**Problem**

**You are playing a board game: It has two sets of n pieces, each of which have a different color (so there are n colors, and two pieces of each color). You scatter one set of n pieces on one half of the board, and the other set of n pieces on the other half of board. You can add a bridge between two pieces of the same color. Given the board’s setup, what’s the maximum number of bridges you can add to the board without two bridges crossing?**

**Solution**

Let’s suppose N pieces of different colors, R, G, B, Y, W, O scattered (sorted) on the first half of the board. In the other half, the same set of pieces is scattered unevenly (unsorted). The coordinates of pieces on the first half and the second half are given as the input. The objective is to connect the pieces in the first half (H1) with that in the second half (H2) with the help of bridges, and the bridges must not overlap with each other.

To Find: Max. No. of bridges not overlapping

**INPUT**

N = 6

A[]= {R, G, B, Y, W, O}

B[]= { O, R, G, B, W, Y}

**OUTPUT**

maxBridges: integer max number of non-overlapping bridges

**DIAGRAM**

**H1**



**H2**

Here, the max no of bridges that can be formed without crossing is 4. (non-overlapping)

The no. of bridges crossing is 2. (overlapping)

1. **Dynamic Programming**

For creating bridges, we must consider all potential bridge combinations and determine which one is most suitable to deliver the most maximal results. To prevent these calculations from happening again (repetition), we will store the results in a memoization array.

**Algorithm-**

1. Store the coordinates of the colored pieces in a pair.
2. Sort the pairs according to the H2 coordinates in increasing order.
3. The H1 and H2 coordinates should both be in either increasing order or decreasing. This will avoid crossing of the bridges.
4. As we sorted the coordinates of the colored pieces on the H2 side, we need to find the coordinates of colored pieces on the H1 side that are increasing or decreasing to find the maximum number of non-intersecting bridges.
5. Now we will find the Longest Increasing Subsequence (LIS) of the coordinates of the H1 side to find the maximum number of bridges.
6. For this problem, a value greater than or equal to the previous value can also be considered part of the increasing subsequence.

**Test Case 1**

N = 4,



Co-ordinates on Side H1= A[] = {6, 4, 2, 1}



Co-ordinates on Side H2= B[] ={2, 3, 6, 5}



Step 1-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| H1 | 6 | 4 | 2 | 1 |
| H2 | 2 | 3 | 6 | 5 |



Step 2- Sorting the H2 coordinates in ascending order

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| H1 | 6 | 4 | **1** | **2** |
| H2 | 2 | 3 | **5** | **6** |

Step 3- Find LIS Longest Increasing Subsequence of H1

LIS is given by {1, 2}. Length of LIS is 2. Hence the maximum number of bridges is also 2.

-------------------------------------------------------------------------------------------------------------

Text

Description automatically generated

Text

Description automatically generated

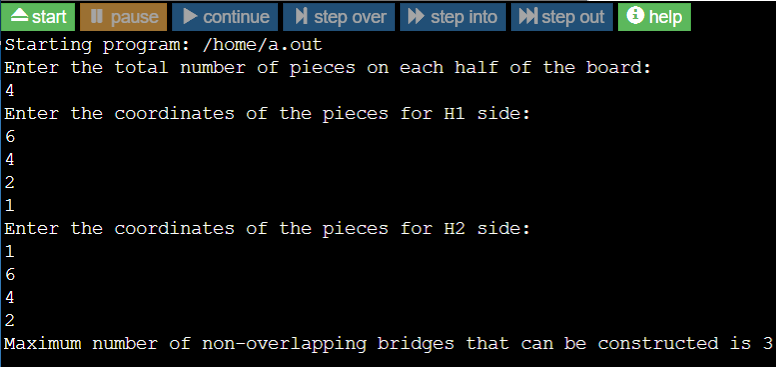
Text

Description automatically generated

Text

Description automatically generated

------------------------------------------------------------------------------------------------------------------**OUTPUT**



-----------------------------------------------------------------------------------------------------------------

**I/P**

Enter the total number of cities on each bank: 4

Enter the coordinates of pieces in H1: 6 4 2 1

Enter the coordinates of pieces in H2: 1 6 4 2

**O/P**

Max no. of bridges that can be constructed is 3.

**Time Complexity-**

For computing LIS, we make use of one nested loop, as a result, the time complexity becomes

***O( N 2)***

**Space Complexity**

We need an extra size N for every index of LIS, the Space Complexity is **O(N)**

**B. Dynamic Programming with Memoization**

**Algorithm**

1. Store the coordinates of the colored pieces in a pair.
2. Sort the pairs according to H1 coordinates in increasing order.
3. As H1 is increasing, we will find the Longest Increasing Subsequence for H2 coordinates.
4. Length of this LIS is solution for the problem.
5. Create a DP array and initialize it with a huge value, say INT\_MAXVAL.
6. Traversing H2 coordinates individually, then find the index of the just next greater element present in the DP array.
7. Update it with the value of this H2 coordinate. Length of the LIS, which is ending at that index, will be increased by 1.
8. Note that the smallest element will have the greatest possibility of contributing to the LIS.
9. Track the max LIS and this max value will be answer.
10. End

**Test Case 2**

Diagram

Description automatically generated



Table

Description automatically generated with low confidence

**Step 2- Sorting items in H1 Coordinates**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| H1 | 1 | 2 | 4 | 6 |
| H2 | 5 | 6 | 3 | 2 |

**Step 3- LIS of H2 coordinates DP[5]**



|  |
| --- |
| INT\_MAX INT\_MAX INT\_MAX INT\_MAX INT\_MAX |

**Step 4- Store LIS in the variable res.**

i = 0 val =5Next greater element in the DP array = INT\_MAXIndex of next greater element = 0 **DP[5]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **5** | **INT\_MAX** | **INT\_MAX** | **INT\_MAX** | **INT\_MAX** |



**res = max (0, 0+1)**

**= max (0,1)**

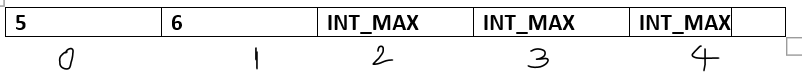
**= 1**

**i = 1**

**value = 6**

Next greater element in the DP array =**INT\_MAX**

Index of next greater element =**1 DP [5]**



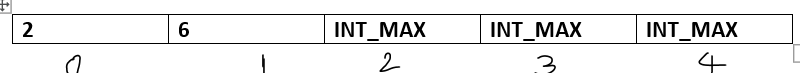
**res = max(1, 1+1)**

**= max (1,2)**

**= 2   i = 2** value **= 3**

Next greater element in the DP array=**5**

Index of next greater element =**0** **DP [5]**



Program:-

Text

Description automatically generated



**OUTPUT**

**Graphical user interface, text

Description automatically generated**

**Time Complexity-**

We have a single loop from index 0 to N-1. For every iteration, we use a lower bound, which costs O(log N) time. Thus, the time complexity for optimized dynamic programming is **O(N \* log N).**

**Space Complexity-**

We have an additional array for storing non-overlapping bridges, hence, the space complexity is**O(N)**

------------------------------------------------------------------------------------------------------------------------------------------

**C) BRUTE FORCE**

In brute force algorithm, we take every possible subset of the n possible bridges, and check whether the bridges are overlapping or not. We can then keep track of the maximum number of non-overlapping bridges.

Algorithm-

1. Choose whether to pick each bridge after carefully considering all possible subsequences. Using this method, we obtain 2n different cases.
2. It is not at all a useful idea to verify each subsequence for non-overlappingness in order to obtain the valid subsequence.
3. This will require time of O(n^2) for each subsequence and will result in an overall time complexity of O((2^n)(n^2).
4. Time Complexity- O(n^2)
5. Space Complexity- O(n)