Multiple Of three:

In this problem I have to find how many numbers are divisible by three in A given range.

Two types of operation :

Update: add one with all the elements in the given range.

Idea:

I have taken a structure with lazy(to track how many times a node added by one),number of value which have reminder value 1,2,0 .

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

using namespace std;


#define ll long long


struct st

{

   ll lazy,zero,one,two;

} tree[100005 * 4];


void init(ll node, ll b, ll e)

{

   if(b == e)

   {

      tree[node].zero = 1;

      tree[node].one = 0;

      tree[node].two = 0;

      return;

   }

   ll left = node * 2;

   ll right = node * 2 + 1;

   ll mid = (b + e) / 2;

   init(left,b,mid);

   init(right,mid + 1, e);

   tree[node].zero = (e - b + 1);

   tree[node].one = 0;

   tree[node].two = 0;

   tree[node].lazy = 0;

}


void update(ll node, ll b, ll e, ll i, ll j,ll carry)

{

// cout<<"update"<<endl;


if(b>j || e<i)

{

   if(carry == 0) return;

   ll x = carry;

   x = x%3;


   ll z = tree[node].zero;

   ll o = tree[node].one;

   ll t = tree[node].two;

   if(x == 1)

   {

      tree[node].zero = t;

      tree[node].one = z;

      tree[node].two = o;

   }

   else if(x == 2)

   {

      tree[node].zero = o;

      tree[node].one = t;

      tree[node].two = z;

   }

   tree[node].lazy+=carry;

   return;

}


if(b>=i && e<=j)

{

  ll x = carry + 1;

   x = x%3;


   ll z = tree[node].zero;

   ll o = tree[node].one;

   ll t = tree[node].two;

   if(x%3 == 1)

   {

      tree[node].zero = t;

      tree[node].one = z;

      tree[node].two = o;
```

```
        }


        else if(x%3 == 2)
        {
            tree[node].zero = o;
            tree[node].one = t;
            tree[node].two = z;
        }
 tree[node].lazy+=carry+1;
        return;
    }


    ll left = node * 2;
    ll right = node * 2 + 1;
    ll mid = (b + e)/2;


//    tree[left].lazy+=tree[node].lazy;
//    tree[right].lazy+=tree[node].lazy;




    update(left,b,mid,i,j,carry + tree[node].lazy);
    update(right,mid + 1, e, i, j,carry + tree[node].lazy);
    tree[node].lazy = 0;


    tree[node].zero = tree[left].zero + tree[right].zero;
    tree[node].one = tree[left].one + tree[right].one;
    tree[node].two = tree[left].two + tree[right].two;


}


ll query(ll node,ll b, ll e, ll i , ll j, ll carry)
{
    // cout<<"Query"<<endl;
    if(b>j || e<i)
    {
        return 0;
    }
    if(b>=i && e<=j)
    {
        if(carry%3  == 0)
            return tree[node].zero ;


        else if(carry%3 == 1)
            return tree[node].two;
        else
            return tree[node].one;
    }


    ll left = node * 2;
    ll right = node * 2 + 1;
    ll mid = (b + e) / 2;
    ll x =  query(left,b,mid,i,j,tree[node].lazy + carry);
    ll y =   query(right,mid + 1, e,i,j,tree[node].lazy + carry);
    return x + y;
}


int main()
{
    ll t,n,q,w = 0;
    scanf("%lld",&t);
    while(t--)
    {
        memset(tree,0,sizeof(tree));
        scanf("%lld %lld",&n,&q);
        init(1,0,n-1);
//       for(int i=1; i<=6; i++)
//       {
//           cout<<"num = "<<i<<" lazy = "<<tree[i].lazy<<"zero = 
"<<tree[i].zero<<" one = "<<tree[i].one<<" two = "<<tree[i].two<<endl;
//       }
        printf("Case %lld:\n",++w);
        while(q--)
        {
            ll type,frm,to;
            scanf("%lld %lld %lld",&type,&frm,&to);
            if(type == 0)
            {
                update(1,0,n-1,frm,to,0);
//               for(int i=1; i<=6; i++)
//               {
```

```
//                      cout<<"num = "<<i<<" lazy = "<<tree[i].lazy<<"zero =
"<<tree[i].zero<<" one = "<<tree[i].one<<" two = "<<tree[i].two<<endl;

//                }

            }

            else printf("%lld\n",query(1,0,n-1,frm,to,0));

        }

    }

    return 0;

}
```