GRAYPES

Algorithmic heart: Closest pair
==> Delaunay

denoting the number of g
easured in meter. Each po
, with $|x|, |y| < 2^{25}$. You r
o two graypes are at the sa
s, then it runs towards th

==> squared distances no problem with
double (53 bits mantissa)

GRAYPES

```
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Exact_predicates_exact_constructions_kernel_with_sqrt.h>
#include <CGAL/Delaunay_triangulation_2.h>
#include <vector>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Exact_predicates_exact_constructions_kernel_with_sqrt::FT EFT;
typedef CGAL::Delaunay_triangulation_2<K> Triangulation;

double ceil_to_double(const EFT& x)
{ ... }

int main()
{
  for (std::size_t n; std::cin >> n && n > 0;) {
    std::vector<K::Point_2> pts;
    pts.reserve(n);
    for (std::size_t i = 0; i < n; ++i) {
      K::Point_2 p;
      std::cin >> p;
      pts.push_back(p);
    }
    Triangulation t;
    t.insert(pts.begin(), pts.end());
    Triangulation::Finite_edges_iterator e = t.finite_edges_begin();
    K::FT minl = t.segment(*e).squared_length();
    while (++e != t.finite_edges_end())
      minl = std::min(minl, t.segment(*e).squared_length());
    std::cout << ceil_to_double(50*sqrt(EFT(minl))) << std::endl;
  }
  return 0;
}
```

GERMS

Algorithmic heart: nearest neighbor graph
==> Delaunay

Twist: for each vertex, we need to find (and save)
its nearest neighbor/squared distance

=> map or store at face directly

GERMS

```
typedef CGAL::Exact_predicates_inexact_constructions_kernel  K;
typedef CGAL::Triangulation_vertex_base_with_info_2<K::FT,K> Vb;
typedef CGAL::Triangulation_face_base_2<K>                   Fb;
typedef CGAL::Triangulation_data_structure_2<Vb,Fb>          Tds;
typedef CGAL::Delaunay_triangulation_2<K,Tds>                Delaunay;
typedef Delaunay::Finite_vertices_iterator                   VI;
typedef Delaunay::Finite_edges_iterator                      EI;

void simulate(std::size_t n) {
  std::vector<K::Point_2> pts;
  // read input ...

  // info (-> squared distance to nearest neighbor, initially, the boundary)
  Delaunay dt;
  dt.insert(pts.begin(), pts.end());
  for (VI v = dt.finite_vertices_begin(); v != dt.finite_vertices_end(); ++v) {
    v->info() = std::min(std::min(v->point().x() - l, r - v->point().x()),
                         std::min(v->point().y() - b, t - v->point().y()));
    v->info() *= v->info();
  }

  // compute all nearest neighbors
    for (EI e = dt.finite_edges_begin(); e != dt.finite_edges_end(); ++e) {
    Delaunay::Vertex_handle v1 = e->first->vertex(dt.cw(e->second));
    Delaunay::Vertex_handle v2 = e->first->vertex(dt.ccw(e->second));
    K::FT d = CGAL::squared_distance(v1->point(), v2->point()) / 4;
    v1->info() = std::min(v1->info(), d);
    v2->info() = std::min(v2->info(), d);
  }

  // build lifetime vector
  std::vector<K::FT> ltv;
  for (VI v = dt.finite_vertices_begin(); v != dt.finite_vertices_end(); ++v)
    ltv.push_back(v->info());
  std::sort(ltv.begin(), ltv.end());
  std::cout << hours(ltv[0]) << " " << hours(ltv[ltv.size()/2]) << " "
            << hours(ltv.back()) << "\n";
}
```

```
// convert squared radius to hours
// (rounded up to next integer)
K::FT hours(K::FT)
```

BISTRO

Algorithmic heart: Post office problem
==> Delaunay

Twist: Query preprocessing using spatial sort.

BISTRO

```cpp
#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Delaunay_triangulation_2.h>
#include <vector>

typedef CGAL::Exact_predicates_inexact_constructions_kernel K;
typedef CGAL::Delaunay_triangulation_2<K> Delaunay;

void find_nearest(std::size_t n) {
  std::vector<K::Point_2> pts;
  pts.reserve(n);
  for (std::size_t i = 0; i < n; ++i) {
    K::Point_2 p;
    std::cin >> p;
    pts.push_back(p);
  }
  Delaunay t;
  t.insert(pts.begin(), pts.end());
  std::size_t m;
  std::cin >> m;
  for (std::size_t i = 0; i < m; ++i) {
    K::Point_2 p;
    std::cin >> p;
    std::cout << CGAL::squared_distance(p, t.nearest_vertex(p)->point()) << "\n";
  }
}

int main()
{
  std::cout << std::setiosflags(std::ios::fixed) << std::setprecision(0);
  for (std::size_t n; std::cin >> n && n > 0;)
    find_nearest(n);
  return 0;
}
```

Results for each test-set

	Name	Result	Points	CPU Time
1	n<2000	CORRECT	20	0.06s
2	n<25000	CORRECT	25	0.275s
3	n<100000	CORRECT	30	0.258s
4	special	CORRECT	15	0.257s
5	manyq	TIMELIMIT	0	0.6s

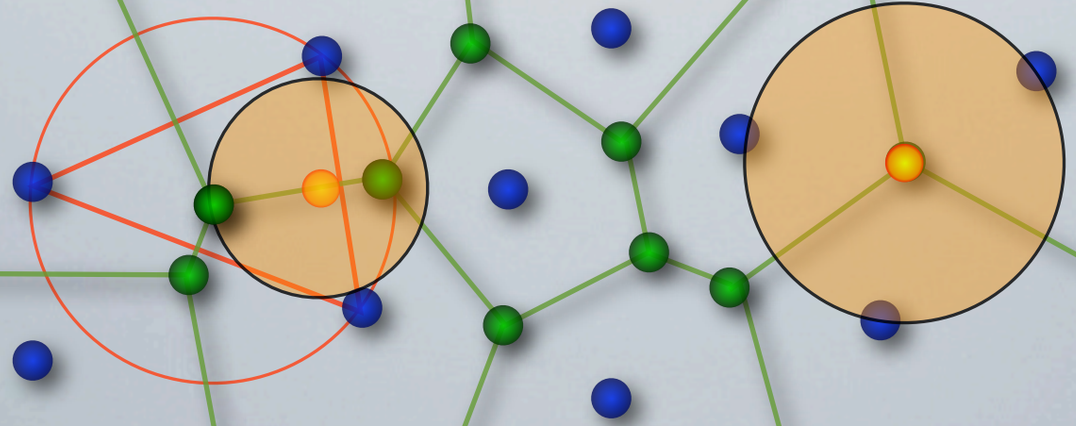
BISTRO

```
struct Mypoint : public K::Point_2 {
  Mypoint(K::FT x, K::FT y, std::size_t i = 0)
    : K::Point_2(x, y), index(i)
  {}
  std::size_t index;
};


std::vector<Mypoint> qpts;
qpts.reserve(m);
for (std::size_t i = 0; i < m; ++i) {
  K::FT x, y;
  std::cin >> x >> y;
  qpts.push_back(Mypoint(x, y, i));
}
CGAL::spatial_sort(qpts.begin(), qpts.end());
std::vector<K::FT> result(m);
Delaunay::Face_handle fh = t.infinite_face();
for (std::vector<Mypoint>::const_iterator i = qpts.begin();
     i != qpts.end();
     ++i)
  {
    Delaunay::Vertex_handle vh = t.nearest_vertex(*i, fh);
    result[i->index] = CGAL::squared_distance(*i, vh->point());
    fh = vh->face();
  }
for (std::size_t i = 0; i < m; ++i)
  std::cout << result[i] << "\n";
```

HINT

Where to go first?
Not to the closest Voronoi vertex, but to the safest!
(remember the "empty circumcircle" property)



locked...

How to move a disk D without colliding with a given point set P?

Hint: If you do not need to construct the path, working with the dual Delaunay triangulation instead is much more efficient.

HINI

Algorithmic heart: Motion planning with a circle among point obstacles
==> Voronoi/Delaunay

Twist: During the search we need to mark faces as visited to avoid cycling.

=> map or store at face directly

HINI

```
typedef CGAL::Triangulation_vertex_base_2<K>                    Vb;
typedef CGAL::Triangulation_face_base_with_info_2<int,K>        Fb;
typedef CGAL::Triangulation_data_structure_2<Vb,Fb>             Tds;
typedef CGAL::Delaunay_triangulation_2<K,Tds>                   Delaunay;

try {
    if (r <= 0) throw true;
    Face_handle f = t.locate(s);
    if (CGAL::squared_distance(s, t.nearest_vertex(s, f)->point()) < r)
        throw false;
    // DFS
    std::vector<Face_handle> stack;
    stack.push_back(f);
    f->info() = i;
    while (!stack.empty()) {
      f = stack.back();
      stack.pop_back();
      if (t.is_infinite(f)) throw true;
      for (int j = 0; j < 3; ++j)
        if (f->neighbor(j)->info() < i &&
            t.segment(Delaunay::Edge(f,j)).squared_length() >= 4 * r) {
          stack.push_back(f->neighbor(j));
          f->neighbor(j)->info() = i;
        }
    }
    throw false;
}
catch (bool solvable)  { std::cout << (solvable ? "y" : "n"); }
```

```
CGAL::squared_distance(f->vertex(f->cw(j))->point(),
                       f->vertex(f->ccw(j))->point())
```