a.

One notable obstacle I encountered while developing my program was caused by my failing to realize that some functions modify the array that I used for testing. In my list of test cases, tests after previous tests that modified the array returned unexpected results because the array was changed. It took me a while to realize why my simplest functions were not returning the correct values. I fixed this problem by creating new arrays for tests after tests that changed the test arrays. Two functions that I had some difficulty creating where the subsequence and split functions. It took me a while to realize the problem in my algorithm for the subsequence function, and I had to redo the function differently to make it work correctly. The split function had me stuck until I realized that I could simply alphabetize the array and then find the first element not less than the splitter to find the return value.

b.

All test cases should run correctly without stopping execution of the program.

**string a[7] = {"bernie", "marco", "hillary", "jeb", "carly", "ben", “donald"};**

-initialize test array

**assert(appendToAll(a, -2, "!!!") == -1 && a[0] == "bernie" && a[6] == “donald");**

**-**incorrect input

**assert(appendToAll(a, 7, "!!!") == 7 && a[0] == "bernie!!!" && a[6] == “donald!!!");**

-the string is appended to the entire array

**string b[5] = {"bernie", "marco", "hillary", "marco", "carly"};**

-initialize test array

**assert(appendToAll(b, 1, "!!!") == 1 && b[0] == "bernie!!!" && b[4] == “carly");**

-the string is only appended to the first element

**assert(lookup(b, 5, "hillary") == 2);**

-looks up the target string in the entire array

**assert(lookup(b, 4, "marco") == 1);**

-looks up the target string in part of the array

**assert(lookup(b, 0, "marco") == -1);**

-looks up the target string in an array of no elements

**assert(lookup(b, -1, "marco") == -1);**

-incorrect input

**assert(lookup(b, 5, "obama") == -1);**

-can’t find the target string in the array

**assert(positionOfMax(b, 5) == 1);**

-finds the position of max in the entire array

**assert(positionOfMax(b, 1) == 0);**

-finds the position of max in an array of one element

**assert(positionOfMax(b, 0) == -1);**

-can’t find the position of max in an empty array

**assert(positionOfMax(a, 7) == 1);**

-finds the position of max in the entire array

**assert(positionOfMax(a, -1) == -1);**

-incorrect input

**string c[5] = {"bernie", "marco", "hillary", "marco", “carly"};**

-initializes test array

**assert(rotateLeft(c, 5, 1) == 1 && c[0] == "bernie" && c[1] == "hillary" && c[4] == "marco");**

-rotates the entire array left

**string d[4] = {"bernie", "marco", "hillary", "carly"};**

-initializes test array

**assert(rotateLeft(d, 3, 0) == 0 && d[0] == "marco" && d[1] == "hillary" && d[2] == “bernie");**

-rotates part of the array left

**assert(rotateLeft(d, -1, -1) == -1);**

-incorrect input

**assert(countRuns(a, 7) == 7);**

-counts the runs in the entire array

**assert(countRuns(a, 3) == 3);**

-counts the runs in part of the array

**assert(countRuns(a, 0) == 0);**

-counts the runs in an array of no elements

**string e[8] = {"bernie", "bernie", "marco", "hillary", "hillary", "hillary", "ben", "donald"};**

-initializes test array

**assert(countRuns(e, 8) == 5);**

-counts runs in the entire array

**assert(countRuns(e, -1) == -1);**

-incorrect input

**assert(flip(e, 8) == 8 && e[0] == "donald" && e[1] == "ben" && e[7] == “bernie");**

-flips the entire array

**assert(flip(e, 8) == 8 && e[0] == "bernie" && e[1] == "bernie" && e[7] == “donald");**

-flips the entire array back to previous

**assert(flip(e, 3) == 3 && e[0] == "marco" && e[2] == "bernie" && e[7] == “donald");**

-flips part of the array

**assert(flip(e, 0) == 0);**

-can’t flip an array of no elements

**assert(flip(e, -1) == -1);**

-incorrect input

**string f[5] = {"bernie", "marco", "chris", "hillary", “donald”};**

-initializes test array

**string g[4] = {"bernie", "marco", "donald", "chris"};**

-initializes test array

**assert(differ(f, 5, g, 4) == 2);**

-checks the entire arrays for differing elements

**assert(differ(f, 4, g, 2) == 2);**

-checks part of the arrays for differing elements

**assert(differ(f, 2, g, 4) == 2);**

-checks part of the arrays for differing elements

**assert(differ(f, 0, g, 2) == 0);**

-checks an empty array for differing elements

**assert(differ(f, -1, g, 2) == -1);**

-incorrect input

**string h[6] = {"marco", "jeb", "mike", "carly", "ted", “donald"};**

-initializes test array

**string i[3] = {"mike", "carly", “ted”};**

-initializes test array

**string j[3] = {"mike", "jeb", “ted"};**

-initializes test array

**string k[0];**

-initializes empty test array

**assert(subsequence(h, 6, i, 3) == 2);**

-checks the entire arrays for subsequences

**assert(subsequence(h, 5, i, 3) == 2);**

-checks parts of the arrays for subsequences

**assert(subsequence(h, 6, i, 2) == 2);**

-checks parts of the arrays for subsequences

**assert(subsequence(h, 6, j, 3) == -1);**

-a subsequence is not found

**assert(subsequence(h, -1, i, 3) == -1);**

-incorrect input

**assert(subsequence(i, 3, k, 0) == 0);**

-a sequence of no elements is a subsequence of any sequence

**assert(subsequence(h, 0, i, 2) == -1);**

-an empty array has no subsequences

**string l[5] = {"donald", "marco", "hillary", "carly", “bernie"};**

-initializes test array

**string m[5] = {"carly", "jeb", "mike", "marco", "ted"};**

-initializes test array

**assert(lookupAny(l, 5, m, 5) == 1);**

-looks up elements in the entire arrays

**assert(lookupAny(l, 4, m, 2) == 3);**

-looks up elements in part of the arrays

**assert(lookupAny(l, 0, m, 2) == -1);**

-elements cannot be found in an empty array

**assert(lookupAny(l, 3, m, 3) == -1);**

-elements are not found

**assert(lookupAny(l, -1, m, 2) == -1);**

-incorrect input

**string o[7] = {"marco", "jeb", "mike", "ted", "hillary", "carly", “bernie"};**

-initializes test array

**string p[7] = {"marco", "jeb", "mike", "ted", "hillary", "carly", “bernie"};**

-initializes test array

**string q[7] = {"marco", "jeb", "mike", "ted", "hillary", "carly", "bernie"};**

-initializes test array

**assert(split(o, 7, "hillary") == 2);**

-splits entire array with element present in array

**assert(split(p, 7, "rand") == 6);**

-splits entire array with element not present in array

**assert(split(q, 4, "mark") == 2);**

-splits part of the array with element not present in array

**assert(split(q, -1, "hillary") == -1);**

-incorrect input

**assert(split(q, 0, "jeb") == 0);**

-tries to split empty array

**cout << “All tests succeeded!” << endl;**

-prints if all tests execute correctly