# Obstacles:

One obstacle I encountered was with the function countRuns() as my algorithm counted the number of times the array changed values from a[i] to a[i+1] and returned the number of changes, plus 1. The issue was that if n was given to be 0, the function would still set the number of changes to be zero and therefore return 1. The two was I though about fixing it were to either initialize changes to -1 and automatically increment it once upon entering the for loop iterating through the array which would have looked like

int changes = -1;

for (int i = 0, changes ++; i < n - 1; i++)

The solution I chose, however was just to check immediately for that edge case and return 0 if n was 0

Another obstacle I encountered was while writing the function divide() because I had to choose how to approach organizing the array as it did not have to be completely in order – it just needed to be split by the divider string. However, I chose to order the array completely using a bubble sort algorithm and find where the divider would belong after the fact.

# Test data:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| appendToAll() test cases | | | | |
| Case | Reason | a[] | n | value |
| n = 0 | To make sure the program handles n=0 correctly | [“hello”, “goodbye”] | 0 | “!” |
| empty value string | To make sure the program handles receiving and empty string for value | [“hello”, “goodbye”] | 2 | “” |
| n < 0 | To make sure the program handles n<0 correctly | [“hello”, “goodbye”] | -12 | “#” |
| Good case | To make sure the program works for valid cases | [“hello”, “goodbye”] | 1 | “hello” |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| lookup() test cases | | | | |
| Case | Reason | a[] | n | target |
| n = 0 | To make sure the program handles n=0 correctly | [“hello”, “goodbye”] | 0 | “hello” |
| empty target string | To make sure the program handles receiving and empty string for target | [“hello”, “goodbye”] | 2 | “” |
| n < 0 | To make sure the program handles n<0 correctly | [“hello”, “goodbye”] | -12 | “#” |
| Good case | To make sure the program works for valid cases | [“hello”, “goodbye”] | 2 | “goodbye” |

|  |  |  |  |
| --- | --- | --- | --- |
| positionOfMax() test cases | | | |
| Case | Reason | a[] | n |
| n = 0 | To make sure the program handles n=0 correctly | [“hello”, “goodbye”] | 0 |
| n = 1 | To make sure the program handles only one string being compared correctly | [“hello”, “goodbye”] | 1 |
| n < 0 | To make sure the program handles n<0 correctly | [“hello”, “goodbye”] | -12 |
| Good case | To make sure the program works for valid cases | [“hello”, “goodbye”] | 2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| rotateLeft() test cases | | | | |
| Case | Reason | a[] | n | pos |
| n = 0 | To make sure the program handles n=0 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 0 | 0 |
| pos too large | To make sure the program handles receiving a position after n items correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 3 | 3 |
| n < 0 | To make sure the program handles n<0 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | -12 | 2 |
| Good case | To make sure the program works for valid cases | ["hello", "maybe", "goodbye", "yes", "no"] | 5 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| countRuns() test cases | | | |
| Case | Reason | a[] | n |
| n = 0 | To make sure the program handles n=0 correctly | [“hello”, “goodbye”, “goodbye”, “yes”, “no”, “yes”] | 0 |
| n = 1 | To make sure the program handles only one string being compared correctly | [“hello”, “goodbye”, “goodbye”, “yes”, “no”, “yes”] | 1 |
| n < 0 | To make sure the program handles n<0 correctly | [“hello”, “goodbye”, “goodbye”, “yes”, “no”, “yes”] | -12 |
| Good case | To make sure the program works for valid cases | [“hello”, “goodbye”, “goodbye”, “yes”, “no”, “yes”] | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| flip() test cases | | | |
| Case | Reason | a[] | n |
| n = 0 | To make sure the program handles n=0 correctly | [“hello”, “goodbye”, “goodbye”, “yes”, “no”, “yes”] | 0 |
| n = 1 | To make sure the program handles only one string being flipped | [“hello”, “goodbye”, “goodbye”, “yes”, “no”, “yes”] | 1 |
| n < 0 | To make sure the program handles n<0 correctly | [“hello”, “goodbye”, “goodbye”, “yes”, “no”, “yes”] | -12 |
| Good case | To make sure the program works for valid cases | [“hello”, “goodbye”, “goodbye”, “yes”, “no”, “yes”] | 2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| differ() test cases | | | | | |
| Case | Reason | a1[] | n1 | a2[] | n2 |
| n1 = 0 | To make sure the program handles n1=0 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 0 | ["hello", "maybe", "goodbye", "yes", "no"] | 2 |
| n2= 0 | To make sure the program handles n2=0 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 2 | ["hello", "maybe", "goodbye", "yes", "no"] | 0 |
| n1 > n2 and a1[] = a2[] | To make sure the program handles n1 > n2 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 3 | ["hello", "maybe", "goodbye", "yes", "no"] | 2 |
| n1 < n2 and a1[] = a2[] | To make sure the program handles n1 > n2 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 1 | ["hello", "maybe", "goodbye", "yes", "no"] | 2 |
| a1 != a2  n1, n2 > 0 | To make sure the program works for valid cases | ["hello", "maybe", "goodbye", "yes", "no"] | 5 | ["goodbye", "yes", "no", "hello", "maybe"] | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| subsequence() test cases | | | | | |
| Case | Reason | a1[] | n1 | a2[] | n2 |
| n1 = 0,  n2 = 0 | To make sure the program handles n1=0, n2=0 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 0 | ["hello", "maybe", "goodbye"] | 0 |
| n2= 0 | To make sure the program handles n2=0 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 2 | ["hello", "maybe", "goodbye"] | 0 |
| n2 not found in n1 | To make sure the program handles n2 not found in n1 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 5 | ["yellow", "maybe", "goodbye"] | 3 |
| a1 = a2 | To make sure the program handles a1 = a2 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 1 | ["hello", "maybe", "goodbye", "yes", "no"] | 2 |
| a2 found within a1 | To make sure the program works for valid cases | ["hello", "maybe", "goodbye", "yes", "no"] | 5 | ["hello", "maybe", "goodbye"] | 3 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| lookupAny() test cases | | | | | |
| Case | Reason | a1[] | n1 | a2[] | n2 |
| n1 = 0,  n2 = 0 | To make sure the program handles n1=0, n2=0 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 0 | ["hello", "maybe", "goodbye"] | 0 |
| n2= 0 | To make sure the program handles n2=0 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 2 | ["hello", "maybe", "goodbye"] | 0 |
| a2 not found in a1 | To make sure the program handles a2 not found in a1 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 5 | ["yellow", "naybe", "doodbye"] | 3 |
| n1 = 0 | To make sure the program handles n1 = 0 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 0 | ["hello", "maybe", "goodbye"] | 2 |
| a2 found within a1 | To make sure the program works for valid cases | ["hello", "maybe", "goodbye", "yes", "no"] | 5 | ["hello", "maybe", "goodbye"] | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| divide() test cases | | | | |
| Case | Reason | a[] | n | divider |
| n = 0 | To make sure the program handles n=0 correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 0 | “new” |
| Divider at end | To make sure the program handles divider at end correctly | ["hello", "maybe", "goodbye", "yes", "no"] | 3 | “z” |
| Divider at start | To make sure the program handles divider at start correctly | ["hello", "maybe", "goodbye", "yes", "no"] | -12 | “a” |
| Divider in middle | To make sure the program works for normal cases | ["hello", "maybe", "goodbye", "yes", "no"] | 5 | “new” |