The most formidable obstacle I faced involved the challenge for Separate: designing the function without the use of an additional array. While I found every other function easy, this one took more time than the others combined. At first, I attempted to use recursion but found it a bit too complicated to code. Then I worked the problem out on paper and developed a basic algorithm involving the rotation function. If the item was larger than separator, I would rotate it left to shift it to the right end. If the item was smaller than the separator, I would keep it there and look at the next item. If the item was equal to the separator, I would rotate it left if it was not at the middle or keep it in its position if it was at the middle. This allowed me to solve the problem. Some minor other problems I encountered involved briefly getting stuck with how to manage the n variable for interesting elements, which I figured out easily that I would need to write it as n – 1.

**TEST CASES:**  
test -1 for n, and n1 and n2 if applicable for ALL CASES

**string h[7] = { "mick", "marie", "fiona", "rudy", "", "gordon", "lindsey" };**

appendToAll(h, 0, "fiona"); test 0 case, should return 0  
appendToAll(h, 2, "fiona"); test an in-between case  
appendToAll(h, 7, "fiona"); test max case, all should be edited

lookup(h, 0, "fiona"); test 0 case, should return -1 since no match  
lookup(h, 3, "fiona"); test 3 case, should return 2 since match  
lookup(h, 7, "fiona"); test max case, should return 2 since match

positionOfMax(h, 0); test 0 case, should return -1 since no elements  
positionOfMax(h, 3); test an in-between case, should return 0 for mick  
positionOfMax(h, 7); test max case, should return 3

rotateLeft(h, 4, 0); test 0 case for pos and in-between for n  
rotateLeft(h, 2, 4); pos > n – 1, return -1  
rotateLeft(h, 7, 5); test in-between for pos and max case for n  
  
**string g[7] = { "mick", "mick", "fiona", "rudy", "", "mick", "rudy" };**  
  
countRuns(g,7); test max case, should return 4  
countRuns(g,0); test for 0 case, should return 0  
countRuns(g,3); test for in-between case, should return 2

flip(h,7); test max case   
flip(h,0); test zero case should return 0  
flip(h,3); test case for odd  
flip(h,4); test case for even

differ(h, 1, g, 2); test case for both no difference, returns lowest -> 1  
differ(h, 0, g, 2); test for 0, return -1 since error in comparing to no elements  
differ(h, 7, g, 2); test for max and in-between

**string a[7] = { "mick", "marie", "fiona", "marie", "mick", "gordon", "lindsey" };  
string b[7] = { "mick", "gordon", "fiona", "rudy", "", "mick", "rudy" };**

subsequence(a, 7, b, 2); test case for max including a subsequence  
subsequence(a, 7, b, 3); test case for not including subsequence  
subsequence(a, 7, b, 0); test case for 0 as a subsequence of any n1

**string c[7] = { "mick", "jack", "fiona", "marie", "mick", "gordon", "lindsey" };  
string d[7] = { "bill", "gordon", "fiona", "rudy", "", "jack", "rudy" };**lookupAny(a, 5, b, 6); test case for two in-betweens  
lookupAny(a, 2, b, 4); test case for no matches

**string num[10] = { "3","5","7","2","6","4","8","9","0","83"};**

separate(num, 3, "4"); test case for in-between  
separate(num, 10, "99"); test case for all items less than separator  
separate(num, 10, "0"); test case separator equal to lowest item  
separate(num, 5, "9"); test case where it needs to return n, all items less than separator