|  |  |
| --- | --- |
| **Longest Bitonic Subseq In C++** | |
| #include <iostream>  #include <vector>  using namespace std;  int LongestBitonicSubseq(int arr[], int n) {      vector<int> lis(n, 1); // lis[i] will store the length of LIS ending at index i      vector<int> lds(n, 1); // lds[i] will store the length of LDS starting at index i      // Computing LIS      for (int i = 1; i < n; i++) {          for (int j = 0; j < i; j++) {              if (arr[j] <= arr[i]) {                  lis[i] = max(lis[i], lis[j] + 1);              }          }      }      // Computing LDS      for (int i = n - 2; i >= 0; i--) {          for (int j = n - 1; j > i; j--) {              if (arr[j] <= arr[i]) {                  lds[i] = max(lds[i], lds[j] + 1);              }          }      }      int omax = 0; // To store the overall maximum length of LBS  // Finding the length of the Longest Bitonic Subsequence      for (int i = 0; i < n; i++) {          omax = max(omax, lis[i] + lds[i] - 1);      }  return omax;  }  int main() {      int arr[] = {10, 22, 9, 33, 21, 50, 41, 60, 80, 3};      int n = sizeof(arr) / sizeof(arr[0]);      cout << LongestBitonicSubseq(arr, n) << endl;      return 0;  } | ****Step-by-Step Dry Run********Step 1: Compute**** lis[] ****(Longest Increasing Subsequence)**** We iterate from **left to right**, storing the longest increasing sequence **ending at each index**.   | **i** | **arr[i]** | **LIS Calculation (lis[i] = max(lis[i], lis[j] + 1))** | **lis[i]** | | --- | --- | --- | --- | | 0 | 10 | lis[0] = 1 (base case) | 1 | | 1 | 22 | 10 < 22 → lis[1] = lis[0] + 1 = 2 | 2 | | 2 | 9 | No valid j | 1 | | 3 | 33 | 10 < 33 → lis[3] = lis[0] + 1 = 2 | 2 | |  |  | 22 < 33 → lis[3] = lis[1] + 1 = 3 | 3 | | 4 | 21 | 10 < 21 → lis[4] = lis[0] + 1 = 2 | 2 | | 5 | 50 | 10 < 50 → lis[5] = lis[0] + 1 = 2 | 2 | |  |  | 22 < 50 → lis[5] = lis[1] + 1 = 3 | 3 | |  |  | 33 < 50 → lis[5] = lis[3] + 1 = 4 | 4 | | 6 | 41 | 10 < 41 → lis[6] = lis[0] + 1 = 2 | 2 | |  |  | 22 < 41 → lis[6] = lis[1] + 1 = 3 | 3 | |  |  | 33 < 41 → lis[6] = lis[3] + 1 = 4 | 4 | | 7 | 60 | 10 < 60 → lis[7] = lis[0] + 1 = 2 | 2 | |  |  | 22 < 60 → lis[7] = lis[1] + 1 = 3 | 3 | |  |  | 33 < 60 → lis[7] = lis[3] + 1 = 4 | 4 | |  |  | 50 < 60 → lis[7] = lis[5] + 1 = 5 | 5 | | 8 | 80 | 10 < 80 → lis[8] = lis[0] + 1 = 2 | 2 | |  |  | 22 < 80 → lis[8] = lis[1] + 1 = 3 | 3 | |  |  | 33 < 80 → lis[8] = lis[3] + 1 = 4 | 4 | |  |  | 50 < 80 → lis[8] = lis[5] + 1 = 5 | 5 | |  |  | 60 < 80 → lis[8] = lis[7] + 1 = 6 | 6 | | 9 | 3 | No valid j | 1 |  ****Final**** lis[] ****Array**** lis = [1, 2, 1, 3, 2, 4, 4, 5, 6, 1] ****Step 2: Compute**** lds[] ****(Longest Decreasing Subsequence)**** We iterate from **right to left**, storing the longest decreasing sequence **starting from each index**.   | **i** | **arr[i]** | **LDS Calculation (lds[i] = max(lds[i], lds[j] + 1))** | **lds[i]** | | --- | --- | --- | --- | | 9 | 3 | lds[9] = 1 (base case) | 1 | | 8 | 80 | lds[8] = 1 | 1 | | 7 | 60 | lds[7] = max(lds[7], lds[8] + 1) = 2 | 2 | | 6 | 41 | lds[6] = max(lds[6], lds[7] + 1) = 3 | 3 | | 5 | 50 | lds[5] = max(lds[5], lds[6] + 1) = 4 | 4 | | 4 | 21 | lds[4] = 2 | 2 | | 3 | 33 | lds[3] = max(lds[3], lds[4] + 1) = 3 | 3 | | 2 | 9 | lds[2] = max(lds[2], lds[4] + 1) = 2 | 2 | | 1 | 22 | lds[1] = max(lds[1], lds[2] + 1) = 3 | 3 | | 0 | 10 | lds[0] = max(lds[0], lds[2] + 1) = 2 | 2 |  ****Final**** lds[] ****Array**** lds = [2, 3, 2, 3, 2, 4, 3, 2, 1, 1] ****Step 3: Compute**** omax ****(Overall Maximum LBS)**** Using:  omax = max(lis[i] + lds[i] - 1)   | **i** | **lis[i]** | **lds[i]** | **lis[i] + lds[i] - 1** | | --- | --- | --- | --- | | 0 | 1 | 2 | **2** | | 1 | 2 | 3 | **4** | | 2 | 1 | 2 | **2** | | 3 | 3 | 3 | **5** | | 4 | 2 | 2 | **3** | | 5 | 4 | 4 | **7** | | 6 | 4 | 3 | **6** | | 7 | 5 | 2 | **6** | | 8 | 6 | 1 | **6** | | 9 | 1 | 1 | **1** |   The **maximum** value in this list is 7. |
| Output:-7 | |