CDQ 分治，偏序问题

CDQ 分治

Idea：普通的分治中子问题之间互不影响，CDQ 分治中左区间可以对右区间产生影响。**必须离线**。

Steps：分 $\Rightarrow$ 递归处理左、右区间 $\Rightarrow$ 处理左区间对右区间的影响，调整答案

Application：点对有关问题（偏序，动态逆序对），1D/1D 动态规划的优化……

二维偏序

Idea#1：先按第一维排序，然后第二维值域树状数组维护

Idea#2：先按第一维排序，然后 CDQ 分治

注：逆序对的两种实现分别对应上述两种做法

三维偏序

Idea#1：先按第一维排序，

Idea#2：先按第一维排序，然后 CDQ 分治：`solve(l, r)` 时，递归进行 `solve(l, mid)` 和 `solve(mid+1, r)`，考虑如何统计 $l\leq i\leq mid, mid+1\leq j\leq r$ 的满足 $b_i<b_j$ 且 $c_i<c_j$ 的点对数。把所有 $[l,r]$ 区间的元素拿出来按第二维排序，遍历一遍，遇到左区间的就按第三维丢到值域树状数组里去，遇到右区间的就查询。

ATT：如果题目存在重复元素，需要去重（因为 CDQ 分治只能统计左区间对右区间的答案），丢值域树状数组的时候丢重复次数。

Complexity：$T(n)=2\cdot T\left(\frac{n}{2}\right)+O(n\lg n)=O(n\lg^2 n)$

Code（有重复元素）：

```cpp
#include<algorithm>
#include<cstdio>

using namespace std;

const int N = 100005;
const int K = 200005;

int n, k, tot, ans[N];
struct Node{
    int a, b, c, dc;    // dc is c after discretization
    int newid, cnt, num; // num is the number of elements which are less than this node
}node[N];
bool cmpa(const Node &A, const Node &B){
    return A.a == B.a ? (A.b == B.b ? A.c < B.c : A.b < B.b) : A.a < B.a;
}
bool cmpb(const Node &A, const Node &B){
    return A.b == B.b ? (A.c == B.c ? A.a < B.a : A.c < B.c) : A.b < B.b;
}

int tc[N];
void disc(){
    sort(tc+1, tc+tc[0]+1);
    tc[0] = unique(tc+1, tc+tc[0]+1) - (tc+1);
    for(int i = 1; i <= n; i++)
        node[i].dc = lower_bound(tc+1, tc+tc[0]+1, node[i].c) - tc;
}

int c[N];
inline int lowbit(int x){ return x & -x; }
void add(int x, int val){ while(x <= n){ c[x] += val; x += lowbit(x); } }
int sum(int x){
    int res = 0;
    while(x){ res += c[x]; x -= lowbit(x); }
    return res;
}

void cdq(int l, int r){
    if(l == r)  return;
    int mid = (l + r) >> 1;
    cdq(l, mid);
    cdq(mid+1, r);
    sort(node+l, node+r+1, cmpb);
    for(int i = l; i <= r; i++){
        if(node[i].newid <= mid)    add(node[i].dc, node[i].cnt);
        else    node[i].num += sum(node[i].dc);
    }
    for(int i = l; i <= r; i++)         // clear BIT
        if(node[i].newid <= mid)
            add(node[i].dc, -node[i].cnt);
}

int main(){
    scanf("%d%d", &n, &k);
    for(int i = 1; i <= n; i++){
        scanf("%d%d%d", &node[i].a, &node[i].b, &node[i].c);
        tc[++tc[0]] = node[i].c;
    }
    disc();
    sort(node+1, node+n+1, cmpa);
    for(int i = 1; i <= n; i++){ // unique
        if(node[i-1].a != node[i].a || node[i-1].b != node[i].b || node[i-1].c != node[i].c)
            node[++tot] = node[i], node[tot].newid = tot, node[tot].cnt = 1;
        else    node[tot].cnt++;
    }
    cdq(1, tot);
    for(int i = 1; i <= tot; i++)   ans[node[i].num + node[i].cnt - 1] += node[i].cnt;
    for(int i = 0; i < n; i++)  printf("%d\n", ans[i]);
    return 0;
}
```

四维偏序

Idea：CDQ 套 CDQ分治。先按第一维 a 排序，然后对第一维 CDQ 分治：递归回来后根据第一维的位置打上左右标记，然后按 b 排序，得到一系列形如 $(L/R,b,c,d)$ 的元素且 b 递增；复制一下，对第二维 CDQ 分治并在同时对第三维排序：递归回来后左子区间都是 $(L/R,L,c,d)$，右子区间都是 $(L/R,R,c,d)$，且各自区间内的 c 是递增的，然后双指针把第四维丢值域树状数组里或者查询，丢的时候记得参考第一维的 L/R 情况。

实现时用与 CDQ 浑然一体的归并排序而非 `sort` 来减少一些常数。

ATT：清空树状数组时能少清就少清，这对减少常数很关键。

Complexity：$T(n)=2\cdot T\left(\frac{n}{2}\right)+O(n\lg^2 n)=O(n\lg^3 n)$

Code：

```
#include<algorithm>
#include<cstdio>

using namespace std;

template<typename T>void read(T&x){x=0;int fl=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')
fl=-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=(x<<1)+(x<<3)+ch-'0';ch=getchar();}x*=fl;}
template<typename T,typename...Args>inline void read(T&t,Args&...args){read(t);read(args...);}

const int N = 50005;

int T, q, a[N], tot, ans[N];

#define LEFT 0
#define RIGHT 1
struct Node{
    int t, x, y, z, k;
    bool mark;
}node[N<<3];
bool cmpt(const Node &A, const Node &B){
```

```cpp
    if(A.t == B.t) if(A.x == B.x) if(A.y == B.y) return A.z < B.z;
    else return A.y < B.y; else return A.x < B.x; else return A.t < B.t;
}

int t[N<<3];
void disc(){
    sort(t+1, t+t[0]+1);
    t[0] = unique(t+1, t+t[0]+1) - (t+1);
    for(int i = 1; i <= tot; i++)
        node[i].z = lower_bound(t+1, t+t[0]+1, node[i].z) - t;
}

int c[N<<3];
inline int lowbit(int x){ return x & -x; }
inline void add(int x, int val){ while(x <= q * 8){ c[x] += val; x += lowbit(x); } }
inline int sum(int x){ int res = 0; while(x){ res += c[x]; x -= lowbit(x); } return res; }

Node tmp[N<<3], tmp2[N<<3];
void cdq2(int l, int r){
    if(l == r)  return;
    int mid = (l + r) >> 1;
    cdq2(l, mid), cdq2(mid+1, r);
    int ptl = l, ptr = mid+1, tid = l-1, lastl;
    while(ptl <= mid && ptr <= r){
        if(tmp[ptl].y <= tmp[ptr].y){
            if(tmp[ptl].k == 0 && tmp[ptl].mark == LEFT)
                add(tmp[ptl].z, 1);
            tmp2[++tid] = tmp[ptl++];
        }
        else{
            if(tmp[ptr].k != 0 && tmp[ptr].mark == RIGHT)
                ans[tmp[ptr].t] += tmp[ptr].k * sum(tmp[ptr].z);
            tmp2[++tid] = tmp[ptr++];
        }
    }
    lastl = ptl - 1;
    while(ptl <= mid)   tmp2[++tid] = tmp[ptl++];
    while(ptr <= r){
        if(tmp[ptr].k != 0 && tmp[ptr].mark == RIGHT)
            ans[tmp[ptr].t] += tmp[ptr].k * sum(tmp[ptr].z);
        tmp2[++tid] = tmp[ptr++];
    }
    for(int i = l; i <= lastl; i++) // crucial for decreasing constant
        if(tmp[i].k == 0 && tmp[i].mark == LEFT)
            add(tmp[i].z, -1);
    for(int i = l; i <= r; i++) tmp[i] = tmp2[i];
}
void cdq1(int l, int r){
    if(l == r)  return;
    int mid = (l + r) >> 1;
    cdq1(l, mid), cdq1(mid+1, r);
    int ptl = l, ptr = mid+1, tid = l-1; // tid must be l-1 or it can't be used in cdq2
    while(ptl <= mid && ptr <= r){
        if(node[ptl].x <= node[ptr].x){
            node[ptl].mark = LEFT;
            tmp[++tid] = node[ptl++];
        }
        else{
            node[ptr].mark = RIGHT;
            tmp[++tid] = node[ptr++];
        }
    }
    while(ptl <= mid)   node[ptl].mark = LEFT, tmp[++tid] = node[ptl++];
    while(ptr <= r)     node[ptr].mark = RIGHT, tmp[++tid] = node[ptr++];
    for(int i = l; i <= r; i++)   node[i] = tmp[i];
    cdq2(l, r);
}

inline void init(){
    tot = t[0] = 0;
    for(int i = 1; i <= q; i++) ans[i] = 0;
}

int main(){
    read(T);
    while(T--){
        read(q);
        init();
        int _x1, _y1, _z1, _x2, _y2, _z2;
        for(int i = 1; i <= q; i++){
            read(a[i], _x1, _y1, _z1);
            if(a[i] == 1){
                node[++tot] = (Node){i, _x1, _y1, _z1, 0};
                t[++t[0]] = _z1;
            }
            else{
                read(_x2, _y2, _z2);
                node[++tot] = (Node){i, _x2, _y2, _z2, 1};
                node[++tot] = (Node){i, _x1-1, _y2, _z2, -1};
                node[++tot] = (Node){i, _x2, _y1-1, _z2, -1};
                node[++tot] = (Node){i, _x2, _y2, _z1-1, -1};
                node[++tot] = (Node){i, _x1-1, _y1-1, _z2, 1};
                node[++tot] = (Node){i, _x1-1, _y2, _z1-1, 1};
                node[++tot] = (Node){i, _x2, _y1-1, _z1-1, 1};
                node[++tot] = (Node){i, _x1-1, _y1-1, _z1-1, -1};
                t[++t[0]] = _z2, t[++t[0]] = _z1-1;
            }
        }
        disc();
        sort(node+1, node+tot+1, cmpt);
        cdq1(1, tot);
        for(int i = 1; i <= q; i++)
            if(a[i] == 2)
                printf("%d\n", ans[i]);
    }
    return 0;
}
```