005994409

Timothy Liu

Dis 2A

1. When I was implementing my countXPairs function, my program kept counting duplicate comparisons for two different elements. Initially, I used a for loop to go through each element of the array, then compared that element to every other element in the array using a nested for loop. However, the later elements would also compare itself to the previous elements resulting in the function returning double the pairs that it’s supposed to. To fix this, I had the second nested for loop only go from the current element index to the elements after. This prevented the current element from being compared with elements that have already been checked.

Another obstacle I encountered was when implementing the moveToEnd function. Initially, I moved the target item to the end first, then moved every item down, but this resulted in the last two items being duplicates. To fix this, I had to store the target item first, move all the elements down 1 index, then move the target item to the end.

1. Test Data

Arrays

|  |  |  |  |
| --- | --- | --- | --- |
| string test1a[5] = {“2”, ”3”, ”4”, ”5”, “2”} | string test1b[4] = {“2”, ”2”, “2”, “1”} | string test1c[4] = {“a”, “aa”, “aaa”, “aaaa”} | string test1d[5] = {"orange","apple", "banana", "carrot", "pear"}; |
| string test2a[5] = {"2", "3", "3", "4", "1"}; | string test2b[4] = {"1", "2", "3", "4"}; | string test2c[3] = {"a", "A", " a"}; | string test2d[3] = {"", "", ""}; |
| string test3a[4] = {“1”, “2”, “3”, “0”} | string test3b[3] = {“0”, “0”, “0”} | string test3c[3] = {“-2”, “0”, “0”} |  |
| string test 4a[6] = {“2”, “2”, “3”, “0”, ”3”, “4”} | string test4b[3] = {“0”, “0”, “0”} |  |  |
| string test5a[5] = {"3.3.3.3.3.3.3" , ".","-" , "?", "1" } | string test5b[4] = {“3”, “4”, “hello”, “test” } | string test5c[10] = {".",",","!", ";", "'", "-", "/", ":","?","\"" }; |  |
| string test6a[6] = {“2”, “2”, “3”, “0”, ”3”, “4”} | string test6b[5] = {“0”, “0”, “0”, “0”, “0”} |  |  |
| string test7a[5] = {"a","b","c","d","e"}; | string test7b[6] = {"a","b","c","d","e","f"}; |  |  |

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Handles Correctly or Not** | **Comments** |
| locateMaximum(test1a, 5) | Handles correctly | Largest item is 5 |
| locateMaximum(test1a, 2) | Handles correctly | Array size of 2 |
| locateMaximum(test 1a, 1) | Handles correctly | Array size of 1 |
| locateMaximum(test 1a, 0) | Handles correctly | Array size of 0 |
| locateMaximum(test 1a, -1) | Handles correctly | Negative array size |
| locateMaximum(test1b, 4) | Handles correctly | Multiple duplicate largest items |
| locateMaximum(test1c, 4) | Handles correctly | Different string lengths |
| locateMaximum(test1d, 1) | Handles correctly | Pear is largest item |
| locateMaximum(test5b, 4) | Handles correctly | Numbers and letters in string |
| hasNoDuplicates(test2a, 5) | Handles correctly | Array has duplicates |
| hasNoDuplicates(test2b, 4) | Handles correctly | Array has no duplicates |
| hasNoDuplicates(test2a, 2) | Handles correctly | Array size of 2 |
| hasNoDuplicates(test2a, 1) | Handles correctly | Array size of 1 |
| hasNoDuplicates(test2a, 0) | Handles correctly | Array size of 0 |
| hasNoDuplicates(test2a, -2) | Handles correctly | Negative array size |
| hasNoDuplicates(test 2c, 3) | Handles correctly | Lowercase and uppercase strings |
| hasNoDuplicates(test 2d, 3) | Handles correctly | Empty strings |
| countXPairs(test3a, 4, 3) | Handles correctly | Only one pair adds up to 1 |
| countXPairs(test3a, 1, 1) | Handles correctly | Array of size 1 |
| countXPairs(test3a, 0, 1) | Handles correctly | Array of size 0 |
| countXPairs(test3a, -9, 1) | Handles correctly | Negative array size |
| countXPairs(test3a, 4, 1) | Handles correctly | Single element is = 1, nothing else added together =1 |
| countXPairs(test3b, 3, 0) | Handles correctly | Duplicate elements |
| countXPairs(test3c, 3, -2) | Handles correctly | Pairs add up to -2 |
| findSecondToLastOccurrence(test4a, 2, “3”) | Handles correctly | Array size of 2 |
| findSecondToLastOccurrence(test4a, 1, “2”) | Handles correctly | Array of size 1 |
| findSecondToLastOccurrence(test4a, 0, “2”) | Handles correctly | Array of size 0 |
| findSecondToLastOccurrence(test4a, -2, “2”) | Handles correctly | Negative array size |
| findSecondToLastOccurrence(test4a, 6, “2”) | Handles correctly | Two occurrences |
| findSecondToLastOccurrence(test4b, 3, “0”) | Handles correctly | Three occurrences |
| countPunctuation(test5a, 5) | Handles correctly | Mix of different punctuations |
| countPunctuation(test5a, 1) | Handles correctly | Array size of 1 |
| countPunctuation(test5a, 0) | Handles correctly | Array size of 0 |
| countPunctuation(test5a, -2) | Handles correctly | Negative array size |
| countPunctuation(test5b, 4) | Handles correctly | No punctuations |
| countPunctuation(test5c, 4) | Handles correctly | One of every punctuation mark |
| flipAround(test6a, 3) | Handles correctly | Array size of 3 |
| flipAround(test6a, 2) | Handles correctly | Array size of 2 |
| flipAround(test6a, 1) | Handles correctly | Array of size 1 |
| flipAround(test6a, 0) | Handles correctly | Array of size 0 |
| flipAround(test6a, -2) | Handles correctly | Negative array size |
| flipAround(test6a, 6) | Handles correctly | Array size of 6 |
| flipAround(test6b, 5) | Handles correctly | Array with all duplicate elements |
| moveToEnd(test7a, 5, -10) | Handles correctly | Negative target index |
| moveToEnd(test7a, 2, 0) | Handles correctly | Array size of 2 |
| moveToEnd(test7a, 1, 0) | Handles correctly | Array of size 1 |
| moveToEnd(test7a, 0, 0) | Handles correctly | Array of size 0 |
| moveToEnd(test7a, -20, 0) | Handles correctly | Negative array size |
| moveToEnd(test7a, 5, 4) | Handles correctly | Moving the last element |
| moveToEnd(test7a, 5, 3) | Handles correctly | Odd array size |
| moveToEnd(test6a, 6, 3) | Handles correctly | Even array size |