1395. Count Number of Teams

There are `n` soldiers standing in a line. Each soldier is assigned a **unique** `rating` value.

You have to form a team of 3 soldiers amongst them under the following rules:

- Choose 3 soldiers with index (`i`, `j`, `k`) with rating (`rating[i]`, `rating[j]`, `rating[k]`).
- A team is valid if: (`rating[i] < rating[j] < rating[k]`) or (`rating[i] > rating[j] > rating[k]`) where (`0 <= i < j < k < n`).

Return the number of teams you can form given the conditions. (soldiers can be part of multiple teams).

Example 1:

```
Input: rating = [2,5,3,4,1]
Output: 3
Explanation: We can form three teams given the conditions. (2,3,4), (5,4,1),
(5,3,1).
```

Example 2:

```
Input: rating = [2,1,3]
Output: 0
Explanation: We can't form any team given the conditions.
```

Example 3:

```
Input: rating = [1,2,3,4]
Output: 4
```

Constraints:

- `n == rating.length`
- `3 <= n <= 1000`
- `1 <= rating[i] <= 105`
- All the integers in `rating` are **unique**.

1395. Count Number of Teams

```
/*
   Dynamic programming
   Time complexity: O(n^2+n+3n^2+n) = O(n^2)
   Extra space complexity= O(3n*2)=O(n)
*/

typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;
typedef std::vector<vii> vvii;

class Solution {
   public:
      int numTeams(std::vector<int>& rating) {
         int n=rating.size();

         // O(n^2)
         vvii dp(3,vii(n,{0,0}));

         // O(n)
         for(int i=0;i<n;++i) dp[0][i]={1,1};

         // O(3n^2)
         for(int i=1;i<3;++i){
            for(int j=i;j<n;++j){
               for(int k=i-1;k<j;++k){
                  if(rating[k]<rating[j]) dp[i][j].first+=dp[i-1][k].first;
                  else if(rating[k]>rating[j]) dp[i][j].second+=dp[i-1][k].second;
               }
            }
         }

         // O(n)
         int ans=0;
         for(int i=0;i<n;++i) ans+=dp[2][i].first+dp[2][i].second;

         return ans;
      }
};
```

```
/*
   Segment tree
   Time complexity: O(n^2+n+3n^2+n) = O(n^2)
   Extra space complexity= O(3n*2)=O(n)
*/
typedef std::pair<int,int> pii;
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::vector<pii> vii;

class SegmentTree{
   private:
      vi tree;
      int n; //Size of the original array
   public:
      SegmentTree(int n){
         this->n=n;
         int k=ceil(log2(n));
         int m=2*(1<<k)-1;
         tree.resize(m,0);
      }

      int query(int s, int e,int p,int l,int r){
         if(l>e || r<s) return 0;
         if(l<=s && r>=e) return tree[p];
         int mid=(s+e)>>1;
         return query(s,mid,2*p+1,l,r)+query(mid+1,e,2*p+2,l,r);
      }

      void update(int s,int e,int p, int i, int val){
         if(s==e){
            tree[p]=val;
            return;
         }
         int mid=(s+e)>>1;
         if(i>=s&&i<=mid) update(s,mid,2*p+1,i,val);
         else update(mid+1,e,2*p+2,i,val);
         tree[p]=tree[2*p+1]+tree[2*p+2];
      }

};
```

```
class Solution {
    public:
        int numTeams(std::vector<int>& rating) {
            int n=rating.size();

            vi arr=rating;

            std::sort(rating.begin(),rating.end());

            std::map<int,int> positions;
            for(int i=0;i<n;++i) positions[rating[i]]=i;


            SegmentTree seg=SegmentTree(n);

            int ans=0;

            for(int i=0;i<n;++i){
                int pos=positions[arr[i]];

                int nb_greater_in_left=seg.query(0,n-1,0,pos+1,n-1);

                int nb_smaller_in_left=i-nb_greater_in_left;

                int nb_smaller_in_right=pos-nb_smaller_in_left;

                int nb_greater_in_right=(n-pos-1)-nb_greater_in_left;

                ans+=nb_greater_in_left*nb_smaller_in_right + nb_smaller_in_left*nb_greater_in_right;


                seg.update(0,n-1,0,pos,1);
            }
            return ans;
        }
};
```