↓**Mutation** \ **Test_method** →	*Randomized*	*Pairwise*
Line #25 in ListOperators.java (method: bubbleSort) Change the message of the NullPointerException	Only one test required (not repeatedTest)	Only one test required (not repeatedTest)
Line #31 in ListOperators.java (imethod: bubbleSort) Change “int swap = A[j-1]” to “int swap = A[j]” Instead of swapping two elements in bubbleSort(), one element is changed to be equal to the other element. This may return a list that does not have the same set of elements as the input array.	Passes 157/210 repeated tests in total <u>trueMembershipTest</u> Passes 52/105 test cases A key that exists in the input array can be changed to some other element in the list. A list that only contains one integer will not be affected by the mutation. The errors will become more frequent as the size of the array decreases. This is because the likelihood of the array already being sorted increases which lowers the probability of the mutated line being reached. <u>falseMembershipTest</u> Passes 105/105 test cases searches for a key that doesn’t exists. The mutation only changes elements in the array to something that already exists in the list. The tests will not find a key if it didn’t exist in the first place.	Passes 2/211 repeated tests The first case is an array full of zero’s. Since elements can only be changed into something else in the array, all elements will be zero either way. The second case is when all elements are zero, except for the last element that is >0. The if-statement on line #31 will never be evaluated to true because the array is already sorted. The mutated line will not be reached in this case.
Line #62 in ListOperators.java (method: binarySearch) Remove/comment line #61.	Only one test required (not repeatedTest) The randomized repeated tests does not test empty lists since one case is sufficient. If the repeated tests allowed empty lists, more cases would fail.	Only one test required (not repeatedTest)
Line #69 in ListOperators.java (method: binarySearch) Change the logical operator “&&” to “||” in the predicate of the while-loop.	No repeated test case terminates If the key does not exist, the first clause “key != A[x]” will always be true. Because of this, the loop will never exit.	Only one repeated test case terminates. This case uses an input array full of zeros. The first clause “key != A[x]” will always be false since A[x] == key for all 0<= x <A.length-1. The second clause will eventually become false since the if-statement on line #71 (“key<A[x]”) will always be false. This will increase the value of the variable *l* until it becomes greater than or equal to the variable *r*, thus breaking the loop.

Line #73 in ListOperators.java (method: binarySearch) Change *"r = x-1*" to be *"r = x+1"*	No repeated test case terminates. If the first element is greater than the key, r will be incremented by one. Since "l<=r" is true initially, and the value of r is incremented while the value of l is unchanged, the variable l will always be lesser than the variable r. This leads to an infinite loop. Small size on the input array increases the risk of unintentionally passing tests. For instance, an array with all zeros would not detect the mutation.	Passes 211/211 test cases The key that is being searched for is an integer greater than zero. Since all other elements are <= 0, the clause in the if-statement on line #71 will never become true. Since the mutated line in the code is never reached, all tests are still passed. In order to detect the mutation, the array can be sorted in reverse order, or the key that is being searched for can be changed to a negative value.
Line #105 in ListOperators.java (method: membership) Change the return-statement from "n > -1" to "n > 0"	Passes 86/210 test cases in total <u>trueMembershipTest</u> Passes 86/105 of test cases. The ones that passes is when the key that is being searched for is the smallest element. This will make it the first element when sorted, and it's index will be 0 which is not greater than itself. Smaller sizes of array will give more errors since the risk of the key being at index 0 increases. <u>falseMembershipTest</u> Passes 105/105 test cases. This is because an element that should not exist will not mistakenly appear as the first element.	Passes 211/211 test cases In PairWiseTestList.java, a static final variable *nonZero* is declared to be =3 (which is greater than 0). When searching for the key=3 in the test arrays, it will always appear last in the array when sorted. If we change *nonZero* to be negative, all of the 211 tests will fail except for the first case with all zeros.