Day 46 / 100 :

Topic - Array, Binary search

1 Problem statement: Minimize the maximum difference pairs (medium)

You are given a 0-indexed integer array nums and an integer p. Find p pairs of indices of nums such that the maximum difference amongst all the pairs is minimized. Also, ensure no index appears more than once amongst the p pairs.

Note that for a pair of elements at the index i and j, the difference of this pair is |nums[i] - nums[j]|, where |x| represents the absolute value of x.

Return the minimum maximum difference among all p pairs. We define the maximum of an empty set to be zero.

Example 1:
Input: nums = [10,1,2,7,1,3], p = 2
Output: 1
Explanation: The first pair is formed from the indices 1 and 4, and the second pair is formed from the indices 2 and 5.
The maximum difference is max(|nums[1] - nums[4]|, |nums[2] - nums[5]|) = max(0, 1) = 1. Therefore, we return 1.

Example 2:
Input: nums = [4,2,1,2], p = 1
Output: 0
Explanation: Let the indices 1 and 3 form a pair. The difference of that pair is |2 - 2| = 0, which is the minimum we can attain.

Solutions :

Approach 1 - Hash table

Intuition:

The binary search solution takes ***O(nlog(n)+log(max(nums)n))*** times. If the DP is applied, alone the DP states has O(pn). The constraints say
1 <= nums.length <= 10^5
0 <= nums[i] <= 10^9
0 <= p <= (nums.length)/2
If you look at the recurrence relation

fn(i, x) = min(fn(i+1, x), max(abs(nums[i]-nums[i+1]), fn(i+2, x-1));
the TC could be O(n2)
it could lead to TLE or MLE. So binary search is the choosen one for this question.

Approach:

- Sort the array nums. Define the function numsPairs returning true when there exists at least p different pairs with nums[i+1]-nums[i]<=diff which has the elapsed time O(n)O(n)O(n).
- Apply binarch search to nums with repect to the function numsPairs which will take O(logmax(nums))O(\log \max(nums))O(logmax(nums)) times searches.
-
- So the main part of TC is O(logmax(nums))×O(n)=O(log(max(nums))n)O(\log \max(nums))\times O(n)=O(\log(\max(nums))n)O(logmax(nums))×O(n)=O(log(max(nums))n)

Time complexity:
O(nlog(n)+log(max(nums))n)O(n\log(n)+\log(\max(nums))n)O(nlog(n)+log(max(nums))n)

Space complexity:
O(1)

```
class Solution {
public:
    int n;
    bool numsPairs(vector<int>& nums, int diff, int p){
        int c=0;
        for(int i=0; i<n-1; i++)
            if (nums[i+1]-nums[i]<=diff) {
                c++;
                if (c>=p) return 1;
                i++;//can not be adjacent
            }
        return 0;
    }
    int minimizeMax(vector<int>& nums, int p) {
        if (p==0) return 0;//edge case
        n=nums.size();
        sort(nums.begin(), nums.end());
    //    for(int d: nums) cout<<d<<","; cout<<endl;

        int l=0, r=nums[n-1]-nums[0], m, ans;
```

```
        while(l<=r){
            m=l+(r-l)/2;
            if (numsPairs(nums, m, p)){
                ans=m;
                r=m-1;
            }
            else l=m+1;
        }

        return ans;
    }
};
```

1 Problem statement: Largest Prime Factor (medium)

Given a number N, the task is to find the largest prime factor of that number.

Example 1:
Input:
N = 5
Output:
5
Explanation:
5 has 1 prime factor i.e 5 only.

Example 2:
Input:
N = 24
Output:
3
Explanation:
24 has 2 prime factors 2 and 3 in which 3 is greater.

Solutions :

Approach 1 - Maths

Intuition:

- To find the largest prime factor of a number N, we can iterate through potential prime factors starting from 2 and keep dividing N by the current factor until it's no longer divisible. The largest prime factor of N will be the last prime factor we find before N becomes 1.

Code Explanation:

- The code initializes ans to 2, which is the smallest prime number.
- It enters a loop that continues as long as the square of ans is less than or equal to N. This optimization is used to limit the search for prime factors.
- Inside the loop, it checks if N is divisible by ans. If it is, it updates N by dividing it by ans.
- If N is not divisible by ans, it increments ans to move on to the next potential prime factor.
- This process continues until N becomes 1. At that point, ans will hold the largest prime factor of the original number.
-

```
long long int largestPrimeFactor(int N){
    // code here
    long long ans = 2;
    while((ans*ans)<=N){

        if(N%ans == 0){
            N = N/ans;
        }
        else{
            ans++;
        }
    }
    return N;
 }
```