Assignment

Category Partition Method

Sort

Category	Condition	Inputs	Justification
1	List Class is ArrayList	An List that implements ArrayList	Due to the Java documentation. A List is an interface and ArrayList is an implementation of a List. Note ArrayList RandomAccess.
2	List Class is LinkedList	An List that implements LinkedList	LinkedList is a different implementation of List. Since pointers are involved then the increased complexity may lead to issues. Note LinkedList does not implement RandomAccess.
3	List has size of 0	The input would be the empty list. list = []	To see if the function can handle sorting an empty list edge case.
4	List has size of 1	The input will be a list of one element. list = [x]	To see if the function can handle sorting a list of one element. Therefore list x == sort list x.
5	List has size of many	The input will be a list of many elements. list = [x_0, x_1, …, x_n]	To see if a list of many elements can be sorted. As this is the most likely situation for this function.
6	List is a list of Int	All list elements will be a int e.g. List<int>	To see if a list of Integers can be sorted.
7	List is a list of Strings	All list elements will be a int e.g. List<String>	To see if a list of Strings can be sorted. Note a Strings natural ordering will be in Lexicographic order.
8	List is a list of Chars	All list elements will be a int e.g. List<Char>	To see if a list of Characters can be sorted. Note that chars are compared numerically
9	List is a list of Double	All list elements will be a int e.g. List<Double>	To see if a list of real numbers can be sorted. Since floating point precision may be an issue.

Rotate

Category	Condition	Inputs	Justification
10	List Class is ArrayList	An List that implements ArrayList	Due to the Java documentation. A List is an interface and ArrayList is an implementation of a List. Note ArrayList RandomAccess.
11	List Class is LinkedList	An List that implements LinkedList	LinkedList is a different implementation of List. Since pointers are involved then the increased complexity may lead to issues. Note LinkedList does not implement RandomAccess.
12	List has size of 0	The input would be the empty list. list = []	To see if the function can handle rotate an empty list edge case. Secondly, the documentation uses "(i - distance) mod list.size()" to calculate its position but if the list is of size zero then the mod function should fail. The documentation doesn't mention the case of the empty list and so will be tested.
13	List has size of 1	The input will be a list of one element. list = [x]	To see if the function can handle sorting a list of one element. Therefore list x == rotate list x.
14	List has size of many	The input will be a list of many elements. list = [x_0, x_1, …, x_n]	To see if a list of many elements can be rotate. As this is the most likely situation for this function.
15	List is a list of Int	All list elements will be a int e.g. List<int>	To see if a list of Integers can be rotated.
16	List is a list of Strings	All list elements will be a int e.g. List<String>	To see if a list of Strings can be rotated.
17	List is a list of Chars	All list elements will be a int e.g. List<Char>	To see if a list of Chars can be rotated.
18	List is a list of Double	All list elements will be a int e.g. List<Double>	To see if a list of real numbers can be sorted. Since floating point precision may be an issue.
19	distance is negative	Distance will be a negative signed integer in the range -2^{31} <= distance <= -0	To see if the list rotates in the negative direction
20	distance is positive	Distance will be a positive signed integer in the range +0 <= distance <= 2^{31} - 1	To see if the list rotates in the positive direction
21	Distance is max Int	Distance will be the largest value that can be represented as an Integer in Java distance = 2^31 - 1	To see if the function can parse the largest 32 bit two's compliment representation of an integer

Category	Condition	Inputs	Justification
22	Distance is min Int	Distance will be the smallest value that can be represented as an Integer in Java distance = -2^31	To see if the function can parse the smallest 32 bit two's compliment representation of an integer
23	Distance is 0	Distance will be a value of 0	Zero should return the original list.
24	distance 1	Distance will be only 1	To see if it moves once
25	many	Distance will be many e.g. a value of 100 but not as large as a integer limit	To see if it moves many times
26	greater than list.size()	Distance = list.size + n	To see if it moves correctly and should rotate n times

Min

Category	Condition	Inputs	Justification
27	Collection that implements List	An List that implements LinkedList	LinkedList is a different implementation of List. Since pointers are involved then the increased complexity may lead to issues. Note LinkedList does not implement RandomAccess.
28	Collection Implements Queue	An Queue that implements PriorityQueue	A Queue is a type of "first in first out" data structure. A PriorityQueue is one where an item is ranked. So this priory complexity may lead to issues
29	Collection Implements Set	An Set that implements HashSet	A Set is another form of data structure where no duplicates exist. Furthermore, a Set does not guarantee that the iteration order will remain constant. This may lead to issues when finding the minimum value in the Set
30	List has size of 1	The input will be a list of one element. list = [x]	To see if the function can handle sorting a list of one element. Therefore list x == min list x.
31	List has size of many	The input will be a list of many elements. list = [x_0, x_1, …, x_n]	To see if a list of many elements can return its min. As this is the most likely situation for this function.
32	Collection element is of type Int	All list elements will be a int e.g. List<int>	To see if a list of Integers can return its min.
33	Collection element is of type String	All list elements will be a int e.g. List<String>	To see if a list of Strings can return its min. Note a Strings natural ordering will be in Lexicographic order.

Category	Condition	Inputs	Justification
34	Collection element is of type Char	All list elements will be a int e.g. List<Char>	To see if a list of Characters can return its min. Note that chars are compared numerically
35	Collection element is of type Double	All list elements will be a int e.g. List<Double>	To see if a list of Doubles can return its min. With the added complexity of floating point precision errors.

Test Cases

Sort

Category	Condition
1	List Class is ArrayList
2	List Class is LinkedList
3	List has size of 0
4	List has size of 1
5	List has size of many
6	List is a list of Int
7	List is a list of Strings
8	List is a list of Chars
9	List is a list of Double

Combination of Tests	Input	Justification
1,3,6	An empty ArrayList of type int. E.g. List<int> n = new ArrayList<int>();	To see if sort correctly returns an empty list of ints.
2,4,7	A LinkedList of type String with only one element. I.e. n.size = 1. The Sting will be 10 concatenated “a”s	To see if sort correctly returns a list of one String. Strings are stored by value. Therefore, the combination of LinkedList pointers, as well as string pointers, may lead to issues.
2,5,8	A LinkedList of type Double with of size 500. Each double will be a ‘1.0’	To see if a reasonable use of sort behaves as expected.

Rotate

Category	Condition
10	List Class is ArrayList
11	List Class is LinkedList
12	List has size of 0
13	List has size of 1
14	List has size of many
15	List is a list of Int
16	List is a list of Strings
17	List is a list of Chars
18	List is a list of Double

Category	Condition
19	distance is negative
20	distance is positive
21	max Int
22	min Int
23	0
24	1
25	many
26	greater than list.size()

Combinations	Input	Justifications
10, 12, 15, 19, 23	An empty ArrayList of type Int. Distance will be negative 0	To see that rotate correctly returns the empty list of ints. The use of -0 is to see if java correctly interpreted as 0. This is due to java using a Two's complement representation for integers. So hopefully, it does not suffer from this issue.
11, 13, 16, 20, 21	A LinkedList of size 1 and a type of String. The String will be 10 concatenated "a"s. With a positive distance of the integer maximum signed value.	Again, due to a LinkedList, and String's use of pointers. This is to see that a List of one element is correctly rotated and returned. A maximum int will be used to see if a very large value can be parsed.
11, 14, 17, 19, 22	A LinkedList of type Char and a size of 500. Each char will be an 'a'. The Distance will be a negative integer minimum value.	To see if a reasonable use of sort behaves as expected. Furthermore, a LinkedList doesn't implement the RandomAccess interface, and due to the list being greater than a size of 100. Then rotate uses a different implementation. One that splits the list in two and reverses it. Therefore, this could be quite a challenging task due to all the pointers in the LinkedList involved. The method used found in "Section 2.3 of Jon Bentley's *Programming Pearls* (Addison-Wesley, 1986)."

min

Category	Condition
10	Collection Implements List
27	Collection Implements Queue
29	Collection Implements Set
31	List has size of 1
32	List has size of many
33	List has size of many many
34	Collection element is of type Int
35	Collection element is of type float
36	Collection element is of type Char
37	Collection element is of type Double

Combination	Input	Justification
10, 31, 33	An List of size 1 of type int	To see if it returns the only element in the list
27, 33, 35	A queue of type float with many many elements. Where the element will be 10000 f to 1	To see if the correct value is return from a very large list and ordered in the worst case for min. A PriorityQueue will also be used to see if the use of priorities effect the results
29, 32, 36	A set of type Double with 500 doubles from 0 to 499	To see if a reasonable use of sort behaves as expected. Secondly, a Set doesn't guarantee the iteration order which may lead to wrong results.

Metamorphic Relations

Sort

Relation One

x’ = x.append(y): where y is some arbitrary value
length(x) < length(x’)

So: z = Collection.sort(x), z’ = Collections.sort(x’)
Therefore: length(z) < length(z’)

z	x
[1, 2, 3]	[3,2,1]
[1, 2, 3]	[2, 3, 1]
[-3, -2, -1, 0, 1, 2, 3]	[3, 2, 1, 0, -1, -2, -3]

z’	x’	length(z) < length(z’)
[0, 1, 2, 3]	[3,2,1,0]	TRUE
[0, 1, 2, 3]	[2, 3, 1, 0]	TRUE
[-4, -3, -2, -1, 0, 1, 2, 3]	[3, 2, 1, 0, -1, -2, -3, -4]	TRUE

Relation Two

A list X is a list of n elements such that X = [x_0, x_1, … , x_n]
Let X + 1 mean X + 1 = [x_0 + 1, x_1 + 1, … , x_n + 1]
Therefore:
X’ = X + 1

So: z = Collection.sort(X), z’ = Collections.sort(X’)
Therefore: Z’ == Z + 1 where Z is a list of n elements

z	x
[1, 2, 3]	[3,2,1]
[1, 2, 3]	[2, 3, 1]
[-3, -2, -1, 0, 1, 2, 3]	[3, 2, 1, 0, -1, -2, -3]

z’	x’	Z’ = Z + 1
[2,3,4]	[1+ 1, 2 + 1, 3 + 1]	TRUE
[2, 3, 4]	[2 + 1, 3 + 1, 1 + 1]	TRUE
[-2, -1, 0, 1, 2, 3, 4]	[3 + 1, 2 + 1, 1 +1 , 0 + 1, -1 + 1, -2 + 1, -3 +1]	TRUE

Note: This relation doesn’t naturally hold on Strings. However, appending one character to the string could be regarded as equivalent. This will be implemented but commented out.

Min

Relation One

A list X is a list of n elements such that X = [x_0, x_1, … , x_n]
Let X + 1 mean X + 1 = [$x_0 + 1$, $x_1 + 1$, … , x_n + 1]
Therefore:
X’ = X + 1

So: z = Collection.min(x), z’ = Collections.min(x’)
Therefore: z’ == z + 1

z	x
1	[1, 2, 3]
1	[3, 1, 2]
-1	[-1, 0, 1]

z’	x’	z’ == z + 1
2	[2,3,4]	TRUE
2	[4, 2, 3]	TRUE
0	[0, 1, 2]	TRUE

Relation Two

A list X is a list of n elements such that X = [x_0, x_1, … , x_n]
Let -X mean -X = [$-x_0$, $-x_1$, … , $-x_n$]

Therefore:
X’ = max(-X) assumption where max returns the largest value in the collection

so z = Collection.min(X), z’ = Collection.min(X’)

Therefore: -z == z’

z	x
1	[1, 2, 3]
1	[3, 1, 2]
-1	[-1, 0, 1]

z’	x’	-z == z’
-1	[-1]	TRUE
-1	[-1]	TRUE
1	[1]	TRUE

Rotate

Relation One

d’ = d + length(x)
So: z = Collection.min(x, d), z = Collection.min(x, d’)
Therefore: z == z’

z	x	d
[2, 1, 3]	[3, 2,1]	2
[4, 3, 2, 1, 5]	[5,4,3,2,1]	4
[1, 2, 3]	[3,1,2]	2

z’	x’	d’	z == z’
[2, 1, 3]	[3,2,1]	2 + 3	TRUE
[4, 3, 2, 1, 5]	[5,4,3,2,1]	4 + 5	TRUE
[1, 2, 3]	[3,1,2]	2 + 3	TRUE

Relation Two

This relation will also be used to see if the statement in the documentation that the “method has no effect on the size of the list” holds.

x’ = x.append(y): where y is some arbitrary value
<u>length(x) < length(x’)</u>

So: z = Collection.rotate(x), z’ = Collections.rotate(x’)
Therefore: <u>length(z) < length(z’)</u>

z	x	d
[2, 1, 3]	[3, 2,1]	2
[4, 3, 2, 1, 5]	[5,4,3,2,1]	4
[1, 2, 3]	[3,1,2]	2

z’	x’	d’	length(z) < length(z’)
[1, 0, 3, 2]	[3,2,1,0]	2	TRUE
[3, 2, 0, 5, 4]	[5,4,3,2, 1, 0]	4	TRUE
[2, 0, 3, 1]	[3,1,2, 0]	2	TRUE

Remarks

In conclusion, none of the tests failed to conclude. This was to be expected due to Java's wide use and testing over its lifetime.

Surprisingly, the test case "metaOneCategoryTwo()
" which used the first metamorphic relation and the categories 11, 13, 16, 20, and 21 passed. I personally thought this method would fail due to an arithmetic overflow during runtime. This leads distance to be a very small negative integer. My thinking is that this change of sign would lead to an integer that wouldn't be divisible by the length of the list, thus leading to a broken relationship. However, by imagining the domain of integers to be a cycle, rather than a line with boundaries, leads to its passing not to be surprising at all.