### Week 19: Computational Geometry Advanced Topics

**Topics:** - Convex Hull Algorithms: Graham Scan, Andrew’s Monotone Chain - Rotating Calipers for diameters and width - Line Intersection & Segment Intersection - Polygon Area, Centroid, and Point in Polygon - Sweep Line Algorithms for intersections and closest pairs

**Weekly Tips:** - Always represent points using structures/classes for clarity. - Pay attention to precision; use integers where possible to avoid floating point errors. - Rotating Calipers technique helps in computing diameters, widths, or farthest pairs. - Sweep line is powerful for interval or event-based geometric problems. - Practice with both integer and floating-point geometric problems.

**Problem 1: Convex Hull (Graham Scan)** **Link:** [CSES Convex Hull](https://cses.fi/problemset/task/2165/) **Difficulty:** Intermediate

**C++ Solution with Explanation Comments:**

#include <bits/stdc++.h>  
using namespace std;  
struct Point { long long x,y; };  
bool cmp(Point a, Point b){ return a.x==b.x?a.y<b.y:a.x<b.x; }  
long long cross(Point O, Point A, Point B){  
 return (A.x-O.x)\*(B.y-O.y)-(A.y-O.y)\*(B.x-O.x);  
}  
vector<Point> convexHull(vector<Point> P){  
 int n=P.size(); sort(P.begin(),P.end(),cmp);  
 vector<Point> H(2\*n);  
 int k=0;  
 for(int i=0;i<n;i++){  
 while(k>=2 && cross(H[k-2],H[k-1],P[i])<=0) k--;  
 H[k++]=P[i];  
 }  
 for(int i=n-2,t=k+1;i>=0;i--){  
 while(k>=t && cross(H[k-2],H[k-1],P[i])<=0) k--;  
 H[k++]=P[i];  
 }  
 H.resize(k-1); return H;  
}  
int main(){  
 int n; cin>>n;  
 vector<Point> P(n);  
 for(int i=0;i<n;i++) cin>>P[i].x>>P[i].y;  
 vector<Point> hull=convexHull(P);  
 cout<<hull.size()<<endl;  
}

**Explanation Comments:** - Sort points lexicographically. - Build lower and upper hull using cross product to maintain convexity. - Resulting hull represents minimal convex polygon enclosing all points.

**Problem 2: Line Segment Intersection** **Link:** [GeeksforGeeks Line Intersection](https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/) **Difficulty:** Intermediate

**C++ Solution with Explanation Comments:**

#include <bits/stdc++.h>  
using namespace std;  
struct Point{ int x,y; };  
int orientation(Point p, Point q, Point r){  
 int val=(q.y-p.y)\*(r.x-q.x)-(q.x-p.x)\*(r.y-q.y);  
 if(val==0) return 0;  
 return (val>0)?1:2;  
}  
bool onSegment(Point p, Point q, Point r){  
 return q.x<=max(p.x,r.x) && q.x>=min(p.x,r.x) && q.y<=max(p.y,r.y) && q.y>=min(p.y,r.y);  
}  
bool intersect(Point p1,Point q1,Point p2,Point q2){  
 int o1=orientation(p1,q1,p2), o2=orientation(p1,q1,q2);  
 int o3=orientation(p2,q2,p1), o4=orientation(p2,q2,q1);  
 if(o1!=o2 && o3!=o4) return true;  
 if(o1==0 && onSegment(p1,p2,q1)) return true;  
 if(o2==0 && onSegment(p1,q2,q1)) return true;  
 if(o3==0 && onSegment(p2,p1,q2)) return true;  
 if(o4==0 && onSegment(p2,q1,q2)) return true;  
 return false;  
}  
int main(){  
 Point p1,p2,q1,q2;  
 cin>>p1.x>>p1.y>>q1.x>>q1.y;  
 cin>>p2.x>>p2.y>>q2.x>>q2.y;  
 cout<<(intersect(p1,q1,p2,q2)?"YES":"NO")<<endl;  
}

**Explanation Comments:** - Use orientation to determine relative position of points. - Check special cases where points are collinear and on segment. - Returns true if segments intersect.

**End of Week 19** - Master advanced computational geometry techniques. - Practice convex hull, segment intersections, and geometric queries. - These skills are crucial for ACM-ICPC and geometry-heavy contests.