Problem
The first two paragraphs (not counting this one) of this problem and "Juggle Struggle: Part 1" are identical. The prob
ems can otherwise be solved independently; you do not need to read or solve one in order to read or solve the other.

As manager of the Graceful Chainsaw Jugglers group, you have decided to spice the show up a bit. Instead of havin
 each juggler individually juggle their own chainsaws, you want them to form pairs, with each pair throwing the chai
saws back and forth to each other. In this new performance, 2 × N jugglers will be on stage at the same time, arrange
 into N pairs, with each juggler belonging to exactly one pair.

You think the show will be more impressive if the chainsaws being juggled by different pairs of jugglers are at risk
f collision. Let the stage be a two-dimensional plane, and let the straight line segment in that plane that connects the
ositions of two jugglers in a pair be called the pair's juggling path. When two juggling paths instersect, we say the ch
insaws juggled by those pairs are at risk of collision. We call the spatial positions and the pairings of the jugglers an
rrangement. An arrangement is magnificent if every two pairs of jugglers' chainsaws are at risk of collision. That is,
or the arrangement to be magnificent, each of the N juggling path segments must intersect each of the other N-1 jugg
ing path segments (but these intersections do not necessarily all have to be in the same place).

After some last minute fixes, you have what you think is a magnificent arrangement. Given the rush to put it togethe
, you want to write a checker that can determine whether it is indeed magnificent. If it is not, then at most 25 juggler
airs fail to intersect every other pair. You want your checker to report a list of all those juggler pairs for inspection.

Input
The first line of the input gives the number of test cases, T. T test cases follow. Each test case starts with one line co
taining a single integer N, the number of pairs of jugglers. Then, N lines follow. The i-th of these lines contains four
ntegers Xi, Yi, X'i, Y'i. (Xi, Yi) and (X'i, Y'i) are the coordinates of the positions of the two jugglers comprising the
-th juggler pair.

Output
For each test case, output one line containing Case #x: y, where y is uppercase MAGNIFICENT if the input represe
ts a magnificent arrangement. Otherwise, y should be a strictly increasing list of integers. Integer i should be on that
ist if and only if the juggling path of the i-th juggler pair fails to intersect at least one other juggling path.

Limits
Memory limit: 1GB.
-109 ≤ Xi ≤ 109, for all i.
-109 ≤ Yi ≤ 109, for all i.
-109 ≤ X'i ≤ 109, for all i.
-109 ≤ Y'i ≤ 109, for all i.
No three juggler positions are collinear. (Note that this also implies that no two jugglers are in the same position.)
For all but up to 25 pairs of jugglers, their juggling paths intersect all N - 1 other juggling paths.
Note: There may or may not exist a way to pair the jugglers such that the resulting arrangement is magnificent.

Test set 1 (Visible)
Time limit: 20 seconds.
1 ≤ T ≤ 100.
2 ≤ N ≤ 100.

Test set 2 (Hidden)
Time limit: 45 seconds.
1 ≤ T ≤ 13.
2 ≤ N ≤ 105.

Sample

Input

Output

```
4
2
-1 -1 -1 1
1 1 1 -1
2
-1 -1 1 1
-1 1 1 -1
4
1 2 4 2
2 1 3 1
2 4 3 0
3 3 2 3
3
1 1 2 2
3 7 4 8
8 3 9 3
```

```
Case #1: 1 2
Case #2: MAGNIFICENT
Case #3: 1 2 4
Case #4: 1 2 3
```

In Sample Case #1, there are only two pairs, and their paths do not cross.

In Sample Case #2, the arrangement is magnificent: every pair's path crosses every other pair's path.

In Sample Case #3, only pair 3's path crosses every other pair's path.

Solution:

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

using namespace std;

template <typename A, typename B>
string to_string(pair<A, B> p);

template <typename A, typename B, typename C>
string to_string(tuple<A, B, C> p);

template <typename A, typename B, typename C, typename D>
string to_string(tuple<A, B, C, D> p);
string to_string(const string& s) {
  return '"' + s + '"';
}

string to_string(const char* s) {
  return to_string((string) s);
}

string to_string(bool b) {
  return (b ? "true" : "false");
}

string to_string(vector<bool> v) {
  bool first = true;
  string res = "{";
  for (int i = 0; i < static_cast<int>(v.size()); i++) {
    if (!first) {
      res += ", ";
    }
    first = false;
    res += to_string(v[i]);
  }
  res += "}";
  return res;
}

template <size_t N>
string to_string(bitset<N> v) {
  string res = "";
  for (size_t i = 0; i < N; i++) {
    res += static_cast<char>('0' + v[i]);
  }
  return res;
}

template <typename A>
string to_string(A v) {
  bool first = true;
  string res = "{";
  for (const auto &x : v) {
    if (!first) {
      res += ", ";
    }
    first = false;
    res += to_string(x);
  }
  res += "}";
  return res;
}

template <typename A, typename B>
string to_string(pair<A, B> p) {
  return "(" + to_string(p.first) + ", " + to_string(p.second) + ")";
}

template <typename A, typename B, typename C>
string to_string(tuple<A, B, C> p) {
  return "(" + to_string(get<0>(p)) + ", " + to_string(get<1>(p)) + ", " + to_string(get<2>(p)) + ")";
}

template <typename A, typename B, typename C, typename D>
string to_string(tuple<A, B, C, D> p) {
  return "(" + to_string(get<0>(p)) + ", " + to_string(get<1>(p)) + ", " + to_string(get<2>(p)) + ", " + to_string(get<3
(p)) + ")";
}

void debug_out() { cerr << endl; }

template <typename Head, typename... Tail>
void debug_out(Head H, Tail... T) {
  cerr << " " << to_string(H);
  debug_out(T...);
}

#ifdef LOCAL
#define debug(...) cerr << "[" << #__VA_ARGS__ << "]:", debug_out(__VA_ARGS__)
#else
#define debug(...) 42
#endif

struct Point {
  int x;
  int y;
  int id;
};

inline int Side(int x, int y) {
  return (y > 0 || (y == 0 && x >= 0));
}

bool Good(const Point& a, const Point& b, const Point& c, const Point& d) {
  long long A = b.y - a.y;
  long long B = a.x - b.x;
  long long C = -A * a.x - B * a.y;
  long long z1 = A * c.x + B * c.y + C;
  long long z2 = A * d.x + B * d.y + C;
  return (z1 > 0 && z2 < 0) || (z1 < 0 && z2 > 0);
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  cout << fixed << setprecision(17);
  int tt;
  cin >> tt;
  for (int qq = 1; qq <= tt; qq++) {
    cout << "Case #" << qq << ":";
    int n;
    cin >> n;
    vector<Point> p(n), q(n);
    for (int i = 0; i < n; i++) {
      cin >> p[i].x >> p[i].y >> q[i].x >> q[i].y;
    }
    vector<int> bad;
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        if (i == j) {
          continue;
        }
        if (!Good(p[i], q[i], p[j], q[j]) || !Good(p[j], q[j], p[i], q[i])) {
          bad.push_back(i);
          break;
        }
      }
    }
    for (int x : bad) {
      cout << " " << x + 1;
    }
    if (bad.empty()) {
      cout << " MAGNIFICENT";
    }
    cout << '\n';
  }
  return 0;
}
```