1. Notable Obstacles:
   1. The first issue I ran into was in the making of the flip function. At first I had a forloop that ran until the integer got up to the value of n. This was incorrect as elements that had already been flipped wer being flipped once again, reversing what was being done. To fix this I thought about the nature of the flip and integers and decided that the maximum the integer arrayPos should be n divided by 2, as with odd numbers it would round down and leave the object in the middle alone, which was perfect for the uses of the funciton.
   2. Another issue I had was with the logic I should use for subsequence. I tried many approaches at first including just an if statement nested within a forloop. After mulling over it for a while I decided to think about how I would go about doing the task and wrote out my work. This led me to try to apply that line of thinking to my code and I found that nesting a forloop within a forloop for comparison along with an integer variable count that would be reset would allow me to accurately see if a2 would be a subsequence of a1 with the help of an if and else statement.
   3. The most notable issue I had though was when I realized that I did not want to move strings that were properly placed as a result of the seperate function. To get around having to flip two strings that are both greater than or less than seperator, I made a check to see if the other string to be swapped has the same condition as the string we are swapping. Through counting this with an integer I was able to avoid that swap and instead swap with the next string that does not share the same condition of being greater than or less than as the original string.
2. Test Cases:

**ALL ARRAYS USED COME FROM SPEC**

appendtoAll(people, 0, “yeet”) - should give 0 and not modify any strings

appendtoAll(people, -1, “yeet”) - check if error code correctly give

appendtoAll(people, 4, “nnn”) - check if string is properly appended to 4 strings

appendtoAll(people, 2, “\_!”) - check if value is properly appended to 2 strings

lookup(people, 5, “adam”) - check for -1 when target is not present

lookup(people, 0, “rudy”) - check for -1 as no elements are being checked

lookup(people, -99, “mike”) - check for correct error reutrn, -1 as n is neg.

lookup(people, 5, “rudy”) - check for return of position when target is present

positionofMax(persons, -27) - check for -1 as n is negative

positionofMax(persons, 0) - return -1, no elements have been checked so no max

positionofMax(persons, 6) - check for correct return of 3

positionofMax(persons, 3) - check for proper function, return2

rotateLeft(politician, -5, 1) - check for -1 as n is negative

rotateLeft(politician, 0, -1) - check for -1 as pos is negative

rotateLeft(politician, 0, 3) - check for -1 as n should be greater than pos

rotateLeft(politician, 5, 0) - check if all rotated, return 0

countRuns(d,9) - check for correct counting, return 5

countRuns(d, 0) - return 0 as no elements are being checked

countRuns(d, 3) - return 3 as 3 different runs

countRuns(d, -6) - check for -1 as n is negative

flip(folks, 3) - check if flipped, return 2

flip(folks, 0) - check if none flipped, return 0

flip(folks, -9) - check if none flipped, return -1 as n neg.

flip(folks, 7) - check if all flipped, return 7

differ(folks, -5, group, 6) - return -1 as n1 negative

differ(folks, 4, group, -3) - return -1 as n2 negative

differ(folks, 0, group, 3) - check for return 0 as no elements actually checked

differ(folks, 3, group, 0) - check for return 0 as none being compared

differ(folks, 5, group, 4) - check for proper function, should return 3

subsequence(names, 2, names1, 3) - return -1 as n1 < n2

subsequence(names, 0, names1, 0) - check return 0, as stated in spec

subsequence(names, 3, names1, 0) - same reason as prior

subsequence(names, -3, names1, 3) - return -1 as n1 negative, error

subsequence(names, 6, names1, 3) - return 1 checks proper function

subsequence(names, 6, names1, 1) - check for return -1 as not subsequence

lookupAny(names, 0, set1, 4) - check return -1, as none checked

lookupAny(names, 3, set1, 0) - check return -1, none in set1 being checked

lookupAny(names, -3, set1, 4) - check return -1 as n1 < 0

lookupAny(names, 4, set1, -2) - return -1 as n2 <0

lookupAny(names, 6, set1, 3) - return -1 as none matching, proper function

lookupAny(names, 6, set1, 4) - check for return 1 as a1 has “marie” there

separate(persons, -3, “gordon”) - check for return -1 as n negative

separate(persons, 0, “gordon”) - check for return 1, none have been moved

separate(persons, 6, “gordon”) - check for return 3, proper function