DP
Lesson 7

1. Longest Increasing Subsequence
2. minimum-number-of-deletions-to-make-a-sorted-sequence
3. increasing-subsequence
4. Maximum sum increasing subsequence
5. printing-maximum-sum-increasing-subsequence
6. longest-common-increasing-subsequence
7. count-increasing-subsequences

Longest Increasing Subsequence

Given an array of integers, find the length of the longest (strictly) increasing subsequence from the given array.

Example 1:

Input:

N = 16

A[]={0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15}

Output: 6

Explanation:Longest increasing subsequence 0 2 6 9 13 15, which has length 6

Example 2:

Input:

N = 6

A[] = {5,8,3,7,9,1}

Output: 3

Explanation:Longest increasing subsequence 5 7 9, with length 3

Expected Time Complexity : O(N*log(N))

Expected Auxiliary Space: O(N)

Constraints:

$1 \leq N \leq 10^5$

$0 \leq A[i] \leq 10^6$

```
int longestSubsequence(int n, int a[])
  {
     int t[n];
     for(int i=0;i<n;i++)
        t[i]=1; //every element is LIS
     int ans=0;
     for(int i=1;i<n;i++)
     {
        for(int j=0;j<i;j++)
        {
           if(a[i]>a[j])
           {
              t[i]=max(t[i],1+t[j]);
           }
           ans=max(ans,t[i]);
        }
     }
     return ans;
}
```

```
int longestSubsequence(int n, int a[])
  {
     vector<int>v;
     v.push_back(a[0]);
     for(int i=1;i<n;i++)
     {
        if(a[i]>v[v.size()-1])
           v.push_back(a[i]);
        else
        {
           int idx=upper_bound(v.begin(), v.end(),a[i])-v.begin();
           v[idx]=a[i];
        }
     }
     return v.size();
  }
```

Minimum number of deletions to make a sorted sequence

Given an array arr of size N, the task is to remove or delete the minimum number of elements from the array so that when the remaining elements are placed in the same sequence order form an increasing sorted sequence.

Example 1:

Input: N = 5, arr[] = {5, 6, 1, 7, 4}
Output: 2
Explanation: Removing 1 and 4leaves the remaining sequence order as 5 6 7 which is a sorted sequence.

Example 2:
Input: N = 3, arr[] = {1, 1, 1}
Output: 2
Explanation: Remove 2 1's

Expected Time Complexity: O(N2)
Expected Auxiliary Space: O(N)

Constraints:
$1 \leq N \leq 103$

```
int minDeletions(int a[], int n)
        {
            if(n==1)
              return 0;
            int t[n];
      for(int i=0;i<n;i++)
         t[i]=1;
      int ans=0;
      for(int i=1;i<n;i++)
      {
         for(int j=0;j<i;j++)
         {
            if(a[i]>a[j])
            {
               t[i]=max(t[i],1+t[j]);
            }
            ans=max(ans,t[i]);
         }
      }

      return n-ans;
            // Your code goes here
         }
```

Maximum sum increasing subsequence

Given an array arr of N positive integers, the task is to find the maximum sum increasing subsequence of the given array.

Example 1:

Input: N = 5, arr[] = {1, 101, 2, 3, 100}

Output: 106

Explanation:The maximum sum of increasing sequence is obtained from{1, 2, 3, 100}

Example 2:

Input: N = 3, arr[] = {1, 2, 3}

Output: 6

Explanation:The maximum sum of increasing sequence is obtained from{1, 2, 3}

Expected Time Complexity: O(N2)

Expected Auxiliary Space: O(N)

Constraints:

$1 \leq N \leq 103$

$1 \leq arr[i] \leq 105$

```
1.      int maxSumIS(int arr[], int n)
2.              {int i, j, max1 = 0;
3.          int dp[n];
4.          for ( i = 0; i < n; i++ )
5.             dp[i] = arr[i];
6.          for ( i = 1; i < n; i++ )
7.             for ( j = 0; j < i; j++)
8.                if(arr[i]>arr[j])
9.                   dp[i]=max(dp[i],dp[j]+arr[i]);
10.         for ( i = 0; i < n; i++ )
11.            if ( max1 < dp[i] )
12.               max1 = dp[i];
13.         return max1;}
```

Count Increasing Subsequences

Given an array of digits (values lie in range from 0 to 9). The task is to count all the sub sequences possible in an array such that in each subsequence every digit is greater than its previous digits in the subsequence.

Example 1:

Input :

a[] = {1, 2, 3, 4}

Output: 15

Explanation :There are total increasing subsequences{1}, {2}, {3}, {4}, {1,2}, {1,3}, {1,4},{2,3}, {2,4}, {3,4}, {1,2,3}, {1,2,4},{1,3,4}, {2,3,4}, {1,2,3,4}

Example 2:

Input :

a[] = {4, 3, 6, 5}

Output:8

Explanation :Sub-sequences are {4}, {3}, {6}, {5},{4,6}, {4,5}, {3,6}, {3,5}

Example 3:

Input :

a[] = {3, 2, 4, 5, 4}

Output : 14

Explanation :Sub-sequences are {3}, {2}, {4}, {3,4},{2,4}, {5}, {3,5}, {2,5}, {4,5}, {3,2,5}{3,4,5}, {4}, {3,4}, {2,4}

Expected Time Complexity: O(N)

Expected Auxiliary Space: O(max(a[i])) = O(10)

Constraints:

$1 \leq N \leq 500$

$1 \leq a[i] \leq 9$

We can use this fact by using an array count[] such that count[d] stores current count digits smaller than d.

For example, arr[] = {3, 2, 4, 5, 4}

We create a count array and initialize it as 0.

count[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

Note that here value is used as index to store counts

count[3] += 1 = 1 // i = 0, arr[0] = 3

count[2] += 1 = 1 // i = 1, arr[0] = 2

Let us compute count for arr[2] which is 4

count[4] += 1 + count[3] + count[2] += 1 + 1 + 1 = 3

Let us compute count for arr[3] which is 5

count[5] += 1 + count[3] + count[2] + count[4]

+= 1 + 1 + 1 + 3

= 6

Let us compute count for arr[4] which is 4

count[4] += 1 + count[0] + count[1]

+= 1 + 1 + 1

+= 3

= 3 + 3

= 6

Note that count[] = {0, 0, 1, 1, 6, 6, 0, 0, 0, 0}

Result = count[0] + count[1] + ... + count[9]

= 1 + 1 + 6 + 6 {count[2] = 1, count[3] = 1

count[4] = 6, count[5] = 6}

= 14.

```
1.      unsigned long long int countSub(int arr[], int n)
2.        {
3.           vector<unsigned long long > v(10,0);
4.           unsigned long long int ans=0;
5.           for(int i=0;i<n;i++)
6.           {
7.              for(int j=0;j<arr[i];j++)
8.              {
9.                 v[arr[i]]+=v[j];
10.             }
11.             v[arr[i]]++;
12.
13.          }
14.          for(int i=0;i<10;i++)
15.          ans+=v[i];
16.          return ans;
17.          // Your code goes here
18.       }
```

Longest Common Increasing Subsequence

Given two arrays, arr1[] and arr2[] of sizes M and N respectively, find the length of the longest common increasing subsequence(LCIS).

Example 1:

Input:

M = 4

Arr1[] = {3, 4, 9, 1}

N = 7

Arr2[] = {5, 3, 8, 9, 10, 2, 1}

Output: 2

Explanation: The longest increasing subsequence that is common is {3, 9} and its length is 2.

Example 2:

Input:

M = 4

Arr1[] = {1, 1, 4, 3}

N = 4

Arr2[] = {1, 1, 3, 4}

Output: 2

Explanation: There are two common subsequences {1, 4} and {1, 3}both of length 2.

Expected Time Complexity: O(N2)

Expected Auxiliary Space: O(N)

Constraints:

1 <= M, N <= 103

1 <= Arr1[i], Arr2[i] <= 103

```
1.      int LCIS(int arr1[], int m, int arr2[], int n)
2.        {
3.           // code here
4.           int table[n];
5.           memset(table,0,sizeof(table));
6.           for(int i=0; i<m; i++)
7.           {
8.              int cur=0;
9.              for(int j=0; j<n; j++)
10.             {
11.                if(arr1[i]==arr2[j])
12.                   table[j]=max(table[j],cur+1);
13.                else if(arr1[i]>arr2[j])
14.                   cur=max(cur,table[j]);
15.             }
16.          }
17.          return *max_element(table,table+n);
18.          // code here
19.       }
```