贪心算法没有套路，想不到其他方法时候就想想贪心，用举反例的方法验证贪心，
如果举不出来则为正确

860.柠檬水找零

在柠檬水摊上，每一杯柠檬水的售价为 5 美元。

顾客排队购买你的产品，（按账单 bills 支付的顺序）一次购买一杯。

每位顾客只买一杯柠檬水，然后向你付 5 美元、10 美元或 20 美元。你必须给每个顾客正确找零，也就是说净交易是每位顾客向你支付 5 美元。

注意，一开始你手头没有任何零钱。

如果你能给每位顾客正确找零，返回 true ，否则返回 false 。

示例 1：
输入：[5,5,5,10,20]
输出：true
解释：
前 3 位顾客那里，我们按顺序收取 3 张 5 美元的钞票。
第 4 位顾客那里，我们收取一张 10 美元的钞票，并返还 5 美元。
第 5 位顾客那里，我们找还一张 10 美元的钞票和一张 5 美元的钞票。
由于所有客户都得到了正确的找零，所以我们输出 true。

小模拟：
由于本道题目需要记录手上的零钱面值则存在如何找零问题?
比如顾客向你支付了15元你可以用1张10元和1张5元或者3张5元找零
但根据贪心算法的思想:由于5元更具有通用性所以首先考虑用1张10元和1张5元找零
所有找零的情况如下：

- 情况一：账单是5，直接收下。
- 情况二：账单是10，消耗一个5，增加一个10
- 情况三：账单是20，优先消耗一个10和一个5，如果不够，再消耗三个5

本题目的局部最优：如果顾客向你支付20美元，优先用1张10元和1张5元去找零
全局最优：完成全部账单的找零。

```
class Solution {
public:
    bool lemonadeChange(vector<int>& bills) {
 int count=0;//手上的零钱数
 for(int i=0;i<bills.size();i++){
 if((bills[i]-5)<=count){
 count=count+5;
```

```
continue;
}
 return false;
 }
 return true;
    }
};
//以上的方法存在某个小瑕疵，要记录币种
class Solution {
public:
    bool lemonadeChange(vector<int>& bills) {
 int count[3]={0,0,0};//分别表示手上5,10,20元的零钱张数
 int sum=0;//手上的所有钱数
 for(int i=0;i<bills.size();i++){
 if((bills[i]-5)<=sum){//如果找零的钱如果小于sum一定不成立
 sum+=5;
 if(bills[i]==5){
 count[0]++;
 continue;
 }
 if(bills[i]==10&&count[0]>0){
 count[0]--;
 count[1]++;
 continue;
 }
 if(bills[i]==20&&count[1]>0&&count[0]>0){
 count[0]--;
 count[1]--;
 count[2]++;
 continue;
 }
 if(bills[i]==20&&count[0]>=3){
 count[0]-=3;
 count[2]++;
 continue;
 }
 }
 return false;
 }
 return true;
```

```
48         }
49     };
```

406.根据身高重建队列

题目链接：https://leetcode-cn.com/problems/queue-reconstruction-by-height/

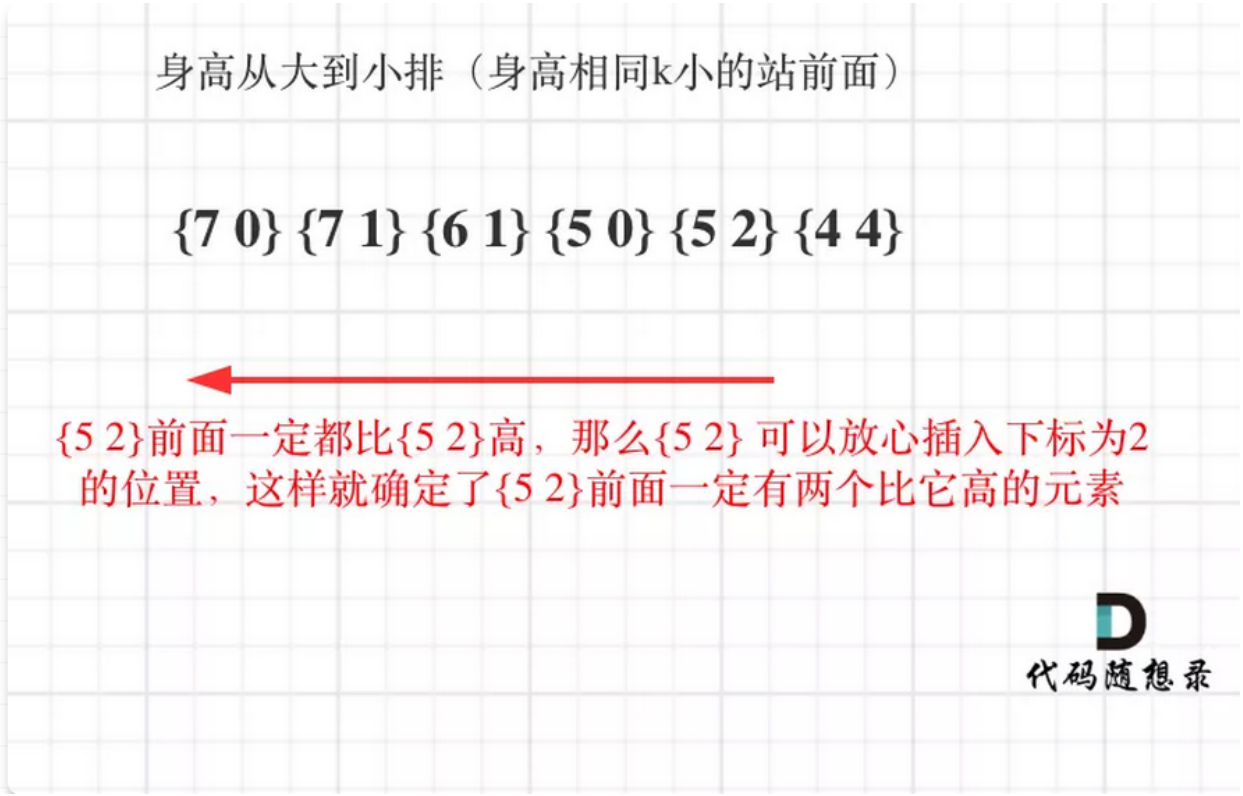
假设有打乱顺序的一群人站成一个队列，数组 people 表示队列中一些人的属性（不一定按顺序）。每个 people[i] = [hi, ki] 表示第 i 个人的身高为 hi ，前面 正好有 ki 个身高大于或等于 hi 的人。

请你重新构造并返回输入数组 people 所表示的队列。返回的队列应该格式化为数组 queue ，其中 queue[j] = [hj, kj] 是队列中第 j 个人的属性（queue[0] 是排在队列前面的人）。

本题有两个维度，h和k，看到这种题目一定要想如何确定一个维度，然后按照另外一个维度重新排列.
final的思想是:
先按照身高从大到小排序(当身高相等的时候,k小的在前面),那么只需要按照k为下标重新插入队列就可以了。**(手动模拟一下)**

为什么只要按照k为下标重新插入队列就可以了?

当队列身高从大到小排(身高相同k小的站前面),



为什么当身高相同时，k小的在前面?

为了后序插入节点不影响已经插入的节点.

```
1  class Solution {
```

```
public:
 static bool cmp(vector<int>& a, vector<int>&b){
 if(a[0]==b[0]) return a[1]<b[1];
 else return a[0]>b[0];
 }
    vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {
 sort(people.begin(),people.end(),cmp);
 vector<vector<int>> que;
  for(int i=0;i<people.size();i++){
  int k=people[i][1];
  que.insert(que.begin()+k,people[i]);
  }
  return que;
     }
};

```

上面那个时间复杂度为O(n^3)数据结构中对于插入和删除操作链表的时间复杂度比数组的时间复杂度低因此采用如下方法:下面的时间复杂度为O(n^2)

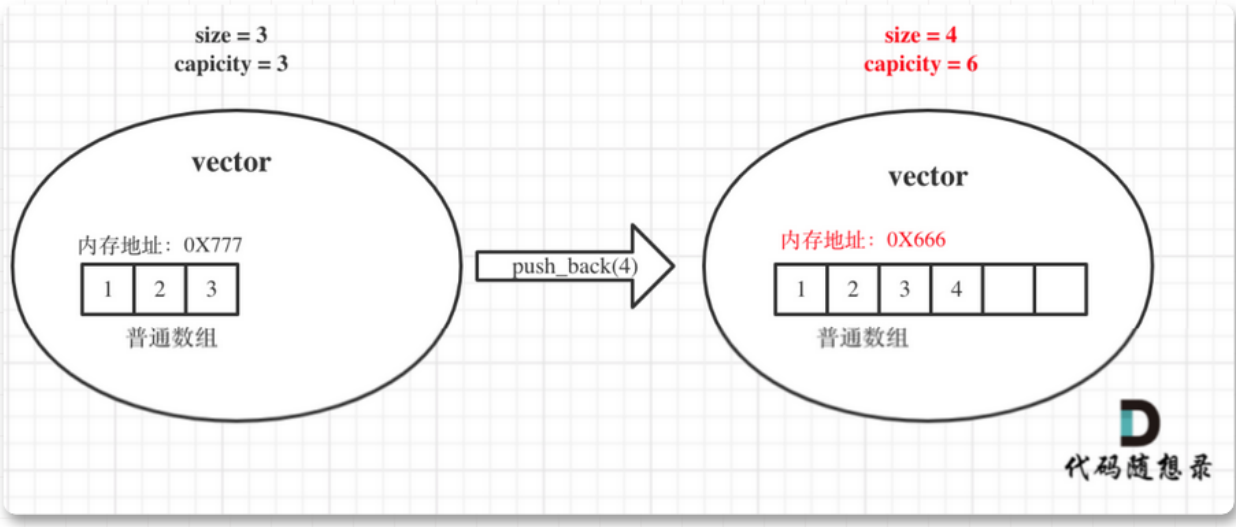
```
class Solution {
public:
 static bool cmp(const vector<int>& a,const vector<int>&b){
 if(a[0]==b[0]) return a[1]<b[1];
 else return a[0]>b[0];
 }
    vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {
 sort(people.begin(),people.end(),cmp);
 list<vector<int>> que;
  for(int i=0;i<people.size();i++){
  int k=people[i][1];
  que.insert(que.begin()+k,people[i]);
  }
  return vector<vector<int>>(que.begin(),que.end());
     }
};
```

对使用某一种语言容器的使用，特性的选择都会不同程度上影响效率

深入理解vector

1、普通数组一旦定义就不能改变数组大小而为什么vector可以任意的用push_back等函数对vector的大小进行操作?
首先vector的底层实现仍然是普通数组.
vector的大小有两个维度一个是size另一个是capicity
size是表征vector中实际存放的元素个数。

> **capicity表征底层数组中的实际开辟数组大小。**
> **首先size不一定等于capicity比如当insert数据的时候，如果已经大于capicity，capicity会成倍扩容，但对外暴露的size仅仅是+1**
> **2、vector底层实现是普通数组，怎么扩容的?**
> 就是重新申请一个二倍于原数组大小的数组，然后把数据拷贝过去，并释放原来的内存。



> 如图所示，原来普通数组size=3，capicity=3，当普通数组插入一个元素时，由于原来的数组不能存放该元素则扩容为原来的两倍即capicity=2*capicity=6，并将要插入的元素插入此时size=4.注意**扩容前的数组**和**扩容后的数组**在内存的地址不相等则**s是重新开辟新的内存**.

总结：

如果抛开语言谈算法，除非从来不用代码写算法纯分析，**「否则的话，语言功底不到位O(n)的算法可能写出O(n^2)的性能」**，哈哈。

452. 用最少数量的箭引爆气球

题目链接：https://leetcode-cn.com/problems/minimum-number-of-arrows-to-burst-balloons/

在二维空间中有许多球形的气球。对于每个气球，提供的输入是水平方向上，气球直径的开始和结束坐标。

由于它是水平的，所以纵坐标并不重要，因此只要知道开始和结束的横坐标就足够了。开始坐标总是小于结束坐标。

一支弓箭可以沿着 x 轴从不同点完全垂直地射出。在坐标 x 处射出一支箭，若有一个气球的直径的开始和结束坐标为 xstart，xend， 且满足 xstart ≤ x ≤ xend，则该气球会被引爆。

可以射出的弓箭的数量没有限制。弓箭一旦被射出之后，可以无限地前进。我们想找到使得所有气球全部被引爆，所需的弓箭的最小数量。

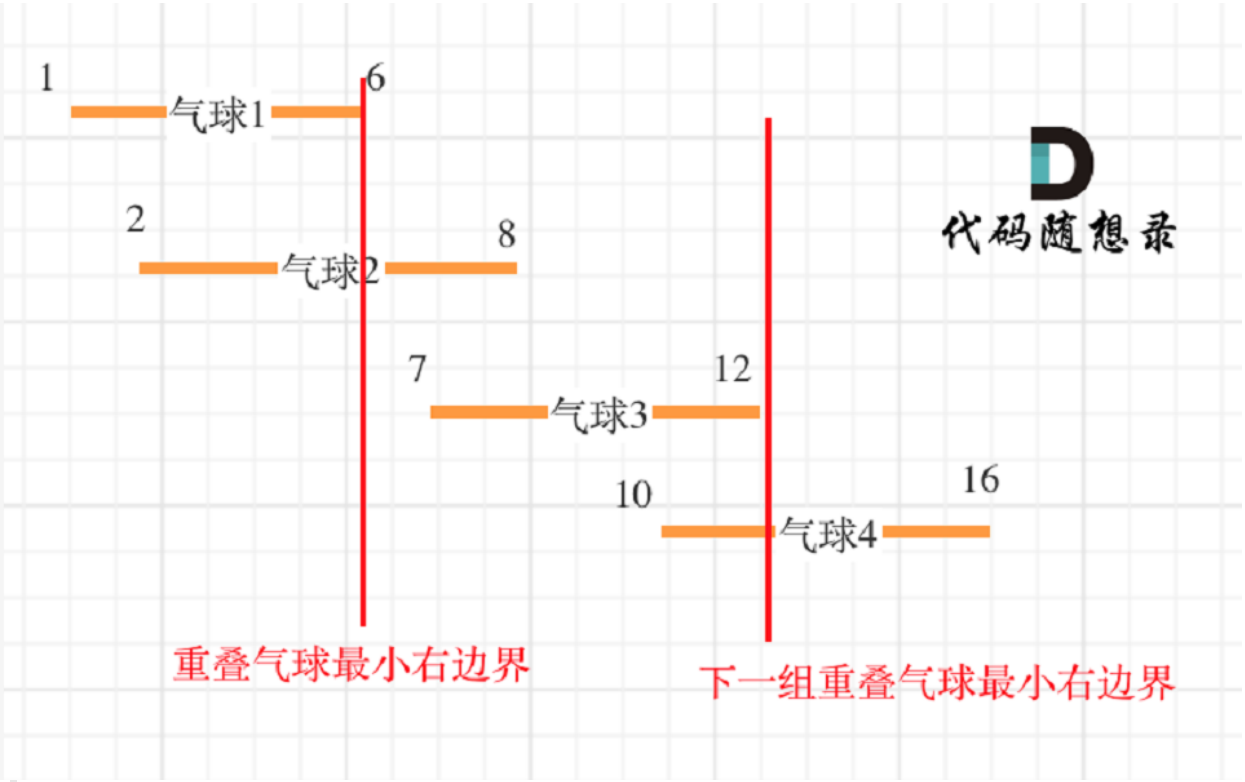
给你一个数组 points ，其中 points [i] = [xstart,xend] ，返回引爆所有气球所必须射出的最小弓箭数。

两个区间如何求交集区域

> [a1,a2],[b1,b2],交集区域为[c1,c2];
> if((a1>=b1&&a1<=b2)||(a2>=b1&&a2<=b2)){
> c1=max(a1,b1);
> c2=min(a2,b2);
> }
> 数据结构：
> vector<vector<int>> unionSet;//存放的现有的区域存在的交集，
> 假设unionSet中存在的交集有[c1,c2][d1,d2],现有一个区域[e1,e2],
> 1、若[e1,e2]与[c1,c2]以及[d1,d2]没有交集,那么就新建一个交集区域[e1,e2]插入unionSet
> 2、若[e1,e2]只与[c1,c2]或[d1,d2]有交集则更新相应的交集区域
> 3、若[e1,e2]与[c1,c2]和[d1,d2]均有交集，那么就选择更新后交集最大的区域更新

进一步优化

> 其实气球的重叠情况没必要用vector<vector<int>> unionSet;显性表示出来。
> 只要按照气球的起始位置排序即可。



只要当前要考虑的气球的左边界不在前面以及考虑的重叠气球的最小右边界那么就需要新的一支箭。

最小右边界的更新时间:在于当前考虑的气球可以与前面气球重叠时

```
class Solution {
public:
 //事先定义如何排序
 static bool cmp(const vector<int> a,const vector<int> b){
 return a[0]<b[0];
 }
    int findMinArrowShots(vector<vector<int>>& points) {
 if(points.size()==0) return 0;
 sort(points.begin(),points.end(),cmp);
  int result=1;//所需要的箭数
  for(int i=1;i<points.size();i++){
  if(points[i][0]>points[i-1][1]){
  //条件一定不能为points[i][0]>=points[i-1][1]
  result++;//此时当前考虑的气球没有与前面重叠
  }else{//存在重叠时
  points[i][0]=min(points[i-1][1],points[i][1]);
  }
  }
  return result;
     }
};
```

注意:

> 注意题目中说的是：满足 xstart ≤ x ≤ xend，则该气球会被引爆。那么说明两个气球挨在一起不重叠也可以一起射爆，
> 所以代码中 `if (points[i][0] > points[i - 1][1])` 不能是>=