1671. Minimum Number of Removals to Make Mountain Array

You may recall that an array `arr` is a **mountain array** if and only if:

- `arr.length >= 3`
- There exists some index `i` (**0-indexed**) with `0 < i < arr.length - 1` such that:
    - `arr[0] < arr[1] < ... < arr[i - 1] < arr[i]`
    - `arr[i] > arr[i + 1] > ... > arr[arr.length - 1]`

Given an integer array `nums`, return *the* ***minimum*** *number of elements to remove to make* `nums` *a* ***mountain array***.

Example 1:

```
Input: nums = [1,3,1]
Output: 0
Explanation: The array itself is a mountain array so we do not need to remove any elements.
```

Example 2:

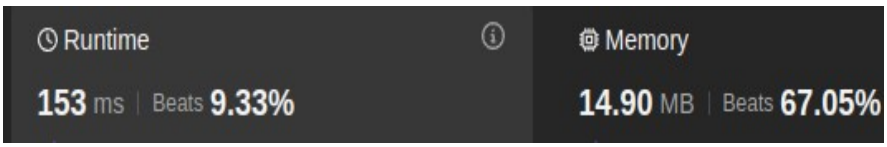
```
Input: nums = [2,1,1,5,6,2,3,1]
Output: 3
Explanation: One solution is to remove the elements at indices 0, 1, and 5, making the array nums = [1,5,6,3,1].
```

Constraints:

- `3 <= nums.length <= 1000`
- `1 <= nums[i] <= 109`
- It is guaranteed that you can make a mountain array out of `nums`.

1671. Minimum Number of Removals to Make Mountain Array

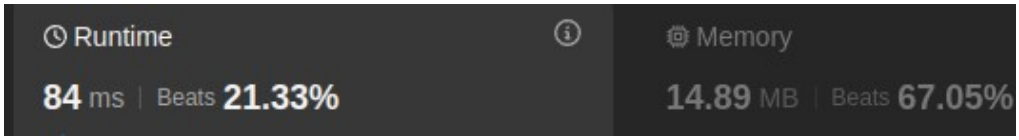
```
/*
   LIS,LDS using recursion+memoization
   Time complexity: O(n^2)
   Space complexity: O(n)
*/
```




```
class Solution {
   public:
      int minimumMountainRemovals(std::vector<int>& nums){
         int n=nums.size();
         // LIS using Recursion+DP (Memoization)
         // Determine LIS ending at index i
         std::vector<int> memo(n,-1);
         auto LIS=[&](int i,auto& self)->int{
            if(i==0) return 1;
            if(memo[i]!=-1) return memo[i];
            int mx=1;
            for(int j=0;j<i;++j){
               if(nums[j]<nums[i]) mx=std::max(mx,1+self(j,self));
            }
            return memo[i]=mx;
         };
         // lis stores LIS ending at index i
         std::vector<int> lis(n,1);
         for(int i=0;i<n;++i) {
            lis[i]=LIS(i,LIS);
         }
         // lds stores LDS starting at index i
         std::reverse(nums.begin(),nums.end());
         std::vector<int> lds(n,1);
         std::fill(memo.begin(),memo.end(),-1);
         for(int i=0;i<n;++i) {
            lds[i]=LIS(i,LIS);
         }
         std::reverse(lds.begin(),lds.end());
```

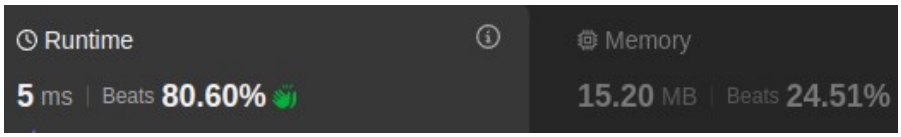
```
        // Minimize the number of removals
        int ans=INT_MAX;
        for(int i=0;i<n;++i){
            if(lis[i]==1 || lds[i]==1) continue;
            ans=std::min(ans,n-lds[i]-lis[i]+1);
        }

        return ans;
    }
};
```

1671. Minimum Number of Removals to Make Mountain Array




```
/*
   LIS,LDS using iterative
   Time complexity: O(n^2)
   Space complexity: O(n)
*/
class Solution {
   public:
      int minimumMountainRemovals(std::vector<int>& nums){
         // LIS using DP
         // Determine LIS ending at index i
         auto LIS=[&](std::vector<int>& A)->std::vector<int>{
            int n=A.size();
            std::vector<int> lis(n,1);
            for(int i=0;i<n;++i) {
               for(int j=0;j<i;++j){
                  if(A[j]<A[i]) lis[i]=std::max(lis[i],1+lis[j]);
               }
            }
            return lis;
         };
         int n=nums.size();
         // lis stores LIS ending at index i
         std::vector<int> lis=LIS(nums);
         // lds stores LDS starting at index i
         std::reverse(nums.begin(),nums.end());
         std::vector<int> lds=LIS(nums);
         std::reverse(lds.begin(),lds.end());
         // Minimize the number of removals
         int ans=INT_MAX;
         for(int i=0;i<n;++i){
            if(lis[i]==1 || lds[i]==1) continue;
            ans=std::min(ans,n-lds[i]-lis[i]+1);
         }
         return ans;
      }
};
```

1671. Minimum Number of Removals to Make Mountain Array




```
/*
   LIS,LDS Binary search
   Time complexity: O(n log n)
   Space complexity: O(n)
*/
class Solution {
   public:
      int minimumMountainRemovals(std::vector<int>& nums){
         // LIS using binary search
         // Determine LIS ending at index i
         auto LIS=[&](std::vector<int>& A)->std::vector<int>{
            int n=A.size();
            std::vector<int> lis(n,1);
            std::vector<int> tmp;
            tmp.push_back(A[0]);
            for(int i=1;i<n;++i){
               int j=std::lower_bound(tmp.begin(),tmp.end(),A[i])-tmp.begin();
               if(A[i]>tmp.back()) tmp.push_back(A[i]);
               else tmp[j]=A[i];

               lis[i]=j+1;
            }
            return lis;
         };
         int n=nums.size();
         // lis stores LIS ending at index i
         std::vector<int> lis=LIS(nums);
         // lds stores LDS starting at index i
         std::reverse(nums.begin(),nums.end());
         std::vector<int> lds=LIS(nums);
         std::reverse(lds.begin(),lds.end());
         // Minimize the number of removals
         int ans=INT_MAX;
         for(int i=0;i<n;++i){
            if(lis[i]==1 || lds[i]==1) continue;
            ans=std::min(ans,n-lds[i]-lis[i]+1);
         }
         return ans;}};
```