arrays for DFS
32      Arrays.fill(pre, -1);
33      Arrays.fill(post, -1);
34      Arrays.fill(scc, -1);
```


```
35      // Search for SCC's in the graph
36      for(int i = 2*n-1; i >= 0; --i)
37        if(order[i] != -1) {
38          if(scc[order[i]] == -1) {
39            C.add(new ArrayList());
40            DFS.dfs(G, pre, post, scc, order[i]);
41          }
42          C.get(scc[order[i]]).add(order[i]);
43        }
44      return scc;
45    }
46  }
```

7 2-SAT

```
1   import java.util.ArrayList;
2   class TwoSat {
3     /**Checks if a 2-CNF (AND of ORs) forumula is satisfiable
4      * Requires SCC
5      * O(m+n)
6      * To get a valid assignment:
7      *     1: Pick an unassigned vertex V
8      *     2: Assign TRUE to all vertices reachable from V
9      *     3: Assign FALSE to their negations
10     *     4: Repeat until all vertices are assigned
11     * @param F The 2-CNF forumla
12     * @param n The number of variables
13     * @return  Whether the formula is satisfiable*/
14    static boolean twoSat(ArrayList<Clause> F, int n) {
15      ArrayList<ArrayList<Integer>> G = new ArrayList();
16      for(int i = 0; i < 2*n; ++i)
17        G.add(new ArrayList());
18      // Construct graph from forumla
19      for(Clause c : F) {
20        int a = c.y + (c.yn ? n : 0);
21        int b = c.x + (c.xn ? n : 0);
22        G.get((a+n)%(2*n)).add(b);
23        G.get((b+n)%(2*n)).add(a);
24      }
25      ArrayList<ArrayList<Integer>> C = new ArrayList();
26      // Check that a variable and it's negation are not in the same SCC
27      int[] scc = SCC.scc(G, C);
28      for(int i = 0; i < n; ++i)
29        if(scc[i] == scc[i+n])
30          return false;
31      return true;
32    }
33    static class Clause {
34      int x, y;
35      boolean xn, yn;
36      /**@param x   First variable
37       * @param xn  Whether first variable is negated
38       * @param y   Second variable
39       * @param yn  Whether second variable is negated*/
40      Clause(int x, boolean xn, int y, boolean yn) {
41        this.x = x;
42        this.xn = xn;
43        this.y = y;
44        this.yn = yn;
45      }
46    }
47  }
```

8 APSP - Floyd-Warshall

```java
import java.util.*;
class FloydWarshall {
  /**Finds shortest paths between all pairs of vertices in the graph
   * Does not detect negative cycles
   * O(n^3)
   * Properties:
   * D[u][v] gives the lenght of the shortest path from vertex u to vertex v
   * P[u][v] gives the last vertex on the shortest path from vertex u to vertex v
   * If v is not reachable from u, D[u][v] == Integer.MAX_VALUE/2 - 1 and P[u][v] == -1
   * @param G Graph represented by adjacency list
   * @param P [n][n] matrix filled with the last vertex on the shortest path between every pair
       of vertices
   * @return  [n][n] matrix with the distance between every pair of vertices*/
  static int[][] floydWarshall(ArrayList<ArrayList<Pair>> G, int[][] P) {
    int n = G.size();
    // Initialize distance array and parent array
    int[][] D = new int[n][n];
    for(int i = 0; i < n; ++i) {
      Arrays.fill(D[i], Integer.MAX_VALUE/2 - 1);
      Arrays.fill(P[i], -1);
    }
    for(int i = 0; i < n; ++i)
      for(Pair a : G.get(i)) {
        D[i][a.id] = a.v;
        P[i][a.id] = i;
      }
    for(int k = 1; k < n; ++k)
      for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++j)
          if(D[i][k]+D[k][j] < D[i][j]) {
            D[i][j] = D[i][k]+D[k][j];
            P[i][j] = P[k][j];
          }
    return D;
  }
  /**Returns the shortest path between s and t, as calculated by floydWarshall()
   * O(n)
   *
   * @param P The [n][n] P matrix filled in by floydWarshall()
   * @param s The start vertex
   * @param t The end vertex
   * @return  The shortest path between s and t*/
  static ArrayList<Integer> getPath(int[][] P, int s, int t) {
    ArrayList<Integer> p = new ArrayList();
    while(s != t && t != -1) {
      p.add(t);
      t = P[s][t];
    }
    if(t == -1)
      return new ArrayList();
    p.add(t);
    Collections.reverse(p);
    return p;
  }
}
```

9 MaxFlow - Edmonds-Karp

```java
import java.util.*;
class EdmondsKarp {
  /** Finds the maximum flow through a graph
   * O(n*m^2)
   * @param G Graph represented by adjacency list
   * @param C Capacity matrix (must be n by n)
   * @param s Source vertex
   * @param t Sink vertex
   * @return  The maximum flow from s to t*/
  static int edmondsKarp(ArrayList<ArrayList<Integer>> G, int[][] C, int s, int t) {
    int n = G.size();
    // Residual capacity from u to v is C[u][v] - F[u][v]
    int[][] F = new int[n][n];
    while(true) {
    // Initialize parent table
      int[] P = new int[n];
      Arrays.fill(P, -1);
      P[s] = s;
      // Initialize capacity of path to node table
      int[] M = new int[n];
      M[s] = Integer.MAX_VALUE;
      LinkedList<Integer> Q = new LinkedList();
      Q.add(s);
      LOOP:
      while(!Q.isEmpty()) {
        int u = Q.poll();
        for(int v : G.get(u)) {
          // There is available capacity, and v is not seen before in search
          if(C[u][v] - F[u][v] > 0 && P[v] == -1) {
            P[v] = u;
            M[v] = Math.min(M[u], C[u][v] - F[u][v]);
            if(v != t)
              Q.add(v);
            else {
              // Backtrack search, updating flow
              while(P[v] != v) {
                u = P[v];
                F[u][v] += M[t];
                F[v][u] -= M[t];
                v = u;
              }
              break LOOP;
            }
          }
        }
      }
      if(P[t] == -1) {
      // No path to t found
        int sum = 0;
        for(int x : F[s])
          sum += x;
        return sum;
      }
    }
  }
}
```

10 Permutations

```
1  import java.util.ArrayList;
2  public class Permutations {
3    /**Generates all permutations of numbers 0..b.length-1
4     * @param b [n] array indicating which numbers have been used (initialized to false)
5     * @param l For buildinga single permutation
6     * @param p Is filled with all the permutations*/
7    static void permutations(boolean[] b, ArrayList<Integer> l, ArrayList<ArrayList<Integer>> P)
         {
8      if(l.size() == b.length)
9        P.add(l);
10     else {
11       // Perform pruning if not all permutations should be generated
12       for(int i = 0; i < b.length; ++i)
13         if(!b[i]) {
14           b[i] = true;
15           ArrayList<Integer> l2 = new ArrayList(l);
16           l2.add(i);
17           permutations(b, l2, P);
18           b[i] = false;
19         }
20     }
21   }
22 }
```

11 Combinations

```
1  import java.util.ArrayList;
2  public class Combinations {
3    /**Generates all combinations of k numbers from 0..n-1
4     * @param n Number of elements to choose from
5     * @param k Number of elements in each combination*/
6    static ArrayList<ArrayList<Integer>> combinations(int n, int k) {
7      ArrayList<ArrayList<Integer>> C = new ArrayList();
8      ArrayList<Integer> c = new ArrayList();
9      for(int i = 0; i < k; ++i)
10       c.add(i);
11     while(true) {
12       C.add(new ArrayList(c));
13       int i = k-1;
14       while(i >= 0 && c.get(i) == n-k+i)
15         i--;
16       if(i < 0)
17         break;
18       c.set(i, c.get(i)+1);
19       for(int j = i+1; j < k; j++)
20         c.set(j, c.get(i)+j-i);
21     }
22     return C;
23   }
24 }
```

12 Fast Input

```java
import java.io.*;
import java.util.*;
/**For performing fast input operations*/
public class Input {
  static BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
  static StringTokenizer st = new StringTokenizer("");
  public static void main(String[] args) throws Exception {}
  static String readString() throws Exception {
    while(!st.hasMoreTokens())
      st = new StringTokenizer(stdin.readLine());
    return st.nextToken();
  }
  static int readInt() throws Exception {
    return Integer.parseInt(readString());
  }
  static double readDouble() throws Exception {
    return Double.parseDouble(readString());
  }
}
```

13 Pair

```java
/**Couples together an id and a weight, for use in priority queues*/
class Pair implements Comparable<Pair> {
  int id, v;
  /**@param id  Identifier
   * @param v   Value*/
  Pair(int id, int v) {
    this.id = id;
    this.v = v;
  }
  @Override
  public int compareTo(Pair o) {
    return Integer.compare(v, o.v);
  }
}
```

14 KMP - StringMatching

```java
/**KMP: Knuth Morris Pratt algorithm.
 * Finds the occurences of a pattern string in a longer text string.
 * O(n + m). n = length of pattern, m = length of text.*/
public class KMP {
  public int lps[];
  /**Fills table lps.
   * lps[i] is the length of the longest border (border = both prefix and suffix) of
   * prefix of ptrn of length i (substring of ptrn starting at ptrn[0] of length i).
   * NB: lps[0] = -1.
   * O(n). n = length of ptrn.*/
  public int[] preProcessPattern(char[] ptrn) {
    int i = 0, j = -1;
    int ptrnLen = ptrn.length;
    int[] b = new int[ptrnLen + 1];
    b[i] = j;
    while (i < ptrnLen) {
      // if there is mismatch consider next widest border
      while (j >= 0 && ptrn[i] != ptrn[j])
        j = b[j];
      i++;
```


```
21        j++;
22        b[i] = j;
23      }
24      return b;
25    }
26    /**Based on the pre processed array, search for the pattern in the text
27     * returns: ArrayList of indexes in textS where ptrnS starts.
28     * O(m). m = length of textS.*/
29    public ArrayList<Integer> searchSubString(String textS, String ptrnS) {
30      char[] text = textS.toCharArray();
31      char[] ptrn = ptrnS.toCharArray();
32      ArrayList<Integer> positions = new ArrayList<Integer>();
33      int i = 0, j = 0;
34      int ptrnLen = ptrn.length;
35      int txtLen = text.length;
36      // initialize new array and preprocess the pattern
37      int[] b = preProcessPattern(ptrn);
38      while (i < txtLen) {
39        while (j >= 0 && text[i] != ptrn[j])
40          j = b[j];
41        i++;
42        j++;
43        // a match is found
44        if (j == ptrnLen) {
45          positions.add((i - ptrnLen));
46          j = b[j];
47        }
48      }
49      return positions;
50    }
51  }
```

15 BinarySearch

```
1   public class BinarySearch {
2     public static double BinarySearch_cont(){
3       double lo = 0;
4       double hi = 1000000; // something large enough NB: can cause problems if too large
5       double m = (lo+hi)/2.0;
6       while(hi-lo > 0.0000001){ // some epsilon
7         m = (lo+hi)/2.0;
8         boolean r = hit(m); // some function used to test if m is high enough
9         if(!r) lo = m;
10        if(r) hi = m;
11      }
12      return m;
13    }
14    /**Searches for the integer key in the sorted array a[].
15     * @param key the search key
16     * @param a   the array of integers, must be sorted in ascending order
17     * @return    index of key in array a[] if present; -1 if not present*/
18    public static int BinarySearch_discrete(int key, int[] a) {
19      int lo = 0;
20      int hi = a.length - 1;
21      while (lo <= hi) {
22        // Key is in a[lo..hi] or not present.
23        int mid = (hi+lo)/2;
24        if      (key < a[mid]) hi = mid - 1;
25        else if (key > a[mid]) lo = mid + 1;
26        else return mid;
27      }
28      return -1;
29    }
30  }
```

16 Ternary Search

```java
public class TernarySearch {
  // f(n) is the function we want to find min or max of on [a,b]
  // Must be strictly decreasing and then strictly increasing (or reverse).
  public static double f(double n){}
  // Returns the value x in [a,b] where f(x) is minimum or maximum (> or <)
  // O(log(b - a))
  public static double ternary_search(double a, double b) {
    double eps = 0.00001; // some small value
    while (Math.abs(b - a) > eps) {
      double l = a + (b - a)/3;
      double r = b - (b - a)/3;
      if (f(l) > f(r)) // < for finding max in interval
        a = l;
      else
        b = r;
    }
    return a;
  }
  // Recursive version
  public static double ternarySearch(double l, double r){
    double absolutePrecision = 0.00000001;
    //left and right are the current bounds; the maximum is between them
    if ((r - l) < absolutePrecision){
      return (l + r)/2;
    }
    double leftThird = (2*l + r)/3;
    double rightThird = (l + 2*r)/3;
    if (f(leftThird) > f(rightThird))
      return ternarySearch(leftThird, r);
    else
      return ternarySearch(l, rightThird);
  }
}
```

17 UnionFind

```java
/**Union & find, O(log*n) (fast!)
 * If Fu[v] >= 0, Fu[v] is the element v is pointing at. Otherwise Fu[v] is -(the number of
    elements in that component)*/
public class UnionFind {
  public int Fu[];
  private int n;
  public int FuFind(int v){
    if(Fu[v] < 0) return v;
    else return (Fu[v] = FuFind(Fu[v]));
  }
  public void FuJoin(int v, int w){
    v = FuFind(v);
    w = FuFind(w);
    if(v == w) return;
    if(Fu[v] > Fu[w]) {
      //swaps elements
      int temp = v;
      v = w;
      w = temp;
    }
    Fu[v] += Fu[w];
    Fu[w] = v;
  }
  private void FuInit(){
    for(int i = 0; i < n; i++)
```


```
25        Fu[i] = -1;
26    }
27    public int numOfElementsInComp(int u){
28      return - Fu[FuFind(u)];
29    }
30    public UnionFind(int numOfElements){
31      n = numOfElements;
32      Fu = new int[n];
33      FuInit();
34    }
35  }
```

18 Counting Tree - AddToCell - SumFromInterval

```
1   /**Counting tree implementation. Add to cell, get sum in interval.*/
2   public class CountingTree_AddToCell_GetSumFromInterval {
3     //Space used is offset,offset + size of table
4     private int OFFSET = 135000;
5     private int tree[];
6     // Add v to cell i. Max, min and gcd in interval in comments
7     public void update(int i, int v) {
8       i += OFFSET;
9       tree[i] += v; // max/min/gcd: tree[i] = v;
10      i /= 2;
11      while(i > 0) {
12        tree[i] = tree[2*i]+tree[2*i+1];  //max: tree[i] = Math.max(tree[2*i], tree[2*i+1]);
13                                          //min: tree[i] = Math.min(tree[2*i], tree[2*i+1]);
14                                          //gcd: tree[i] = gcd(tree[2*i], tree[2*i+1]);
15        i /= 2;
16      }
17    }
18    // Get sum in interval l (inclusive) to r (inclusive)
19    public int query(int l, int r) {
20      int result = 0; // min: result = INF, max/gcd: result = 0
21      l += OFFSET - 1; r+= OFFSET+1;
22      while(l/2 != r/2) {
23        if(l%2 == 0) result += tree[l^1]; //max: result = Math.max(result, tree[l^1]);
24                                          //min: result = Math.min(result, tree[l^1]);
25                                          //gcd: result = gcd(result, tree[l^1]);
26        if(r%2 == 1) result += tree[r^1]; //max: result = Math.max(result, tree[r^1]);
27                                          //min: result = Math.min(result, tree[r^1]);
28                                          //gcd: result = gcd(result, tree[r^1]);
29        l /= 2; r /= 2;
30      }
31      return result;
32    }
33    public CountingTree_AddToCell_GetSumFromInterval(){
34      tree = new int[2*OFFSET];
35    }
36  }
```

19 Counting Tree - Michal - AddToInterval - SumFromInterval

```
1  public class CountingTree_AddToInterval_GetSumFromInterval_Michal {
2    private int OFFSET = 135000; // This may have to be increased for big input
3    private int bonus[];
4    private int tree[];
5    // For setting max value in an interval and retrieving max in an interval, see comments
6    // NB: += becomes =
7    // Returns: what is in interval l to r (inclusive)
8    public int query(int l, int r){
9      l+=OFFSET-1; r+=OFFSET+1;
10     ArrayList<Integer> left = new ArrayList<Integer>();
11     ArrayList<Integer> right = new ArrayList<Integer>();
12     while((l>>1) != (r>>1)) {
13       left.add(l); right.add(r);
14       l>>=1; r>>=1;
15     }
16     int root=(l>>1), bs=0;
17     while(l>>1 != 0) { // while (l>0)
18       bs+=bonus[l>>1]; // bs= Math.max(bs,bonus[l])
19       l>>=1;
20     }
21     int lbonus=bs+bonus[root<<1]; // = Math.max(bs,bonus[root<<1]);
22     int rbonus=bs+bonus[(root<<1)+1]; // = Math.max(bs,bonus[(root<<1)+1]);
23     int result=0;
24     for(int i = left.size()-1; i >= 0; i--){
25       if(left.get(i)%2 != 1)
26         result+=((lbonus+bonus[left.get(i)^1])<<i)+tree[left.get(i)^1];
27         // = Math.max(result,Math.max(tree[left.get(i)^1],lbonus));
28       if(right.get(i)%2 == 1)
29         result+=((rbonus+bonus[right.get(i)^1])<<i)+tree[right.get(i)^1];
30         // = Math.max(result,Math.max(tree[right.get(i)^1],rbonus));
31       lbonus+=bonus[left.get(i)]; // = Math.max(lbonus,bonus[left.get(i)]);
32       rbonus+=bonus[right.get(i)]; // = Math.max(rbonus,bonus[right.get(i)]);
33     }
34     return result;
35   }
36   //Adds v to all cells in interval [l,r]
37   public void add(int l, int r, int v){
38     l+=OFFSET-1; r+=OFFSET+1;
39     int chunk=0,lcount=0,rcount=0;
40     while((l>>1) != (r>>1)){
41       if(l%2 != 1) {  // Substitute whole block with: {tree[l^1]=v; bonus[l^1]=v;}
42         bonus[l^1]+=v;
43         lcount+=v<<chunk;
44       }
45       if(r%2 == 1) { // {tree[r^1]=v; bonus[r^1]=v;}
46         bonus[r^1]+=v;
47         rcount+=v<<chunk;
48       }
49       tree[l>>1]+=lcount; // = Math.max(bonus[l>>1], Math.max(tree[l],tree[l^1]));
50       tree[r>>1]+=rcount; // = Math.max(bonus[r>>1], Math.max(tree[r],tree[r^1]));
51       l>>=1; r>>=1; chunk++;
52     }
53     lcount+=rcount;
54     while(l>>1 != 0) {
55       tree[l>>1]+=lcount; // = Math.max(bonus[l>>1], Math.max(tree[l],tree[l^1]));
56       l>>=1;
57     }
58   }
59   public CountingTree_AddToInterval_GetSumFromInterval_Michal(){
60     bonus = new int[OFFSET*2];
61     tree = new int[OFFSET*2];
62   }
63 }
```

20 Counting Tree - AddToInterval - SumFromCell - NOT TESTED

```java
/** Counting tree implementation. Add to interval, get value in cell.*/
public class CountingTree_AddToInterval_GetValueInCell {
  private int OFFSET = 135000;
  private int bonus[];
  //Returns: what is in cell i
  public int query(int p){
    p+=OFFSET;
    int result=0;
    while (p>0){
      result+=bonus[p];
      p>>=1;
    }
    return result;
  }
  //Adds v to all cells in interval [l,r]
  public void add(int l, int r, int v){
    l+=OFFSET-1; r+=OFFSET+1;
    while ((l>>1)!=(r>>1)){
      if (l%2 != 1)
        bonus[l^1]+=v;
      if (r%2 == 1)
        bonus[r^1]+=v;
      l>>=1; r>>=1;
    }
  }
  public CountingTree_AddToInterval_GetValueInCell(){
    bonus = new int[OFFSET*2];
  }
}
```

21 Geometry 2D

```python
## NOTE: ALL CODE FOR GEOMETRY 2D IS PYTHON 3!
## In particular, note that 5/2 == 2.5  and  5//2 == 2
## / is floating point division, and rationals behave as rationals, not doubles,
## with respect to precicion errors (e.g. precision error only occurs with
## irrational numbers).
import sys
import math
EPS = 0.000000001

def eq(a, b): return abs(a-b) < EPS ## For avoiding floating point errors
def distance_sq(a, b): return (a[0]-b[0])**2 + (a[1]-b[1])**2 ## Note: squared!
def cross_product(a, b): return a[0] * b[1] - b[0] * a[1] ## 2d version
def dot_product(a, b): return sum([p[0]*p[1] for p in zip(a, b)]) ## axbx + ayby
def vsum(a, b): return tuple(map(sum, zip(a,b))) ## Sum of two vectors
def vminus(a): return tuple([-x for x in a]) ## Negation of a vector
def vdiff(a, b): return vsum(a, vminus(b)) ## The vector p such that b+p=a
def ccw(a,b,c): return (c[1]-a[1]) * (b[0]-a[0]) > (b[1]-a[1]) * (c[0]-a[0])
def angle_line(l): return math.atan2(l[1][1]-l[0][1], l[1][0]-l[0][0])

def triangle_area_db(a, b, c):
    ## Returns twice the area of the triangle. Points a,b,c are on form (x, y)
    ## Caller responsibility to divide by two if required.
    ## Negative area if points are in clockwise order.
    return a[0]*(b[1]-c[1]) + b[0]*(c[1]-a[1]) + c[0]*(a[1]-b[1])

def lines_are_parallel(la, lb):
    ## Input is two line segments, la and lb on the form ((x1, y1), (x2, y2))
    adx, ady = la[1][0] - la[0][0], la[1][1] - la[0][1]
    bdx, bdy = lb[1][0] - lb[0][0], lb[1][1] - lb[0][1]
```

```
30      return eq(adx*bdy, ady*bdx)
31
32  def pts_are_collinear(a, b, c):
33      ## Returns true if the three points are on the same line
34      return eq(0, triangle_area_db(a, b, c))
35
36  def pt_on_line(point, line):
37      ## Returns true if the point is on the (infinitely extending) line
38      return pts_are_collinear(point, line[0], line[1])
39
40  def pt_rightof_line(point, line):  ## NOT TESTED
41      ## Returns true if the point is to the rigth of the (infinitely extending)
42      ## line. Here, "to the right" is with respect to the orientation of line
43      ## Step 0: Move origo to starting point of line
44      p = px, py = point[0] - line[0][0], point[1] - line[0][1]
45      a = ax, ay = 0, 0
46      b = bx, by = line[1][0] - line[0][0], line[1][1] - line[0][1]
47      ## Step 1: If cross product is negative, point is to the right of the line
48      return cross_product(b, p) < 0
49
50  def lines_are_collinear(la, lb):
51      ## Input is two line segments, la and lb on the form ((x1, y1), (x2, y2))
52      tarea1 = triangle_area_db(la[0], la[1], lb[0])
53      tarea2 = triangle_area_db(la[0], la[1], lb[1])
54      return eq(0, tarea1) and eq(0, tarea2)
55
56  def line_intersect(la, lb):
57      ## Returns intersection point between the infinitely extending lines
58      ## passing through the line segments la and lb.
59      ## Caller must ensure the following:
60      ##  *  The line segments are not collinear (infinitely many solutions)
61      ##  *  The line segments are not otherwise parallel (no solution)
62      ##  *  Each line segment has strictly positive length (not length 0)
63      xdiff = (la[0][0] - la[1][0], lb[0][0] - lb[1][0])
64      ydiff = (la[0][1] - la[1][1], lb[0][1] - lb[1][1])
65
66      def det(a, b): return a[0] * b[1] - a[1] * b[0] # Determinant of 2x2 matrix
67
68      div = det(xdiff, ydiff)
69      if div == 0: raise Exception('no intersection: '+str(la)+" "+str(lb))
70
71      d = (det(*la), det(*lb))
72      x = det(d, xdiff) / div
73      y = det(d, ydiff) / div
74      return x, y
75
76  def pt_onsegment(point, seg):
77      if not pts_are_collinear(point, seg[0], seg[1]): return False
78      return point == sorted([point, seg[0], seg[1]])[1] ## Is in the middle
79
80  def pt_segdistance_sq(point, seg):
81      ## Return the square of the distance between a point and a line segment.
82      ## Caller responsibility to take square root if required.
83      px, py = point
84      ax, ay = seg[0]
85      bx, by = seg[1]
86      if ax == bx and ay == by: return distance_sq((px, py), (ax, ay))
87      # Step 0: Move everything such that a is at the origin. Now we only care
88      # about the two vectors p and b.
89      p = px, py = px - ax, py - ay
90      b = bx, by = bx - ax, by - ay
91      a = 0, 0
92      # Step 1: Determine the projection factor of p onto q. If f in [0..1],
93      # then shortest distance is between p and f*b. Otherwise, shortest distance
94      # is either between p and a or p and b.
95      f = (px*bx + py*by)/distance_sq(a, b)
```


```
96      q = qx, qy = f*bx, f*by
97      if 0 < f < 1: return distance_sq(p, q)
98      return min(distance_sq(p, a), distance_sq(p, b))
99
100 def pt_closestpt_line(point, line): ## NOT TESTED
101     ## Return the point on the line which is closest to the given pointself.
102     if line[0] == line[1]: return distance_sq((px, py), (ax, ay))
103     # Step 0: Move everything such that a is at the origin. Now we only care
104     # about the two vectors p and b.
105     p = px, py = vdiff(point, line[0])
106     b = bx, by = vdiff(line[1], line[0])
107     a = 0, 0
108     # Step 1: Determine the projection factor of p onto q. If f in [0..1],
109     # then shortest distance is between p and f*b. Otherwise, shortest distance
110     # is either between p and a or p and b.
111     f = (px*bx + py*by)/distance_sq(a, b)
112     q = (f*bx, f*by)
113     return vsum(line[0], q)
114
115 def pt_linedistance_sq(p, ln):  ## NOT TESTED
116     return triangle_area_db(p, ln[0], ln[1])**2 / distance_sq(ln[0], ln[1])
117
118 def perpendicular(p, ln): ## NOT TESTED
119     ## Returns a line perpendicular to ln, passing through the point p.
120     dx, dy = ln[0][0]-ln[1][0], ln[0][1]-ln[1][1]
121     return (p, vsum(p, (-dy, dx))) ## Swappind dx, dy and sign
122
123 def seg_intersect(la, lb):
124     ## Returns true if line segments la and lb intersect, false otherwise
125     ## Step 0: Eliminate cases when line segments have length 0, and
126     ## check cases when the endpoints are touching. By doing this now,
127     ## we don't need to worry about float precision later.
128     if eq(0, pt_segdistance_sq(la[0], lb)): return True
129     if eq(0, pt_segdistance_sq(la[1], lb)): return True
130     if eq(0, pt_segdistance_sq(lb[0], la)): return True
131     if eq(0, pt_segdistance_sq(lb[1], la)): return True
132     if eq(0, distance_sq(la[0], la[1])): return False
133     if eq(0, distance_sq(lb[0], lb[1])): return False
134
135     ## Consider line segment la as (p, p+r) and line segment lb as (q, q+s)
136     p, r = la[0], vsum(la[1], vminus(la[0]))
137     q, s = lb[0], vsum(lb[1], vminus(lb[0]))
138
139     rxs = cross_product(r, s) ## 0 if vectors are parallel (s, r same angle)
140     q_p = vsum(q, vminus(p)) ## vector q-p
141     q_pxr = cross_product(q_p, r) ## 0 if q-p parallel to r (p+t0r hits q(?))
142     q_pxs = cross_product(q_p, s) ## 0 if q-p parallel to s (p+t1r hits q+s(?))
143
144     ## Case 1: Lines are collinear
145     if eq(0, rxs) and eq(0, q_pxr):
146         t0 = dot_product(q_p, r) / dot_product(r, r)
147         t1 = dot_product(vsum(q_p, s), r) / dot_product(r, r)
148
149         ## If t0 and t1 overlap with interval [0, 1], then they intersect
150         t0, t1 = sorted([t0, t1])
151         if t1 < 0 or 1 < t0: return False
152         return True
153
154     ## Case 2: Parallel, but non-collinear
155     if eq(0, rxs): return False
156
157     ## Case 3: Intersecting in single point
158     t = q_pxs / rxs
159     u = q_pxr / rxs
160     if 0 <= t <= 1 and 0 <= u <= 1: return True
161
```

```
162     ## Case 4: Not parallel, but not intersecting
163     return False
164 
165 def seg_intersect2(la, lb): ## Poor man's intersection, NOT TESTED thoroughly
166     if ccw(la[0],lb[0],lb[1]) == ccw(la[1], lb[0], lb[1]): return False
167     return not ccw(la[0],la[1],lb[0]) == ccw(la[0], la[1], lb[1])
168 
169 def seg_overlap(la, lb):
170     ## Returns the intersection subsegment of la and lb. Could be of length 0.
171     if not seg_intersect(la, lb): return None
172     elif lines_are_collinear(la, lb):
173         ## Sort the four points, pick the middle guys
174         pts = sorted([la[0], la[1], lb[0], lb[1]])
175         return (pts[1][0], pts[1][1]), (pts[2][0], pts[2][1])
176     pt = line_intersect(la, lb)
177     return pt, pt
178 
179 def seg_segdistance_sq(la, lb):
180     ## Return the square of the distance between a two line segments.
181     ## Caller responsibility to take square root if required.
182     if seg_intersect(la, lb): return 0
183     adist = min(pt_segdistance_sq(la[0], lb), pt_segdistance_sq(la[1], lb))
184     bdist = min(pt_segdistance_sq(lb[0], la), pt_segdistance_sq(lb[1], la))
185     return min(adist, bdist)
186 
187 
188 def pt_in_poly(pt, polygon):
189     ## Returns tuple (is_in, is_on) indicating whether the point is in and on
190     ## the polygon, repectively. If the point is on, it is also in.
191     isin = False
192 
193     x, y = pt
194     a = polygon[-1]
195     for b in polygon:
196         if pt_onsegment(pt, (a, b)): return True, True
197         ax, ay, bx, by = a[0], a[1], b[0], b[1]
198         if min(ay,by) < y <= max(ay,by) and x < (bx-ax)*(y-ay)/(by-ay) + ax:
199             isin = not isin
200         a = b
201 
202     return isin, False
203 
204 def closestpair(pts): ## NOT TESTED THOROUGHLY
205     ## Given many points, returns two which has a minimum distance
206     ## Caller responsible for length of points being at least two, and that
207     ## the list of points is sorted by x-value.
208     ## Returns a triplet: the distance (squared), point a, and point b
209     ## Base cases:
210     if (len(pts)) == 2: return distance_sq(pts[0], pts[1]), pts[0], pts[1]
211     if (len(pts)) == 3:
212         alt_a = distance_sq(pts[0], pts[1]), pts[0], pts[1]
213         alt_b = distance_sq(pts[0], pts[2]), pts[0], pts[2]
214         alt_c = distance_sq(pts[2], pts[1]), pts[2], pts[1]
215         return min(alt_a, alt_b, alt_c)
216 
217     ## Recursive case:
218     left, right = pts[:len(pts)//2], pts[len(pts)//2:]
219     best, win_a, win_b = min(closestpair(left), closestpair(right))
220     xcut = right[0][0] ## x-value of cut
221 
222     ## The case when the shortest distance is on the border between left and r
223     border = []
224     for i in range(len(left)-1, -1, -1):
225         if (xcut - left[i][0])**2 >= best: break
226         border.append(left[i])
227 
```


```
    for i in range(0, len(right)):
        if (xcut - right[i][0])**2 >= best: break
        border.append(right[i])

    if len(border) <= 1: return best, win_a, win_b
    border.sort(key= lambda tup: (tup[1], tup[0]))

    for i in range(len(border)):
        p1 = border[i]
        for j in range(i+1, len(border)):
            p2 = border[j]
            if (p2[1]-p1[1])**2 >= best: break
            d = distance_sq(p1, p2)
            if d < best:
                best = d
                win_a, win_b = p1, p2

    return best, win_a, win_b

def convexhull(points):
    # Task: Find a convex hull around input points
    # Step 1:  Pick a point with minimum y-value. Ensure only one of each point
    points = set(points)
    inity, initx = INF, INF
    for x, y in points:
        if y < inity or (y == inity and x < initx):
            inity, initx = y, x
    points = list(points)
    if len(points) == 1: return points
    ipt = (initx, inity)

    # Step 2a:  Sort all points by the angle they form with initial point
    #           First priority: angle. Second priority: distance
    def specialgetangle(a, b): return -EPS/100 if a==b else angle_line((a, b))
    points.sort(key=lambda pt:(specialgetangle(ipt,pt), -distance_sq(ipt,pt)))

    # Step 2b:  Ensure there is at most one point for each angle, namely the
    #           point furthest away from the initial point.
    points2 = points[:1]
    for p in points:
        if specialgetangle(ipt, p) != specialgetangle(ipt,points2[-1]):
            points2.append(p)
    points = points2

    # Step 3:   Sweep through points; regret past choices if angle is clockwise
    hull = points[:2]
    for pt in points[2:]:
        while len(hull)>= 2 and not ccw(hull[-2], hull[-1], pt):
            hull.pop()
        hull.append(pt)

    return hull
```

22 Geometry 3D

```
public class Geometry3D {
  /* squared distance */
  public static double distsq(double x1,double y1,double z1,double x2,double y2,double z2) {
    double dx=x1-x2,dy=y1-y2,dz=z1-z2;
    return dx*dx+dy*dy+dz*dz;
  }
  /* distance from point (x,y,z) to line segment (x1,y1,z1)-(x2,y2,z2) */
  public static double pointtolineseg(double x,double y,double z,double x1,double y1,double z1,
      double x2,double y2,double z2) {
```


```
    double vx=x2-x1, vy=y2-y1, vz=z2-z1;
    double wx=x-x1, wy=y-y1, wz=z-z1;
    double c1=vx*wx+vy*wy+vz*wz;
    /* left endpoint closest? */
    if(c1<0) return Math.sqrt(distsq(x,y,z,x1,y1,z1));
    double c2=vx*vx+vy*vy+vz*vz;
    /* right endpoint closest? */
    if(c2<=c1) return Math.sqrt(distsq(x,y,z,x2,y2,z2));
    double b=c1/c2;
    /* find the actual point in the segment */
    double bx=x1+vx*b, by=y1+vy*b, bz=z1+vz*b;
    return Math.sqrt(distsq(bx,by,bz,x,y,z));
  }
  // Finds distance between two linesegments using ternary search
  public static double ternarydist(double x1_start, double y1_start, double z1_start,
      double x1_end, double y1_end, double z1_end, double x2_start, double y2_start,
      double z2_start, double x2_end, double y2_end, double z2_end) {
    double lo=0,hi=1;
    double dx=x1_end-x1_start,dy=y1_end-y1_start,dz=z1_end-z1_start;
    for(int it=0;it<400;it++) {
      double m1=lo+(hi-lo)/3;
      double m2=lo+(hi-lo)/1.5;
      double r1=pointtolineseg(x1_start+dx*m1,y1_start+dy*m1,z1_start+dz*m1,x2_start,y2_start,
          z2_start,x2_end,y2_end,z2_end);
      double r2=pointtolineseg(x1_start+dx*m2,y1_start+dy*m2,z1_start+dz*m2,x2_start,y2_start,
          z2_start,x2_end,y2_end,z2_end);
      if(r1<r2) hi=m2;
      else lo=m1;
    }
    return pointtolineseg(x1_start+dx*lo,y1_start+dy*lo,z1_start+dz*lo,x2_start,y2_start,
        z2_start,x2_end,y2_end,z2_end);
  }
  /* distance from point (x,y,z) to line (extended infinitely)
   * passing through (x1,y1,z1) and (x2,y2,z2) */
  public static double pointtoline(double x,double y,double z,double x1,double y1,double z1,
      double x2,double y2,double z2) {
    double vx=x2-x1, vy=y2-y1, vz=z2-z1;
    double wx=x-x1, wy=y-y1, wz=z-z1;
    double c1=vx*wx+vy*wy+vz*wz;
    double c2=vx*vx+vy*vy+vz*vz;
    double b=c1/c2;
    double bx=x1+vx*b, by=y1+vy*b, bz=z1+vz*b;
    return Math.sqrt(distsq(bx,by,bz,x,y,z));
  }

}
```

23 Euclids algorithm, GCD, Inverse

```
/**Euclidian algorithm. Used for finding gcd, x and y in a*x+b*y=1 and the inverse of a mod b*/
public class Euclid {
  //Euklids algoritme, finner gcd av a og b
  public static long gcd(long a, long b){
    if(a < b){
      long temp = a;
      a = b;
      b = temp;
    }
    if(b == 0) return a;
    return gcd(b, a % b);
  }
  //Extended Euklid, finner x og y, slik at a*x + b*y = 1. NB: gcd(a,b) = 1
  //If found solution (x,y), then all other solutions are of form (x+lb, y-la), l integer
  // O(log(a or b)^2)
```


```
16    public Pair ext_euclid(long a, long b){
17      if(a < b){
18        Pair r = ext_euclid(b,a);
19        return new Pair(r.second, r.first);
20      }
21      if(b == 0) return new Pair(1, 0);
22      long q = a / b;
23      long rem = a - b * q;
24      Pair r = ext_euclid(b, rem);
25      Pair ret = new Pair(r.second, r.first - q * r.second);
26      return ret;
27    }
28    public long inverse(long num, long modulo){
29      num = num%modulo;
30      Pair p = ext_euclid(num, modulo);
31      long ret = p.first;
32      if(ret < 0) return (modulo + ret) % modulo;
33      return ret % modulo;
34    }
35    class Pair{
36      public long first;
37      public long second;
38      public Pair(long frst, long scnd){
39        first = frst;
40        second = scnd;
41      }
42    }
43  }
```

24 Prime Numbers

```
1   import java.util.ArrayList;
2   public class Primes {
3     /**Generate primes. O(n*logn)*/
4     public ArrayList<Integer> primes;
5     public void sieve(int n){
6       primes = new ArrayList<Integer>();
7       boolean vis[] = new boolean[n+1];
8       for(int p = 2; p <= n; p++){
9         if(!vis[p]){
10          primes.add(p);
11          for(int j = 2*p; j <= n; j += p)
12            vis[j] = true;
13        }
14      }
15    }
16    /**Factorize integer. O(sqrt(n))*/
17    public ArrayList<Pair> factorize(int m){
18      ArrayList<Pair> result = new ArrayList<Pair>();
19      for(int i = 0; i < primes.size(); i++){
20        int p = primes.get(i);
21        if(p*p > m) break;
22        int alpha = 0;
23        while(m % p == 0){alpha++; m /= p;}
24        if(alpha > 0) result.add(new Pair(p, alpha));
25      }
26      if(m > 1) result.add(new Pair(m, 1));
27      return result;
28    }
29    /**Finds the number of times n! can be divided by "factor"
30     * <=> number of times "factor" appears in the factorization of n!.
31     * O(logn)*/
32    public int numOf(int factor, int n){
33      //every "factor" number will have a factor div by "factor"
```

```
    //every "factor*factor" number will have a factor div by "factor*factor"
    int p = factor;
    int sum = 0;
    while(true){
      int delsum = n / p;
      if(delsum <= 0) break;
      sum += delsum;
      p *= factor;
    }
    return sum;
  }
  class Pair{
    public int first;
    public int second;
    public Pair(int frst, int scnd){
      first = frst;
      second = scnd;
    }
  }
}
```

25 Catalan Numbers

```
/** Finds the nth cataln number.
* The first catalan numbers including for n=0:
* 1 1 2 5 14 42 132 429 1430 4862 16796
* O(n^2)-runtime and O(n)-space*/
import java.util.ArrayList;
public class Catalan {
  public static long cat(int n){
    ArrayList<Long> catalan = new ArrayList<Long>();
    catalan.add((long) 1);
    for(int i = 1; i <= n; i++){
      long temp = 0;
      for(int k = 0; k < i; k++){
        temp += catalan.get(k) * catalan.get(i - 1 - k);
      }
      catalan.add(temp);
    }
    return catalan.get(n);
  }
}
```

26 Binomial

```
/**Finds binomial factors in O(k)-time and O(1) space*/
import java.util.ArrayList;
public class Binomial {
  public static long bin1(int n, int k){
    long ans = 1;
    if(k > n/2){
      k = n-k;
    }
    for(int i = 1; i <= k; i++){
      ans*=(n-i+1);
      ans/=(i);
    }
    return ans;
  }
}
```

27 Sparse Table

```
public class SparseTable {
  int[] logTable;
  int[][] tab;
  int[] a;
  // Initializes the SparseTable (RMQ). GCD in interval in comments. O(N*logN).
  public SparseTable(int[] a) {
    this.a = a;
    int n = a.length;
    logTable = new int[n + 1];
    for (int i = 2; i <= n; i++)
      logTable[i] = logTable[i >> 1] + 1;

    int k = logTable[n] + 1;
    tab = new int[n][k];

    for (int i = 0; i < n; ++i){
      tab[i][0] = i;
    }
    for(int j = 1; j < k; j++) {
      for (int i = 0; i + (1 << j) <= n; i++) {
        int tv = i + (1 << (j - 1));
        int x = tab[i][j - 1];
        int y = tab[tv][j - 1];
        tab[i][j] = a[x] <= a[y] ? x : y; // gcd: tab[i][j] = gcd(x, y);
      }
    }
  }
  // Position of minimum element in interval [l,r]. O(1) // GCD in interval [l, r]
  public int minPos(int l, int r) {
    int k = logTable[r - l];
    int x = tab[l][k];
    int y = tab[r - (1 << k) + 1][k];
    return a[x] <= a[y] ? x : y;  // gcd: return gcd(y, x);
  }
}
```

28 Gaussian Elimination mod P

```
public class Gaussian_Elimination_mod_p {
  // Transforms A so that the leftmost square matrix has at most one 1 per row,
  // and no other nonzero elements.
  // O(n^3)
  static int prime;
  public static void gauss(int[][] A, int num_columns) {
    int n = A.length;
    int m = A[0].length;
    for (int i = 0; i < num_columns; i++) {
      // Finding row with nonzero element at column i, swap this to row i
      for(int k = i; k < num_columns; k++){
        if(A[k][i] != 0){
          int t[] = A[i];
          A[i] = A[k];
          A[k] = t;
          break;
        }
      }
      // Normalize the i-th row.
      int inverse = (int)inverse((long)A[i][i], prime);
      for (int k = i ; k < m; k++) A[i][k] = (A[i][k]*inverse) % prime;
      // Combine the i-th row with the following rows.
      for (int j = 0; j < n; j++) {
```


```
24          if(j == i) continue;
25          int c = A[j][i];
26          A[j][i] = 0;
27          for (int k = i + 1; k < m; k++){
28            A[j][k] = (A[j][k] - c * A[i][k] + c * prime) % prime;
29          }
30        }
31      }
32    }
33    public static void gauss(int[][] A) {
34      gauss(A, Math.min(A.length, A[0].length));
35    }
36  }
```

29 Gaussian Elimination

```
1   public class Gaussian_Elimination {
2     // Transforms A so that the leftmost square matrix has at most one 1 per row,
3     // and no other nonzero elements.
4     // O(n^3)
5     public static void gauss(double[][] A, int num_pivoting_columns) {
6       double eps = 0.001; //Some epsilon
7       int n = A.length;
8       int m = A[0].length;
9       // Process column i.
10      for (int i = 0; i < num_pivoting_columns; i++) {
11        // Consider swapping the i-th row with another one.
12        int best = i;
13        for (int j = i + 1; j < n; j++)
14          if (Math.abs(A[j][i]) > Math.abs(A[best][i]))
15            best = j;
16        if (Math.abs(A[best][i]) < eps)
17          continue;
18        if (best != i){
19          //swaps
20          double[] temp = A[best];
21          A[best] = A[i];
22          A[i] = temp;
23        }
24        // Normalize the i-th row.
25        for (int k = i + 1; k < m; k++)
26          A[i][k] /= A[i][i];
27        A[i][i] = 1;
28        // Combine the i-th row with the following rows.
29        for (int j = i + 1; j < n; j++) {
30          double c = A[j][i];
31          if (Math.abs(c) < eps)
32            continue;
33          A[j][i] = 0;
34          for (int k = i + 1; k < m; k++)
35            A[j][k] -= c * A[i][k];
36        }
37      }
38      for (int i = num_pivoting_columns; i-- > 0; ) {
39        // Find the pivot, if any.
40        int pivot = 0;
41        while (pivot < num_pivoting_columns && Math.abs(A[i][pivot]) < eps)
42          ++pivot;
43        if (pivot == num_pivoting_columns)
44          continue;
45        if (Math.abs(A[i][pivot] - 1) > eps)
46          break;
47        // Combine A[i] with the previous rows.
48        for (int j = 0; j < i; ++j) {
```

```
        if (Math.abs(A[j][pivot]) < eps)
          continue;
        double c = A[j][pivot];
        A[j][pivot] = 0;
        for (int k = pivot + 1; k < m; ++k)
          A[j][k] -= c * A[i][k];
      }
    }
  }
  public static void gauss(double[][] A) {
    gauss(A, Math.min(A.length, A[0].length));
  }
}
```

30 Monte Carlo Example: Finding Lines

```
import math
import random

EPSILON = 0.000001
def findinglines():
    # https://open.kattis.com/problems/findinglines
    # Task: Determine whether at least p > 20 percent of points are on one line

    # Monte Carlo approach: guess two points; they're both on the line
    # with probability at least 0.064. Count how many points are on the line
    # If more than p*n, return "possible." If no success after k trails,
    # answer is impossible with probability 1-(1-0.064)**k.

    # Step 0: Read input
    n, p = int(raw_input()), int(raw_input())
    limit = math.ceil(1.0*p*n/100)
    points = []
    for i in range(n): points.append(tuple(map(int, raw_input().split())))

    # Step 1: Take care of corner cases
    if len(points) <= 2: return "possible"

    # Step 2: Monte Carlo step with k trails
    k = 250
    for trail in range(k):
        a, b = random.sample(points, 2)
        if limit <= countonline(a,b,points): return "possible"

    # Answer is impossible with probability at least (1-(1-0.064)**k)
    return "impossible"


def countonline(a, b, points):
    # Will return how many points lie on the line determined by the distinct
    # points a and b
    count = 0
    for point in points: count += 1 if are_collinear(a, b, point) else 0
    return count

def are_collinear(a, b, c): return abs(trianglearea(a, b, c)) <= EPSILON

def trianglearea_db(a, b, c):
    ## Returns the twice the area of triangle ABC. Caller responsibility to
  ## divide by 2. Negative if points are given i counterclockwise order.
    return a[0] * (b[1]-c[1])  +  b[0] * (c[1]-a[1])  +  c[0] * (a[1]-b[1])

random.seed(213) ## For consistency in testing/debugging
print(findinglines())
```