Two Pointers Technique

Two pointers is really an easy and effective technique which is typically used for searching pairs in a sorted array.

Given a sorted array A (sorted in ascending order), having N integers, find if there exists any pair of elements (A[i], A[j]) such that their sum is equal to X.

Let’s see the **naive solution**.

|  |
| --- |
| // Naive solution to find if there is a  // pair in A[0..N-1] with given sum.    bool isPairSum(A[], N, X)  {      for (i = 0; i < N; i++) {          for (j = 0; j < N; j++) {              if (A[i] + A[j] == X)                  return true; // pair exists                if (A[i] + A[j] > X)                  break; // as the array is sorted          }      }        // No pair found with given sum.      return false;  } |

The time complexity of this solution is **O(n2)**.

Now let’s see how the two-pointer technique works. We take two pointers, one representing the first element and other representing the last element of the array, and then we add the values kept at both the pointers. If their sum is smaller than X then we shift the left pointer to right or if their sum is greater than X then we shift the right pointer to left, in order to get closer to the sum. We keep moving the pointers until we get the sum as X.

|  |
| --- |
| // Two pointer technique based solution to find  // if there is a pair in A[0..N-1] with a given sum.  bool isPairSum(A[], N, X)  {      // represents first pointer      int i = 0;        // represents second pointer      int j = N - 1;        while (i < j) {            // If we find a pair          if (A[i] + A[j] == X)              return true;            // If sum of elements at current          // pointers is less, we move towards          // higher values by doing i++          else if (A[i] + A[j] < X)              i++;            // If sum of elements at current          // pointers is more, we move towards          // lower values by doing j--          else              j--;      }      return false;  } |

Illustration :

The above solution works in **O(n)**

**How does this work?**  
The algorithm basically uses the fact that the input array is sorted. We start the sum of extreme values (smallest and largest) and conditionally move both pointers. We move left pointer i when the sum of A[i] and A[j] is less than X. We do not miss any pair because the sum is already smaller than X. Same logic applies for right pointer j.