### Week 30: Computational Geometry – Advanced Structures & Queries

**Topics:** - Segment Tree for Points (2D Range Queries) - KD-Tree for Nearest Neighbor Search - Range Trees and Orthogonal Range Queries - Dynamic Convex Hull (Insert/Delete Queries) - Line Sweep with Events and Intersections - Geometric Data Structures for Optimization Problems

**Weekly Tips:** - 2D segment trees allow efficient range sum or count queries for points. - KD-Tree partitions space recursively for nearest neighbor and range search. - Range Trees support orthogonal range queries efficiently. - Dynamic convex hull handles insertion/deletion and queries like extreme points. - Sweep line techniques are crucial for counting intersections and handling events efficiently.

**Problem 1: 2D Range Count with Segment Tree** **Link:** [Codeforces Example](https://codeforces.com/problemset/problem/220/E) **Difficulty:** Advanced

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

#include <bits/stdc++.h>  
using namespace std;  
struct Node{ int y; };  
vector<vector<int>> seg;  
int n;  
void build(vector<pair<int,int>>& pts,int v,int tl,int tr){  
 if(tl==tr){ seg[v]={pts[tl].second}; }  
 else{  
 int tm=(tl+tr)/2;  
 build(pts,2\*v,tl,tm); build(pts,2\*v+1,tm+1,tr);  
 merge(seg[2\*v].begin(),seg[2\*v].end(),seg[2\*v+1].begin(),seg[2\*v+1].end(),back\_inserter(seg[v]));  
 }  
}  
int query(int v,int tl,int tr,int l,int r,int y1,int y2){  
 if(l>r) return 0;  
 if(tl==l && tr==r) return upper\_bound(seg[v].begin(),seg[v].end(),y2)-lower\_bound(seg[v].begin(),seg[v].end(),y1);  
 int tm=(tl+tr)/2;  
 return query(2\*v,tl,tm,l,min(r,tm),y1,y2)+query(2\*v+1,tm+1,tr,max(l,tm+1),r,y1,y2);  
}  
int main(){  
 cin>>n; vector<pair<int,int>> pts(n);  
 for(int i=0;i<n;i++) cin>>pts[i].first>>pts[i].second;  
 sort(pts.begin(),pts.end());  
 seg.resize(4\*n);  
 build(pts,1,0,n-1);  
 int q; cin>>q;  
 while(q--){ int x1,x2,y1,y2; cin>>x1>>x2>>y1>>y2;  
 int l=lower\_bound(pts.begin(),pts.end(),make\_pair(x1,-INT\_MAX))-pts.begin();  
 int r=upper\_bound(pts.begin(),pts.end(),make\_pair(x2,INT\_MAX))-pts.begin()-1;  
 cout<<query(1,0,n-1,l,r,y1,y2)<<endl;  
 }  
}

**Explanation Comments:** - Build segment tree by x-coordinate; each node stores sorted y-coordinates. - Query combines binary search on y-coordinates for 2D range counting. - Efficient O(log^2 n) per query.

**Problem 2: KD-Tree Nearest Neighbor Search** **Link:** [CP-Algorithms KD-Tree](https://cp-algorithms.com/data_structures/kd_tree.html) **Difficulty:** Advanced

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

#include <bits/stdc++.h>  
using namespace std;  
struct Point{ int x,y; };  
struct Node{ Point p; Node\* left; Node\* right; };  
Node\* build(vector<Point>& pts,int depth){  
 if(pts.empty()) return nullptr;  
 int axis=depth%2;  
 sort(pts.begin(),pts.end(),[axis](Point a,Point b){ return axis? a.y<b.y : a.x<b.x; });  
 int mid=pts.size()/2;  
 Node\* node=new Node{pts[mid],nullptr,nullptr};  
 vector<Point> left\_pts(pts.begin(),pts.begin()+mid);  
 vector<Point> right\_pts(pts.begin()+mid+1,pts.end());  
 node->left=build(left\_pts,depth+1);  
 node->right=build(right\_pts,depth+1);  
 return node;  
}  
long long sqDist(Point a,Point b){ return 1LL\*(a.x-b.x)\*(a.x-b.x)+1LL\*(a.y-b.y)\*(a.y-b.y); }  
long long nearest(Node\* root,Point target,int depth,long long best){  
 if(!root) return best;  
 long long d=sqDist(root->p,target); best=min(best,d);  
 int axis=depth%2;  
 Node \*first=root->left,\*second=root->right;  
 if((axis? target.y>root->p.y : target.x>root->p.x)) swap(first,second);  
 best=nearest(first,target,depth+1,best);  
 if((axis? abs(target.y-root->p.y) : abs(target.x-root->p.x))\*(axis? abs(target.y-root->p.y) : abs(target.x-root->p.x))<best)  
 best=nearest(second,target,depth+1,best);  
 return best;  
}  
int main(){  
 int n; cin>>n; vector<Point> pts(n);  
 for(int i=0;i<n;i++) cin>>pts[i].x>>pts[i].y;  
 Node\* root=build(pts,0);  
 Point target; cin>>target.x>>target.y;  
 cout<<nearest(root,target,0,LLONG\_MAX)<<endl;  
}

**Explanation Comments:** - KD-Tree recursively partitions points along x/y axis. - Nearest neighbor query searches closest point efficiently. - Prune branches using distance to splitting line. - Efficient for high-dimensional nearest neighbor search.

**End of Week 30** - Mastering advanced geometric structures enables efficient range queries, nearest neighbor search, and dynamic convex hull operations. - Practice 2D segment trees, KD-Trees, range trees, and sweep line techniques for ACM-ICPC contests.