You have to find the value of element at position C[x] and C[y]. After that you can simply find the answer. To get the value at given position, you can do binary search using a condition which is “find the count of elements less than given middle value”. Each subtask uses different techniques to find the count of elements less than middle value. Finally after finding the values, the answer = max (x , y) – min(x , y) - D(C[x]) - D(C[y]),where D(C[x]) = count of duplicates of C[x], in between min(x , y) and max(x , y).

First Subtask - Since max sum of A[i] + B[j] <= 20000 and n <= 4000, we can precompute the count of possible A[i] + B[j] in cnt array. Now we have cnt array, you can form prefix sum array from that. So, you can query inside binary search and find the count of elements less than middle value. Let’s say Count = count of elements less than middle value. While doing binary search, you can find the end position of mid value's duplicates, it is same as Count. So now you have both end points of duplicates. You don't have to consider duplicates of C[min(x , y)], and you can ignore the range and have to find only count of duplicates of C[max(x ,y)] by using the same trick we did before inside binary search. Time complexity: O(q * log(n) + n * (max(A[i]) + max(B[j])))

Second Subtask - Here we can use Two pointers to do it in O(n + m) time complexity. Let’s say Count = count of elements less than middle value. While doing binary search, you can find the end position of middle value's duplicates, it is same as Count. So now you have both the end points of duplicates. You don't have to consider duplicates of C[min(x , y)], and you can ignore the range and have to find only count of duplicates of C[max(x , y)] by using the same trick we did before inside binary search.

Time Complexity: O(q * (n + m) * log(max(A[i] + B[j]))). However, some solutions may not receive full marks due to the high value constant in time complexity.