1. One big obstacle I faced was in the makemerger function and attempting to iterate through the positions of the 3 arrays at different intervals. I was able to combat this by creating new position variables that tracked the position in both arrays 1 and 2 and incrementing them at different times to vary the iteration and then using only 1 for loop with the i value representing the current position for the larger array encompassing both.

string h[7] = { "neil", "sonia", "john", "amy", "", "elena", "john" };

assert(enumerate(h, 7, "john") == 2); ***- checks if it counts the correct number of strings in the array***

assert(enumerate(h, 7, "") == 1); ***- checks if it counts the correct number of strings in the array with empty string***

assert(findMatch(h, 7, "john") == 2); ***- checks if the correct position of the first occurrence of the string is found***

assert(findMatch(h, 2, "john") == -1); ***- checks if it correctly returns 0 if no target is found within the size indicated***

int bg;

int en;

assert(findRun(h, 7, "amy", bg, en) && bg == 3 && en == 3); ***- checks if the beginning and ending are correctly found in findRun for 1 occurrence of target***

string g[4] = { "neil", "sonia", "amy", "elena" };

string v[9] = {

"clarence", "neil", "amy", "amy", "ketanji", "ketanji", "ketanji", "amy", "amy"

};

int b;

int e;

assert(findRun(v, 9, "amy", b, e) == true && b == 2 && e == 3); // returns true and

// sets b to 2 and e to 3 ***- checks if the beginning and ending are correctly found in findRun for 2 occurrences of target***

assert(findRun(v, 9, "neil", b, e) == true && b == 1 && e == 1); // returns true and

// sets b to 1 and e to 1 ***- checks if the beginning and ending are correctly found in findRun for 1 occurrence of target at beginning***

assert(findRun(v, 9, "samuel", b, e) == false); // returns false and

// leaves b and e unchanged ***- checks if the beginning and ending are left unchanged and bool returns false as target is not found***

assert(findMin(g, 4) == 2); ***- checks if the lowest value string’s position is returned correctly***

assert(findDifference(h, 4, g, 4) == 2); ***-checks if the lower n value is correctly accounted for and the values line up up to that value. If not, returning the position of the values that differ which in this case is 2.***

string big[10] = { "elena", "john", "amy", "ketanji", "neil", "amy" };

string little1[10] = { "john", "ketanji", "neil" };

assert(subsequence(big, 6, little1, 3)) ; // returns true ***- checks if the values in little1 are correctly found in big and returned true***

string little2[10] = { "amy", "john" };

assert(!subsequence(big, 6, little2, 2)); // returns false ***- checks if the values in little2 are not found in big and thus false is returned***

string little3[10] = { "john", "amy", "amy" };

assert(subsequence(big, 6, little3, 3)); // returns true ***- checks if the values in little3 are correctly found in big and returned true***

string little4[10] = { "john", "john", "amy" };

assert(!subsequence(big, 6, little4, 3)); // returns false ***- checks if the values in little4 are not found in big and thus false is returned***

assert(subsequence(big, 6, little4, 0)); // returns true ***- checks if the values in little4 are correctly found in big and returned true***

assert(moveToEnd(g, 4, 1) == 1 && g[1] == "amy" && g[3] == "sonia"); ***- checks if the target value is correctly moved to the end and that the rest of the values are shifted accordingly***

string ppl[5] = { "john", "sonia", "samuel", "elena", "neil" };

assert(moveToBeginning(ppl, 5, 2) == 2 && ppl[3] == "elena"); // returns 2 ***- checks if the target value is correctly moved to the beginning and that the rest of the values are shifted accordingly***

// people now contains: "samuel" "john" "sonia" "elena" "neil"

string f[4] = { "elena", "amy", "sonia", "john" };

assert(moveToBeginning(f, 4, 2) == 2 && f[0] == "sonia" && f[2] == "amy"); ***- checks if the target value is correctly moved to the beginning and that the rest of the values are shifted accordingly***

string x[5] = { "elena", "elena", "elena", "sonia", "sonia" };

assert(removeDups(x, 5) == 2 && x[1] == "sonia"); ***- checks if the duplicates are removed and the value returned is the size of array without duplicates and the values of the changed array are in the correct order and amount***

string h2[9] = {

"clarence", "neil", "amy", "amy", "ketanji", "ketanji", "ketanji", "amy", "amy"

};

assert(removeDups(h2, 9) == 5 && h2[3] == "ketanji"); // returns 5 ***- checks if the duplicates are removed and value returned is the size of array without duplicates and the values of the changed array are in the correct order and amount***

// d[0] through d[4] now contain "clarence" "neil" "amy" "ketanji" "amy"

// We no longer care what strings are in d[5] and beyond.

string l[9] = {

"clarence", "neil", "amy", "amy", "ketanji", "ketanji", "ketanji", "amy", "amy"

};

assert(removeDups(l, 9) == 5); // returns 5 ***- checks if the duplicates are removed and value returned is the size of array without duplicates and the values of the changed array are in the correct order and amount***

// d[0] through d[4] now contain "clarence" "neil" "amy" "ketanji" "amy"

// We no longer care what strings are in d[5] and beyond.

string c[4] = { "john", "john", "samuel", "sonia" };

string y[4] = { "amy", "elena", "john", "ketanji" };

string z[10];

assert(makeMerger(c, 4, y, 4, z, 10) == 8 && z[5] == "ketanji"); ***- checks if the returned value is equal to the size of the first array + the size of the second array and values are in the correct position in the z array which has the result and altered array that is fully in nondescending order by checking the value at position 5***

string x1[5] = { "amy", "elena", "elena", "ketanji", "samuel" };

string y1[4] = { "clarence", "elena", "john", "sonia" };

string z1[20];

assert(makeMerger(x1, 5, y1, 4, z1, 20) == 9 && z1[5] == "john”); ***- checks if returned value of makeMerger is the sum of the sizes of the arrays (9) and that the 5th position in z has been changed to “john”***

string sc[6] = { "john", "amy", "samuel", "elena", "sonia", "neil" };

assert(divide(sc, 6, "ketanji") == 3); // returns 3  ***checks if the returned value is equal to the position where the first value is greater than (in this case) the divider after putting it in order from least to greatest and thus returns that position (3 in this case)***

// sc must now be

// "elena" "john" "amy" "neil" "sonia" "samuel"

// or "john" "amy" "elena" "samuel" "neil" "sonia"

// or one of several other orderings.

// All elements < "ketanji" (i.e., "john", "amy", and "elena")

// come before all others

// All elements > "ketanji" (i.e., "sonia", "neil", and "samuel")

// come after all others

string sc2[4] = { "john", "sonia", "amy", "neil" };

assert(divide(sc2, 4, "neil") == 2) ; // returns 2 ***checks if the returned value is equal to the position where the first value would = (in this case) the divider after putting it in order from least to greatest and thus returns that position (2 in this case)***

// sc2 must now be either

// "john" "amy" "neil" "sonia"

// or "amy" "john" "neil" "sonia"

// All elements < "neil" (i.e., "amy" and "john") come

// before all others.

// All elements > "neil" (i.e., "sonia") come after all others.