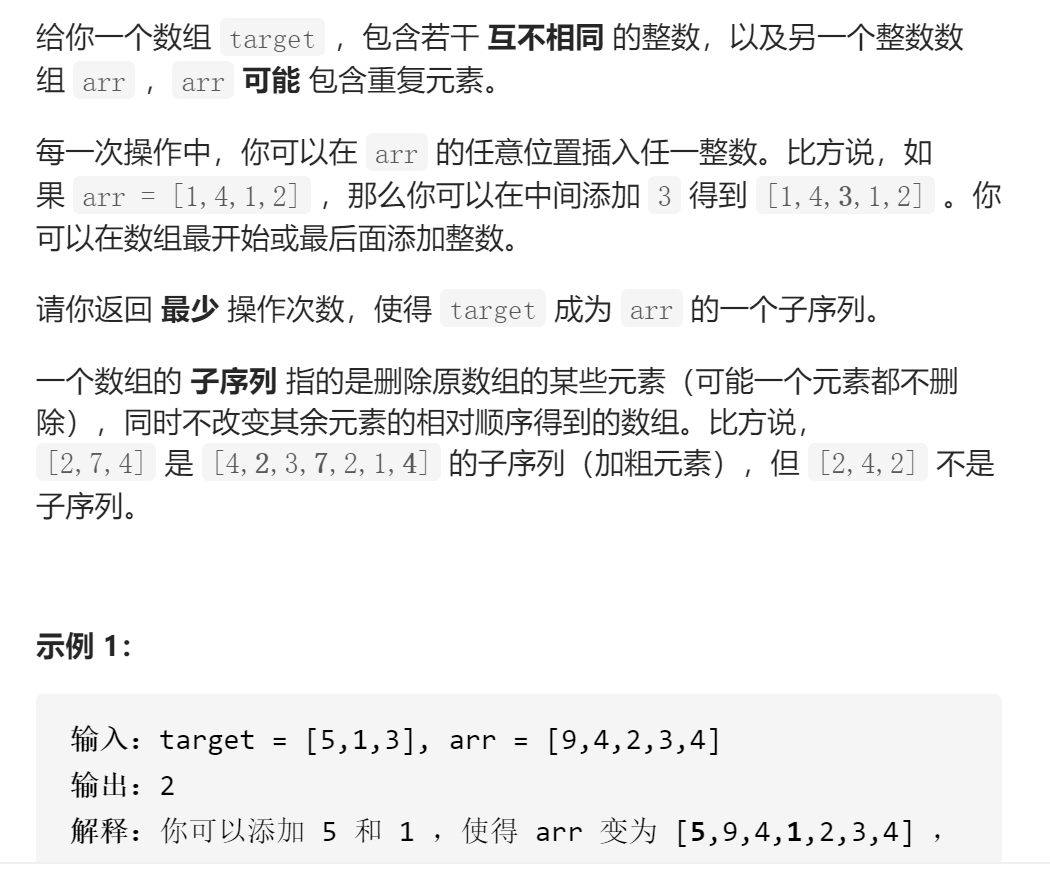
# 困难题：问题转换-贪心+二分查找||动态规划超出时间限制fig:

## 题目展示：



## 题目分析：

**1：将问题转换为求最长递增子序列问题**

**2：动态规划解决最长递增子序列问题可能会超时**

**3：此处用贪心+二分查找来降低时间复杂度**

## 代码展示：

class Solution {  
public:  
 int minOperations(vector<int>& target, vector<int>& arr) {  
 //建立一个哈希映射  
 //想简单了-此处贪心行不通  
 //转向动态规划  
 //动态规划超出时间限制，需要优化  
 //优化-贪心+二分查找  
 unordered\_map<int,int>mymap;  
 for(size\_t i=0;i<target.size();i++){  
 mymap[target[i]]=i;  
 }  
 vector<int>pos;  
 for(auto tp:arr){  
 if(mymap.find(tp)!=mymap.end()){  
 pos.push\_back(mymap[tp]);  
 }  
 }  
 if(pos.size()==0)return target.size();  
 /\*  
 vector<int>dp(pos.size(),1);  
 //问题转换维求dp数组中的最长递增子序列  
 //dp[i]的含义表示的是以下标i结尾的数字递增子序列的长度  
 int res=0;  
 for(int i =0;i<dp.size();i++){  
 int mx=0;  
 for(int j=0;j<i;j++){  
 if(pos[j]<pos[i]){  
 mx=max(mx,dp[j]);  
 }  
 }  
 dp[i]=mx+1;  
 res=max(dp[i],res);  
 }  
 return target.size()-res;  
 \*/  
 //dp[i]表示以长度i结尾的数的最小值  
 vector<int>dp(pos.size()+1,0);  
 int Len=1;  
 dp[1]=pos[0];//初始化  
 for(int i=0;i<pos.size();i++){  
 if(dp[Len]<pos[i]){  
 dp[++Len]=pos[i];  
 }else{  
 int left=1,right=Len, p=0;  
 while(left<=right){  
 int middle=(left+right)/2;  
 if(dp[middle]<pos[i]){  
 p=middle;  
 left=middle+1;  
 }else{  
 right=middle-1;  
 }  
 }  
 dp[p+1]=pos[i];  
 }  
 }  
 return target.size()-Len;  
 }  
};