#5710题目描述-优先队列+结构体pair

5710. 积压订单中的订单总数

给你一个二维整数数组 `orders` ，其中每个 `orders[i] = [price_i, amount_i, orderType_i]` 表示有 `amount_i` 笔类型为 `orderType_i` 、价格为 `price_i` 的订单。

订单类型 `orderType_i` 可以分为两种：

- `0` 表示这是一批采购订单 `buy`
- `1` 表示这是一批销售订单 `sell`

注意，`orders[i]` 表示一批共计 `amount_i` 笔的独立订单，这些订单的价格和类型相同。对于所有有效的 `i` ，由 `orders[i]` 表示的所有订单提交时间均早于 `orders[i+1]` 表示的所有订单。

订单类型 $orderType_i$ 可以分为两种：

- `0` 表示这是一批采购订单 `buy`
- `1` 表示这是一批销售订单 `sell`

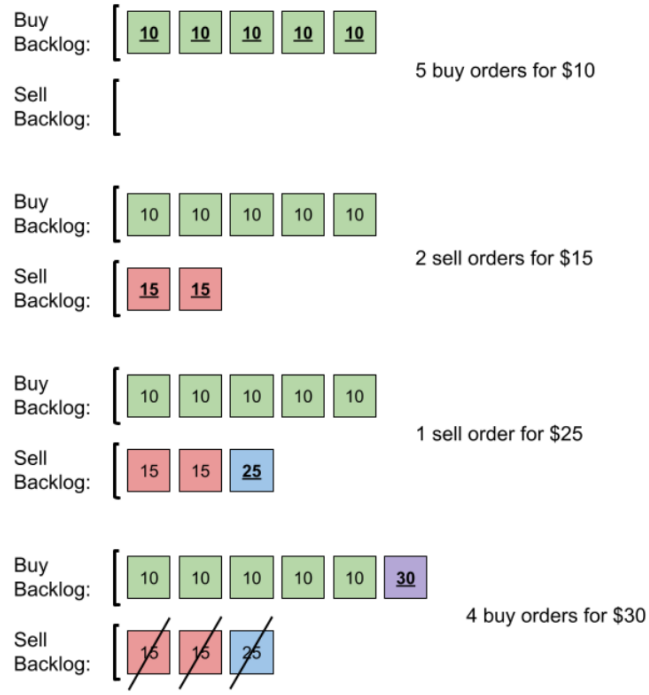
注意，`orders[i]` 表示一批共计 $amount_i$ 笔的独立订单，这些订单的价格和类型相同。对于所有有效的 `i` ，由 `orders[i]` 表示的所有订单提交时间均早于 `orders[i+1]` 表示的所有订单。

存在由未执行订单组成的 **积压订单** 。积压订单最初是空的。提交订单时，会发生以下情况：

- 如果该订单是一笔采购订单 `buy` ，则可以查看积压订单中价格 **最低** 的销售订单 `sell` 。如果该销售订单 `sell` 的价格 **低于或等于** 当前采购订单 `buy` 的价格，则匹配并执行这两笔订单，并将销售订单 `sell` 从积压订单中删除。否则，采购订单 `buy` 将会添加到积压订单中。
- 反之亦然，如果该订单是一笔销售订单 `sell` ，则可以查看积压订单中价格 **最高** 的采购订单 `buy` 。如果该采购订单 `buy` 的价格 **高于或等于** 当前销售订单 `sell` 的价格，则匹配并执行这两笔订单，并将采购订单 `buy` 从积压订单中删除。否则，销售订单 `sell` 将会添加到积压订单中。

输入所有订单后，返回积压订单中的 **订单总数** 。由于数字可能很大，所以需要返回对 $10^9 + 7$ 取余的结果。

示例 1：




```
输入：orders = [[10,5,0],[15,2,1],[25,1,1],[30,4,0]]
输出：6
解释：输入订单后会发生下述情况：
- 提交 5 笔采购订单，价格为 10 。没有销售订单，所以这 5 笔订单添加到积压订单中。
- 提交 2 笔销售订单，价格为 15 。没有采购订单的价格大于或等于 15 ，所以这 2 笔订单添加到积压订单中。
- 提交 1 笔销售订单，价格为 25 。没有采购订单的价格大于或等于 25 ，所以这 1 笔订单添加到积压订单中。
- 提交 4 笔采购订单，价格为 30 。前 2 笔采购订单与价格最低（价格为 15）的 2 笔销售订单匹配，从积压订单中删除这 2 笔销售订
最终，积压订单中有 5 笔价格为 10 的采购订单，和 1 笔价格为 30 的采购订单。所以积压订单中的订单总数为 6 。
```

#比赛时超时代码

```
class Solution {
public:
    int getNumberOfBacklogOrders(vector<vector<int>>& orders) {
        vector<vector<int>>buy_backlog;
        vector<vector<int>>sell_backlog;
        for(int i=0;i<orders.size();i++){
            if(orders[i][2]==0)//表示采购订单
            {
                int amount=orders[i][1];
                int price=orders[i][0];
                for(int k=0;k<sell_backlog.size();k++){
                    if(sell_backlog[k][0]<=price){
                        if(sell_backlog[k][1]>=amount){
                            sell_backlog[k][1]-=amount;
                            amount=0;
                            break;
                        }
                        amount-=sell_backlog[k][1];
                        //删除该订单
                        sell_backlog[k][1]=0;
```

```
//sell_backlog.erase(sell_backlog.begin()+k,sell_backlog.begin()+k+1);
                    }
                }
                //当amount!=0
                if(amount!=0){
                    vector<int>temp;
                    temp.push_back(price);
                    temp.push_back(amount);
                    buy_backlog.push_back(temp);
                    sort(buy_backlog.rbegin(),buy_backlog.rend());
                }
            }
            else if(orders[i][2]==1)//表示销售订单
            {
                int amount=orders[i][1];
                int price=orders[i][0];
                for(int k=0;k<buy_backlog.size();k++){
                    if(buy_backlog[k][0]>=price){
                        if(buy_backlog[k][1]>=amount){
                            buy_backlog[k][1]-=amount;
                            amount=0;
                            break;
                        }
                        amount-=buy_backlog[k][1];
                        //删除该订单
                        buy_backlog[k][1]=0;

//buy_backlog.erase(buy_backlog.begin()+k,buy_backlog.begin()+k+1);
                    }
                }
                //当amount!=0
                if(amount!=0){
                    vector<int>temp;
                    temp.push_back(price);
                    temp.push_back(amount);
                    sell_backlog.push_back(temp);
                    sort(sell_backlog.begin(),sell_backlog.end());
                }
            }


```c++
    }

    //求未完成订单的总和
    int res=0;
    for(int i=0;i<buy_backlog.size();i++){
        res+=buy_backlog[i][1];
```
```

```
            res%=(1000000000 + 7);
        }
        for(int i=0;i<sell_backlog.size();i++){
            res+=sell_backlog[i][1];
            res%=(1000000000 + 7);
        }

        return res;
    }
    ```


    };
```

# #使用优先队列

```
class Solution {
public:
    typedef pair<int,int> pii;
    int getNumberOfBacklogOrders(vector<vector<int>>& orders) {
        priority_queue<pii> q1;//使用优先队列，从小到大的排列
        priority_queue<pii,vector<pii>,greater<pii>> q2;//使用优先队列，从
小到大的排列
        long long ans=0;
        for(auto &v : orders) {
            if(v[2] == 0) {
                while(!q2.empty() && q2.top().first <= v[0]) {
                    int p = min(q2.top().second, v[1]);
                    if(!p) break;
                    v[1] -= p;
                    auto tmp = q2.top(); q2.pop();
                    tmp.second -= p;
                    ans -= p;
                    if(tmp.second) q2.push(tmp);
                }
                if(v[1]) q1.push({v[0], v[1]}), ans += v[1];
            } else {
                while(!q1.empty() && q1.top().first >= v[0]) {
                    int p = min(q1.top().second, v[1]);
                    if(!p) break;
                    v[1] -= p;
                    auto tmp = q1.top(); q1.pop();
                    tmp.second -= p;
                    ans -= p;
                    if(tmp.second) q1.push(tmp);
                }
                if(v[1]) q2.push({v[0], v[1]}), ans += v[1];
            }
        }
```
```

```
        int mod=1e9+7;
        return ans%mod;
    }
};
```