Dynamic Programming | Set 15 (Longest Bitonic Subsequence)

Given an array arr[0 … n-1] containing n positive integers, a subsequence of arr[] is called Bitonic if it is first increasing, then decreasing. Write a function that takes an array as argument and returns the length of the longest bitonic subsequence.
A sequence, sorted in increasing order is considered Bitonic with the decreasing part as empty. Similarly, decreasing order sequence is considered Bitonic with the increasing part as empty.

Examples:

```
Input arr[] = {1, 11, 2, 10, 4, 5, 2, 1};
Output: 6 (A Longest Bitonic Subsequence of length 6 is 1, 2, 10, 4, 2, 1)

Input arr[] = {12, 11, 40, 5, 3, 1}
Output: 5 (A Longest Bitonic Subsequence of length 5 is 12, 11, 5, 3, 1)

Input arr[] = {80, 60, 30, 40, 20, 10}
Output: 5 (A Longest Bitonic Subsequence of length 5 is 80, 60, 30, 20, 10)
```

Source: Microsoft Interview Question

Solution

This problem is a variation of standard Longest Increasing Subsequence (LIS) problem. Let the input array be arr[] of length n. We need to construct two arrays lis[] and lds[] using Dynamic Programming solution of LIS problem. lis[i] stores the length of the Longest Increasing subsequence ending with arr[i]. lds[i] stores the length of the longest Decreasing subsequence starting from arr[i]. Finally, we need to return the max value of lis[i] + lds[i] – 1 where i is from 0 to n-1.

Following is C++ implementation of the above Dynamic Programming solution.

```
/* Dynamic Programming implementation of longest bitonic subsequence problem */
#include<stdio.h>
#include<stdlib.h>

/* lbs() returns the length of the Longest Bitonic Subsequence in
    arr[] of size n. The function mainly creates two temporary arrays
    lis[] and lds[] and returns the maximum lis[i] + lds[i] - 1.

    lis[i] ==> Longest Increasing subsequence ending with arr[i]
    lds[i] ==> Longest decreasing subsequence starting with arr[i]
```

```
*/
int lbs( int arr[], int n )
{
   int i, j;

   /* Allocate memory for LIS[] and initialize LIS values as 1 for
      all indexes */
   int *lis = new int[n];
   for ( i = 0; i < n; i++ )
      lis[i] = 1;

   /* Compute LIS values from left to right */
   for ( i = 1; i < n; i++ )
      for ( j = 0; j < i; j++ )
         if ( arr[i] > arr[j] && lis[i] < lis[j] + 1)
            lis[i] = lis[j] + 1;

   /* Allocate memory for lds and initialize LDS values for
      all indexes */
   int *lds = new int [n];
   for ( i = 0; i < n; i++ )
      lds[i] = 1;

   /* Compute LDS values from right to left */
   for ( i = n-2; i >= 0; i-- )
      for ( j = n-1; j > i; j-- )
         if ( arr[i] > arr[j] && lds[i] < lds[j] + 1)
            lds[i] = lds[j] + 1;


   /* Return the maximum value of lis[i] + lds[i] - 1*/
   int max = lis[0] + lds[0] - 1;
   for (i = 1; i < n; i++)
     if (lis[i] + lds[i] - 1 > max)
         max = lis[i] + lds[i] - 1;
   return max;
}

/* Driver program to test above function */
int main()
{
  int arr[] = {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15};
  int n = sizeof(arr)/sizeof(arr[0]);
  printf("Length of LBS is %d\n", lbs( arr, n ) );

  getchar();
  return 0;
}
```

Run on IDE

Output:

```
Length of LBS is 7
```

Time Complexity: O(n^2)

Auxiliary Space: O(n)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above