2463. Minimum Total Distance Traveled

There are some robots and factories on the X-axis. You are given an integer array `robot` where `robot[i]` is the position of the `ith` robot. You are also given a 2D integer array `factory` where `factory[j] = [positionj, limitj]` indicates that `positionj` is the position of the `jth` factory and that the `jth` factory can repair at most `limitj` robots.

The positions of each robot are **unique**. The positions of each factory are also **unique**. Note that a robot can be **in the same position** as a factory initially.

All the robots are initially broken; they keep moving in one direction. The direction could be the negative or the positive direction of the X-axis. When a robot reaches a factory that did not reach its limit, the factory repairs the robot, and it stops moving.

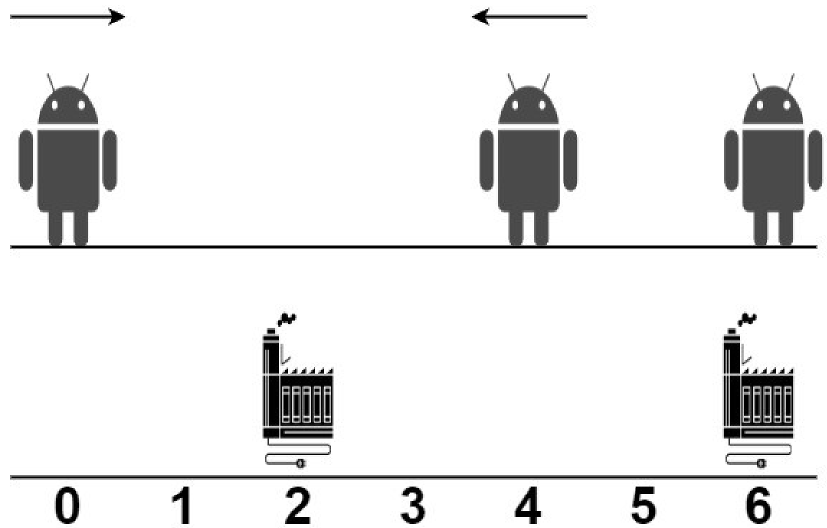
At any moment, you can set the initial direction of moving for **some** robot. Your target is to minimize the total distance traveled by all the robots.

Return *the minimum total distance traveled by all the robots*. The test cases are generated such that all the robots can be repaired.

Note that

- All robots move at the same speed.
- If two robots move in the same direction, they will never collide.
- If two robots move in opposite directions and they meet at some point, they do not collide. They cross each other.
- If a robot passes by a factory that reached its limits, it crosses it as if it does not exist.
- If the robot moved from a position `x` to a position `y`, the distance it moved is `|y - x|`.

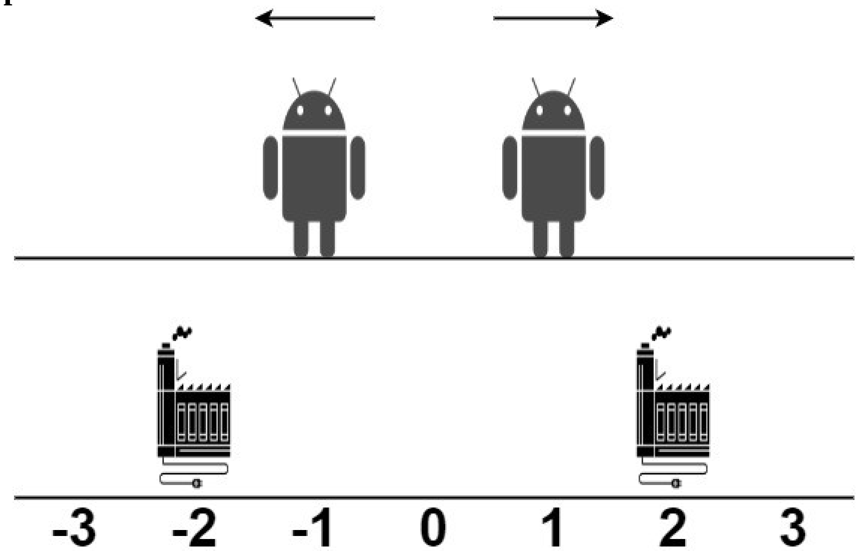
Example 1:



Input: robot = [0,4,6], factory = [[2,2],[6,2]]
Output: 4
Explanation: As shown in the figure:
- The first robot at position 0 moves in the positive direction. It will be repaired at the first factory.
- The second robot at position 4 moves in the negative direction. It will be repaired at the first factory.
- The third robot at position 6 will be repaired at the second factory. It does not need to move.

The limit of the first factory is 2, and it fixed 2 robots.
The limit of the second factory is 2, and it fixed 1 robot.
The total distance is |2 - 0| + |2 - 4| + |6 - 6| = 4. It can be shown that we cannot achieve a better total distance than 4.

Example 2:

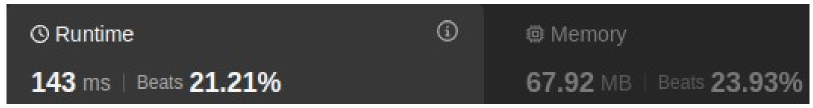



```
Input: robot = [1,-1], factory = [[-2,1],[2,1]]
Output: 2
Explanation: As shown in the figure:
- The first robot at position 1 moves in the positive direction. It will be
repaired at the second factory.
- The second robot at position -1 moves in the negative direction. It will be
repaired at the first factory.
The limit of the first factory is 1, and it fixed 1 robot.
The limit of the second factory is 1, and it fixed 1 robot.
The total distance is |2 - 1| + |(-2) - (-1)| = 2. It can be shown that we cannot
achieve a better total distance than 2.
```

Constraints:

- `1 <= robot.length, factory.length <= 100`
- `factory[j].length == 2`
- `-109 <= robot[i], positionj <= 109`
- `0 <= limitj <= robot.length`
- The input will be generated such that it is always possible to repair every robot.

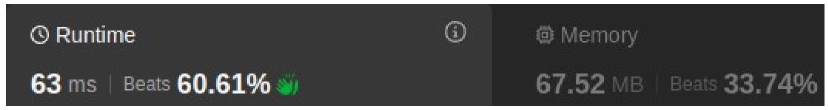
2463. Minimum Total Distance Traveled




```cpp
/*
   Include/Exclude: Recursion+Memoization
   Time complexity: O(nlogn+mlogm+nm)=O(mn)
   Space complexity: O(mn)+O(m+n)
*/
typedef long long ll;
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
class Solution {
    public:
        ll minimumTotalDistance(vi& robot,vvi& factory){
            // To ensure that the robot goes the the nearest factory
            std::sort(robot.begin(),robot.end());
            std::sort(factory.begin(),factory.end());

            /*
                having 1 factory of k limits <=> having k factories of limit 1
                (at the same position)
            */
            vi factories;
            for(auto& f: factory){
                for(int i=1;i<=f[1];++i) factories.push_back(f[0]);
            }

            int n=robot.size();
            int m=factories.size();

            std::vector<std::vector<ll>> memo(n,std::vector<ll>(m,-1));

            auto solve=[&](int robot_index,int factory_index,auto& self)->ll{
                // All robots are repaired
                //(Must be called first)
                if(robot_index==n) return 0;

                // If we still have broken robots and not no more factories
                if(factory_index==m) return LONG_MAX/2;

                 if(memo[robot_index][factory_index]!=-1) return memo[robot_index][factory_index];

                ll include=
                    abs(robot[robot_index]-factories[factory_index]) // current robot moves to current factory
                    +self(robot_index+1,factory_index+1,self); // Pass the the next robot and next factory
                                                                // This is why sorting is important

                // Current robot skip the current factory
                // and go the next nearest factoty
                // This is why sorting is important
                ll exclude=self(robot_index,factory_index+1,self);
                return  memo[robot_index][factory_index]=std::min(include,exclude);
            };

            return solve(0,0,solve);        }};
```

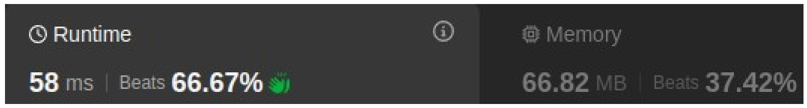
2463. Minimum Total Distance Traveled

```
/*
   Include/Exclude: bottom up
   Time complexity: O(nlogn+mlogm+nm)=O(mn)
   Space complexity: O(m+mn)
*/
```




```
typedef long long ll;
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
class Solution {
   public:
      ll minimumTotalDistance(vi& robot,vvi& factory){
         // To ensure that the robot goes the the nearest factory
         std::sort(robot.begin(),robot.end());
         std::sort(factory.begin(),factory.end());

         vi factories;
         for(auto& f: factory){
            for(int i=1;i<=f[1];++i) factories.push_back(f[0]);
         }

         int n=robot.size();
         int m=factories.size();

         std::vector<std::vector<ll>> dp(n+1,std::vector<ll>(m+1,0));

         for(int robot_index=1;robot_index<=n;++robot_index)
                  dp[robot_index][0]=LONG_MAX/2;

         for(int robot_index=1;robot_index<=n;++robot_index){
            for(int factory_index=1;factory_index<=m;++factory_index){

               ll include=abs(robot[robot_index-1]-factories[factory_index-1])
                        +dp[robot_index-1][factory_index-1];

               ll exclude=dp[robot_index][factory_index-1];

               dp[robot_index][factory_index]=std::min(include,exclude);
            }
         }

         return dp[n][m];
      }
};
```

2463. Minimum Total Distance Traveled




```
/*
   Include/Exclude: bottom up: space optimization
   Time complexity: O(nlogn+mlogm+nm)=O(mn)
   Space complexity: O(m)
*/
typedef long long ll;
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
class Solution {
   public:
      ll minimumTotalDistance(vi& robot,vvi& factory){
         std::sort(robot.begin(),robot.end());
         std::sort(factory.begin(),factory.end());

         vi factories;
         for(auto& f: factory){
            for(int i=1;i<=f[1];++i) factories.push_back(f[0]);
         }

         int n=robot.size();
         int m=factories.size();

         std::vector<ll> prev_row(m+1,0);

         for(int robot_index=1;robot_index<=n;++robot_index){
            std::vector<ll> cur_row(m+1,LONG_MAX/2);
            for(int factory_index=1;factory_index<=m;++factory_index){

               ll include=abs(robot[robot_index-1]-factories[factory_index-1])
                          +prev_row[factory_index-1];

               ll exclude=cur_row[factory_index-1];

               cur_row[factory_index]=std::min(include,exclude);
            }
            prev_row=cur_row;
         }

         return prev_row[m];
      }
};
```