Day 20 / 100 :

Topic - Hashing - DP, Maths

1 Problem statement: Longest arithmetic subsequence of given difference (Medium)

Given an integer array arr and an integer difference, return the length of the longest subsequence in arr which is an arithmetic sequence such that the difference between adjacent elements in the subsequence equals difference.
A subsequence is a sequence that can be derived from arr by deleting some or no elements without changing the order of the remaining elements.

Example 1:
Input: arr = [1,2,3,4], difference = 1
Output: 4
Explanation: The longest arithmetic subsequence is [1,2,3,4].

Example 2:
Input: arr = [1,3,5,7], difference = 1
Output: 1
Explanation: The longest arithmetic subsequence is any single element.

Example 3:
Input: arr = [1,5,7,8,5,3,4,2,1], difference = -2
Output: 4
Explanation: The longest arithmetic subsequence is [7,5,3,1].

Solutions :
Approach 1 - Iterative approach

Intuition:

Suppose we start a for loop and look through each element. Let's say we have stored the first element (= arr[0]) in memory. As we examine the remaining elements, the question we are interested in asking when examining arr[i] is whether we have already seen an element that equals to arr[i]-difference. This is because whenever arr[i]-arr[j]==difference (where i > j >= 0), this satisfies an arithmetic subsequence condition. One way to conveniently keep track of this

information would be to use a HashMap where the key represents a previously seen element arr[i] and the mapped value represents the corresponding maximum subarray length that can be formed using the key value as the tail.

Back to the for loop discussion... as we go through each elements, if the key with value arr[i]-difference is found, we must tell the map that arr[i] should be added to the map with a mapped value of whatever arr[i]-difference 's mapped value + 1 was. If arr[i]-difference was not found, we still need to add it to the map since this could be the beginning of a subsequence yet to be encountered. What about the mapped value? In this case, since the maximum subsequence length is just itself, the mapped value should be 1.

Once we create the aformentioned HashTable, we just need to find the maximum mapped value (= maximum arithmetic subsequence length). One way of doing this is by using iterators starting from the beginning of the map until the end.

(Note that the code can be optimized a bit more from 2 passes [O(2n)] to 1 pass [O(1n)] by keeping track of the longest it->second during the first pass)

```
class Solution {
public:
    int longestSubsequence(vector<int>& arr, int difference) {
          //deal with trivial solutions
        if(arr.size()<=1)
            return arr.size();
        unordered_map<int,int> m;
        m[arr[0]]++;

            //building a HashMap here
            //keys represent elements in arr[i] that have already been seen
            //mapped values represent the longest subarray the key can make
        for(int i=1;i<arr.size();i++){
            auto it=m.find(arr[i]-difference);
            if (it!=m.end())
                m[arr[i]]=(it->second+1);
            else
               m[arr[i]]=1;
        }

        auto it=m.begin();
        int ans=it->second;

        //traverse through the HashMap to find the longest subarray length
        while(it!=m.end()){
```

```
            if(it->second>ans)
                ans=it->second;
            it++;
        }
        return ans;
    }
};
```

Solutions :
Approach 2 - Recursive approach

Observation:
At each integer i in the array we can consider it as the end of AP(Arithmetic Progression/Sequence) and check the length of that AP which will be the (length of AP that ends with i-difference) + 1. We can thus use a hashmap to store this while traversing the array.

```
class Solution {
public:
    int longestSubsequence(vector<int>& arr, int difference)
    {
        unordered_map<int,int> lengths;
        int result=1;
        for(int &i:arr)
            result=max(result,lengths[i]=1+lengths[i-difference]); //Length of AP ending with 'i' with difference of 'difference' will be 1 + length of AP ending with 'i-difference'. result stores Max at each end
        return result;
    }
};
```

Complexity:
Space: O(n).
Time: O(n).

2 Problem statement: Reverse integer (Medium)

Given a signed 32-bit integer x, return x with its digits reversed. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0.
Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

Example 1:
Input: x = 123
Output: 321

Example 2:
Input: x = -123
Output: -321

Example 3:
Input: x = 120
Output: 21

Solutions :
Approach 1 - Maths

Approach:
The Idea is so simple, we need to take each character in the number and reverse the order. So we have 2 question how to take and reorder the character. So easy let calculate with me. I take 321 % 10 == 1 right? and then 32 % 10 = 2, 3 % 10 = 3. So the solution is we take the int x % 10 to take the number in position right most and then update x = x / 10 to calculate take the next character. we need to check the x < 0 or not, if it is, after we reverse the x we time x by -1;

```
class Solution {
public:
    int reverse(int x) {
        long long a = x * 1LL;
        long long result = 0;
    if(a < 0) {
        a = a / -1;
    }
    while(a != 0) {
        int number = a % 10;
        a = a / 10;
        result = result * 10 + number;
    }
```

```
    if(x < 0) {
        result *= -1LL;
    }

    if(result > INT_MAX || result < INT_MIN) {
        return 0;
    }
    return (int) result;
    }
};
```