Table of Contents

1. Questions from PDF

Explanation – Why test cases you have identified fail or are successful?

SORT

sortTestLinear
- Successful, because array sorted in ascending order.
  i.e. {5, 3, 4, 2, 1} -> {1, 2, 3, 4, 5}

sortTestSorted
- Successful, because sorted array sorts back to itself.
  i.e. {1, 10, 100, 1000, 10000} -> {1, 10, 100, 1000, 10000}

sortTestReverseSorted
- Successful, because array is resorted in ascending order.
  i.e. {10000, 1000, 100, 10, 1} -> {1, 10, 100, 1000, 10000}

sortTestSmall
- Successful, because numbers are sorted in ascending order.
  i.e. {16, 4, 2, 0, 8} -> {0, 2, 4, 8, 16}

sortTestLarge
- Successful, because numbers are sorted in ascending order.
  i.e. {1024, 64, 256, 65536, 8} -> {8, 64, 256, 1024, 65536}

sortTestPos32Bits
- Fail, because variables that are declared as type int can only hold 32-bit numbers (depending on the operating system), anything beyond that will be considered as buffer overflow.

sortTestNeg32Bits
- Fail, because variables that are declared as type int can only hold 32-bit numbers (depending on the operating system), anything beyond that will be considered as buffer overflow.

sortTestRepeated
- Successful, because all the ones are sorted together in ascending order.
  i.e. {5, 1, 1, 1, 5} -> {1, 1, 1, 5, 5}

sortTestZero
- Successful, because all zeroes are smaller than positive numbers, and are sorted in ascending order.
  i.e. {1, 0, 0, 2, 1} -> {0, 0, 1, 1, 2}

sortTestNegative
- Successful, because negative numbers are sorted in ascending order.
  i.e. {-1, 0, -10, -5, -3, 10} -> {-10, -5, -3, -1, 0, 10}

sortTestDecimal

- Successful, because numbers after decimal point is truncated, leaving integer parts. The array is the sorted as if it is an integer array.
  i.e. {432.133, -11.5, 12.3, 0.39, 2.4} -> {-11, 0, 2, 12, 432}

sortTestDecimalWithIntegers

- Successful, because numbers after decimal point is truncated, leaving integer parts. Integer numbers are left untouched.
  i.e. {432.133, -11, 12, 0.39, 2.4} -> {-11, 0, 2, 12, 432}

sortTestOneElement

- Successful, because it just compares with itself.
  i.e. {10} -> {10}

sortTestEmpty

- Fail because in C, we cannot declare arrays of size zero, and its useless to sort an empty array.
  i.e. {} -> {}

SHUFFLE

shuffleTestEmpty

- Fail, because in C, we cannot declare arrays of size zero, and its useless to shuffle an empty array.

shuffleTestOneElement

- Fail, because if statement in shuffle function prevents elements in the array to map back to its original index.

shuffleTestIndividual

- Successful, because if statement in shuffle function prevents elements in the array to map back to its original index.

shuffleTestToItself

- Successful, because if statement in shuffle function prevents elements in the array to map back to its original index.

shuffleTestNotBias

- Fail, because shuffle algorithm is bias towards some pattern more than others.

sortAndShuffle.c

- Toggle DEBUG flag to prevent random seed generator from being seeded (Set to 1 for testing).
- Toggle DEFAULT flag to switch between which shuffle algorithm to use. There's two, the unbiased version and the given version. Note that the unbiased algorithm will fail the shuffleTestOneElement, shuffleTestIndividual and shuffleTestToItself because it assumes that the elements in the array can map back to itself.

test.c

- Toggle VERBOSE flag to know algorithm is bias towards which pattern.
- Toggle DEBUG and VERBOSE flags to know percentage of bias.
- Toggle DEVELOP flag to know each tries' patterns.

2. <u>Implementation Details</u>

Programming Language

C

Compiler Used

gcc (Debian 7.2.0-19) 7.2.0

Function Information

***NOTE*:** More specific information regarding the respective functions are located in their respective files above their function definitions.

Function Information	*Function Location*
sortTestLinear ()	test.c
Function tests a given unsorted array, if it sorts linearly (i.e. in ascending order).	
sortTestSorted()	test.c
Function tests a given sorted array if its elements after sort will be untouched.	
sortTestReverseSorted()	test.c
Function tests a given reversed sorted array if it will sort back in ascending order.	
sortTestSmall()	test.c
Function tests a given array with small numbers if it sorts in ascending order.	
sortTestLarge()	test.c
Function tests a given array with large numbers if it sorts in ascending order.	
sortTestPos32Bits()	test.c
Function tests a given array with extremely large numbers if it sorts in ascending order.	
sortTestNeg32Bits()	test.c
Function tests a given array with extremely small numbers if it sorts in ascending order.	
sortTestRepeated()	test.c
Function tests a given array with repeated numbers if it sorts in ascending order.	
sortTestZero()	test.c
Function tests a given array with zeroes if it sorts in ascending order.	
sortTestNegative()	test.c
Function tests a given array with negative integers if it sorts in ascending order.	
sortTestDecimal()	test.c
Function tests a given array with decimals if it sorts in ascending order.	
sortTestDecimalWithIntegers()	test.c
Function tests a given array of decimals and integers if it sorts in ascending order.	
sortTestOneElement()	test.c
Function tests a given array with only 1 elements if it sorts in ascending order.	
sortTestEmpty()	test.c
Function tests a given array with no elements if it sorts in ascending order.	

shuffleTestEmpty()   test.c
Function tests a given array with no elements if it shuffles.

shuffleTestOneElement()   test.c
Function tests a given array with only 1 element if it shuffles to itself.

shuffleTestIndividual()   test.c
Function tests a given array by shuffling it 1000 times until we shuffled an array that has elements that map back to itself. If no such array exists, shuffle algorithm passes the test. NOTE: Function assumes that the elements in the array is not suppose to shuffle back to itself, whereas a real shuffle can shuffle back to itself.

shuffleTestToItself()   test.c
Function tests given array by shuffling it 1000 times until we shuffled an array that maps all its elements back to itself. If no such array exists, shuffle algorithm passes the test. NOTE: Function assumes that the array is not suppose to shuffle back to itself, whereas a real shuffle can shuffle back to itself.

shuffleTestNotBias()   test.c
Function tests a given array for any bias. By testing a specific example where the number of elements is 3, and the array elements are 1, 2 and 3. NOTE: Function assumes that mapping back to itself is a shuffle, i.e., 1,2,3 after shuffle, can be 1,2,3 - 1,3,2 - 3,2,1 - 2,1,3.