Minimum number of swaps required for arranging pairs adjacent to each other

There are n-pairs and therefore 2n people. everyone has one unique number ranging from 1 to 2n. All these 2n persons are arranged in random fashion in an Array of size 2n. We are also given who is partner of whom. Find the minimum number of swaps required to arrange these pairs such that all pairs become adjacent to each other.

Example:

```
Input:
n = 3
pairs[] = {1->3, 2->6, 4->5}  // 1 is partner of 3 and so on
arr[] = {3, 5, 6, 4, 1, 2}

Output: 2
We can get {3, 1, 5, 4, 6, 2} by swapping 5 & 6, and 6 & 1
```

Source: Google Interview Question

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to start from first and second elements and recur for remaining elements. Below are detailed steps/

```
1) If first and second elements are pair, then simply recur
   for remaining n-1 pairs and return the value returned by
   recursive call.

2) If first and second are NOT pair, then there are two ways to
   arrange. So try both of them return the minimum of two.
   a) Swap second with pair of first and recur for n-1 elements.
      Let the value returned by recursive call be 'a'.
   b) Revert the changes made by previous step.
   c) Swap first with pair of second and recur for n-1 elements.
      Let the value returned by recursive call be 'b'.
   d) Revert the changes made by previous step before returning
      control to parent call.
   e) Return 1 + min(a, b)
```

Below is C++ implementation of above algorithm.

```
// C++ program to find minimum number of swaps required so that
// all pairs become adjacent.
#include<bits/stdc++.h>
using namespace std;

// This function updates indexes of elements 'a' and 'b'
void updateindex(int index[], int a, int ai, int b, int bi)
{
    index[a] = ai;
    index[b] = bi;
}

// This function returns minimum number of swaps required to arrange
// all elements of arr[i..n] become aranged
int minSwapsUtil(int arr[], int pairs[], int index[], int i, int n)
{
    // If all pairs procesed so no swapping needed return 0
    if (i > n) return 0;

    // If current pair is valid so DO NOT DISTURB this pair
    // and move ahead.
    if (pairs[arr[i]] == arr[i+1])
         return minSwapsUtil(arr, pairs, index, i+2, n);

    // If we reach here, then arr[i] and arr[i+1] don't form a pair
    // Swap pair of arr[i] with arr[i+1] and recursively compute
    // minimum swap required if this move is made.
    int one = arr[i+1];
    int indextwo = i+1;
    int indexone = index[pairs[arr[i]]];
    int two = arr[index[pairs[arr[i]]]];
    swap(arr[i+1], arr[indexone]);
    updateindex(index, one, indexone, two, indextwo);
    int a = minSwapsUtil(arr, pairs, index, i+2, n);

    // Backtrack to previous configuration. Also restore the
    // previous indices, of one and two
    swap(arr[i+1], arr[indexone]);
    updateindex(index, one, indextwo, two, indexone);
    one = arr[i], indexone = index[pairs[arr[i+1]]];

    // Now swap arr[i] with pair of arr[i+1] and recursively
    // compute minimum swaps required for the subproblem
    // after this move
    two = arr[index[pairs[arr[i+1]]]], indextwo = i;
    swap(arr[i], arr[indexone]);
    updateindex(index, one, indexone, two, indextwo);
    int b = minSwapsUtil(arr, pairs, index, i+2, n);

    // Backtrack to previous configuration.  Also restore
    // the previous indices, of one and two
    swap(arr[i], arr[indexone]);
    updateindex(index, one, indextwo, two, indexone);

    // Return minimum of two cases
    return 1 + min(a, b);
}

// Returns minimum swaps required
int minSwaps(int n, int pairs[], int arr[])
{
    int index[2*n + 1]; // To store indices of array elements

    // Store index of each element in array index
    for (int i = 1; i <= 2*n; i++)
        index[arr[i]] = i;

    // Call the recursive function
    return minSwapsUtil(arr, pairs, index, 1, 2*n);
}

// Driver program
int main()
{
    // For simplicity, it is assumed that arr[0] is
    // not used.  The elements from index 1 to n are
    // only valid elements
    int arr[] = {0, 3, 5, 6, 4, 1, 2};

    // if (a, b) is pair than we have assigned elements
    // in array such that pairs[a] = b and pairs[b] = a
    int pairs[] = {0, 3, 6, 1, 5, 4, 2};
    int m = sizeof(arr)/sizeof(arr[0]);

    int n = m/2;  // Number of pairs n is half of total elements

    // If there are n elements in array, then
    // there are n pairs
    cout << "Min swaps required is " << minSwaps(n, pairs, arr);
    return 0;
}
```

Output:

```
Min swaps required is 2
```