KEY: Orange highlight = Test Added during this milestone
Red highlight = Impossible test case

Test	Equivalence class partitioning(Wect, Sect, Wrect, Srect)

testAdd

- Variables:
  - List length L
  - Object O
- Equivalence classes
  - L1:L=0
  - L2:L>0
  - O1: O does not already exist in list
  - O2: O already exists in list
- Wect

ID	L	O
WECT1	L1	O1
WECT2	L2	O2

- Sect

ID	L	I
SECT1	L1	O1
SECT2	L2	O1
SECT3	L1	O2
SECT4	L2	O2

- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are covered by testing of a null variable list

testClear

- Variables:
  - List length L
- Equivalence classes
  - L1: L=0
  - L2: L>0
- Thus, wect and sect are both covered by using an empty list and a nonempty
  - This is already done
- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and rect are covered by testing of a null variable list

testContains

- Variables:

- List length L
    - Object O
- Equivalence classes
    - L1:L=0
    - L2:L>0
    - O1:O is in list
    - O2:O is not in list
- Wect

ID	L	O
WECT1	L1	O1
WECT2	L2	O2

- Sect

ID	L	I
SECT1	L1	O1
SECT2	L2	O1
SECT3	L1	O2
SECT4	L2	O2

- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are covered by WECT1, SECT1
- Wrect and srect are also covered by testing of a null variable list

testContainsAll

- Variables:
    - List length L
    - Object list O
- Equivalence classes
    - L1: L=0
    - L2: L>0
    - O1: All of O is in list
    - O2: Not all of O is not in list
- Wect

ID	L	O
WECT1	L1	O1
WECT2	L2	O2

- Sect

ID	L	I
SECT1	L1	O1

SECT2	L2	O1
SECT3	L1	O2
SECT4	L2	O2

- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are also covered by testing of a null variable list

testEquals

- Variables:
    - List length L
- Equivalence classes
    - L1:L=0
    - L2:L>0
- Thus, wect and sect are both covered by using an empty list and a nonempty
    - This is already done
- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are covered by testing of a null variable list

testHashCode

- Variables:
    - List length L
- Equivalence classes
    - L1:L=0
    - L2:L>0
- Thus, wect and sect are both covered by using an empty list and a nonempty
    - This is already done
- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are covered by testing of a null variable list

testGet

- Variables:
    - List length L
    - Index I
- Equivalence classes
    - L1: L=0
    - L2: L>0
    - I1: I<0
    - I2: I>=0&&I<L
    - I3: I>=L

- Wect

ID	L	I
WECT1	L1	I1
WECT2	L2	I2

WECT3	L1	I3

- Sect

ID	L	I
SECT1	L1	I1
SECT2	L2	I1
SECT3	L1	I2
SECT4	L2	I2
SECT5	L1	I3
SECT6	L2	I3

- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are covered by WECT1, WECT3, SECT1, SECT2, SECT5, and SECT6
- Wrect and srect are also covered by testing of a null variable list

testIndexOf

- Variables:
    - List length L
    - Object O
- Equivalence classes
    - L1: L=0
    - L2: L>0
    - O1: O is in list
    - O2: O is not in list
- Wect

ID	L	O
WECT1	L1	O1
WECT2	L2	O2

- Sect

ID	L	I
SECT1	L1	O1

SECT2	L2	O1
SECT3	L1	O2
SECT4	L2	O2

- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are covered by WECT1, SECT1
- Wrect and srect are also covered by testing of a null variable list

testIsEmpty	● Variables: ○ List length L ● Equivalence classes ○ L1: L=0 ○ L2: L>0 ● Thus, wect and sect are both covered by using an empty list and a nonempty ○ This is already done ● Wrect and Srect would involve L < 0 but this is not possible ● Wrect and rect are covered by testing of a null variable list
testIterator	● Variables: ○ List length L ● Equivalence classes ○ L1: L=0 ○ L2: L>0 ● Thus, wect and sect are both covered by using an empty list and a nonempty ○ This is already done ● Wrect and Srect would involve L < 0 but this is not possible ● Wrect and Srect are covered by testing of a null variable list
testRemove	● Variables: ○ List length L ● Equivalence classes ○ L1: L=0 ○ L2: L>0 ● Thus, wect and sect are both covered by using an empty list and a nonempty ○ This is already done ● Wrect and Srect would involve L < 0 but this is not possible ● Wrect and rect are covered by testing of a null variable list
testRemoveAll	● Variables: ○ List length L ● Equivalence classes ○ L1: L=0

	○ L2: L>0 ● Thus, wect and sect are both covered by using an empty list and a nonempty ○ This is already done ● Wrect and Srect would involve L < 0 but this is not possible ● Wrect and rect are covered by testing of a null variable list
testRemoveByIndex	● Variables: ○ List length L ○ Index I ● Equivalence classes ○ L1:L=0 ○ L2:L>0 ○ I1:I<0 ○ I2:I>=0 && I<L ○ I3:I>=L ● Wect

ID	L	I
WECT1	L1	I1
WECT2	L2	I2
WECT3	L1	I3

● Sect

ID	L	I
SECT1	L1	I1
SECT2	L2	I1
SECT3	L1	I2
SECT4	L2	I2
SECT5	L1	I3
SECT6	L2	I3

- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are covered by WECT1, WECT3, SECT1, SECT2, SECT5, and SECT6
- Wrect and srect are also covered by testing of a null variable list

testRetainAll	● Variables: ○ List length L

  - List of objects O
- Equivalence classes
  - L1:L=0
  - L2:L>0
  - O1: All elements of O exist in list
  - O2: Some elements of O exist in list
  - O3: No elements of O exist in list
- Wect

ID	L	O
WECT1	L1	O1
WECT2	L2	O2
WECT3	L1	O3

- Sect

ID	L	I
SECT1	L1	O1
SECT2	L2	O1
SECT3	L1	O2
SECT4	L2	O2
SECT5	L1	O3
SECT6	L2	O3

  - Wrect and Srect would involve L < 0 but this is not possible
  - Wrect and srect are also covered by testing of a null variable list

testSet

- Variables:
  - List length L
  - Index I
  - Object O
- Equivalence classes
  - L1: L=0
  - L2: L>0
  - I1: I<0
  - I2: I>=0 && I<L
  - I3: I>=L
  - O1: O is not a duplicate
  - O2: O is a duplicate
- Wect

ID	L	I	O

WECT1	L1	I1	O1
WECT2	L2	I2	O2
WECT3	L1	I3	O1

- Sect

ID	L	I	O
SECT1	L1	I1	O1
SECT2	L2	I1	O1
SECT3	L1	I2	O1
SECT4	L2	I2	O1
SECT5	L1	I3	O1
SECT6	L2	I3	O1
SECT7	L1	I1	O2
SECT8	L2	I1	O2
SECT9	L1	I2	O2
SECT10	L2	I2	O2
SECT11	L1	I3	O2
SECT12	L2	I3	O2

- Wrect and Srect would involve L < 0 but this is not possible
- Wrect and srect are covered by testing of a null variable list

testSublist	• Variables: ○ List length L • Equivalence classes ○ L1: L=0 ○ L2: L>0 • Thus, wect and sect are both covered by using an empty list and a nonempty ○ This is already done • Wrect and Srect would involve L < 0 but this is not possible • Wrect and Srect are covered by testing of a null variable list
testSubListAddBegin	• Variables: ○ List length L • Equivalence classes

	○ L1: L=0 ○ L2: L>0 ● Thus, wect and sect are both covered by using an empty list and a nonempty ○ This is already done ● Wrect and Srect would involve L < 0 but this is not possible ● Wrect and Srect are covered by testing of a null variable list
testSubListAddEnd	● Variables: ○ List length L ● Equivalence classes ○ L1: L=0 ○ L2: L>0 ● Thus, wect and sect are both covered by using an empty list and a nonempty ○ This is already done ● Wrect and Srect would involve L < 0 but this is not possible ● Wrect and Srect are covered by testing of a null variable list
testSubListAddMiddle	● Variables: ○ List length L ● Equivalence classes ○ L1: L=0 ○ L2: L>0 ● Thus, wect and sect are both covered by using an empty list and a nonempty ○ This is already done ● Wrect and Srect would involve L < 0 but this is not possible ● Wrect and Srect are covered by testing of a null variable list
testSubListRemove	● Variables: ○ List length L ● Equivalence classes ○ L1: L=0 ○ L2: L>0 ● Thus, wect and sect are both covered by using an empty list and a nonempty ○ This is already done ● Wrect and Srect would involve L < 0 but this is not possible ● Wrect and Srect are covered by testing of a null variable list
testToArray	● Variables: ○ List length L ● Equivalence classes ○ L1: L=0 ○ L2: L>0 ● Thus, wect and sect are both covered by using an empty list and a nonempty ○ This is already done ● Wrect and Srect would involve L < 0 but this is not possible

	• Wrect and Srect are covered by testing of a null variable list

Test	Boundary Value Analysis
testSet	• Variables: ○ List length L ○ Index I ○ Object O • Test Set ○ BVL:{nom:L=0,nom:L>0} ○ BVI:{min-:I=-1,min:I=0,min+:I=1,nom:1<I<L, max-: I = L - 2, max: I = L - 1, max+: I = L} ○ BVO: {nom: O can be inserted, nom: O cannot be inserted} • BVA (basic)

ID	L	I	O
BVA (basic) 1	L>0	0	O can be inserted
BVA (basic) 2	L>0	1	O can be inserted
BVA (basic) 3	L>0	nom	O can be inserted
BVA (basic) 4	L>0	L-2	O can be inserted
BVA (basic) 5	L>0	L-1	O can be inserted

• BVA (robust)

ID	L	I	O
BVA (robust) 1	L>0	-1	O can be inserted
BVA (robust) 2	L>0	0	O can be inserted
BVA (robust) 3	L>0	1	O can be inserted
BVA (robust) 4	L>0	nom	O can be inserted

BVA (robust) 5	L>0	L-2	O can be inserted
BVA (robust) 6	L>0	L-1	O can be inserted
BVA (robust) 7	L>0	L	O can be inserted

testGet

- Variables:
  - List length L
  - Index I
- Test Set
  - BVL:{nom:L=0,nom:L>0}
  - BVI:{min-:I=-1,min:I=0,min+:I=1,nom:1<I<L, max-: I = L - 2, max: I = L - 1, max+: I = L}
- BVA (basic)

ID	L	I
BVA (basic) 1	L>0	0
BVA (basic) 2	L>0	1
BVA (basic) 3	L>0	nom
BVA (basic) 4	L>0	L-2
BVA (basic) 5	L > 0	L - 1

- BVA (robust)

ID	L	I
BVA (robust) 1	L>0	-1
BVA (robust) 2	L>0	0
BVA (robust) 3	L>0	1
BVA (robust) 4	L>0	nom
BVA (robust) 5	L>0	L-2
BVA (robust) 6	L>0	L-1
BVA (robust) 7	L>0	L

Test	Decision Table

testAdd

- Conditions
  - C1:L<=0?
  - C2: Object can be inserted?
- Actions
  - A1: Object is inserted
- Table

Test Case ID	C1	C2	Expected Output
TC1	Y/T	-	-
TC2	N/F	Y/T	Object inserted
TC3	N/F	N/F	No change

testContains

- Conditions
  - C1:L<=0?
  - C2: Object in list?
- Actions
  - A1: Object found in list
- Table

Test Case ID	C1	C2	Expected Output
TC1	Y/T	-	-
TC2	N/F	Y/T	True
TC3	N/F	N/F	False

testContainsAll

- Conditions
  - C1:L<=0?
  - C2: Object in list?
- Actions
  - A1: Object found in list
- Table

Test Case ID	C1	C2	Expected Output
TC1	Y/T	-	-
TC2	N/F	Y/T	True

TC3	N/F	N/F	False

testEquals

- Conditions
    - C1:L<=0?
    - C2: Objects are equal?
- Actions
    - A1: Objects are equal
- Table

Test Case ID	C1	C2	Expected Output
TC1	Y/T	-	-
TC2	N/F	Y/T	True
TC3	N/F	N/F	False

testGet

- Conditions
    - C1:L<=0?
    - C2: Index in range?
- Actions
    - A1: Object is gotten
- Table

Test Case ID	C1	C2	Expected Output
TC1	Y/T	-	-
TC2	N/F	Y/T	Object at index
TC3	N/F	N/F	Error

testIndexOf

- Conditions
    - C1:L<=0?
    - C2: Object in list?
- Actions
    - A1: Index is gotten
- Table

Test Case ID	C1	C2	Expected Output
TC1	Y/T	-	-
TC2	N/F	Y/T	Index of object
TC3	N/F	N/F	-1

testRemove

- Conditions
  - C1:L<=0?
  - C2: Object in list?
- Actions
  - A1: Object is removed
- Table

Test Case ID	C1	C2	Expected Output
TC1	Y/T	-	-
TC2	N/F	Y/T	Object removed
TC3	N/F	N/F	No change

testRemoveAll

- Conditions
  - C1:L<=0?
  - C2: Object in list?
- Actions
  - A1: Object is removed
- Table

Test Case ID	C1	C2	Expected Output
TC1	Y/T	-	-
TC2	N/F	Y/T	Objects removed
TC3	N/F	N/F	No change

testRetainAll

- Conditions
  - C1:L<=0?
  - C2: Objects exist to be retained?
- Actions
  - A1: Objects retained
- Table

Test Case ID	C1	C2	Expected Output
TC1	Y/T	-	-
TC2	N/F	Y/T	Objects retained
TC3	N/F	N/F	Empty List

testSet

- Conditions
    - C1:L<=0?
    - C2: Index in range?
    - C3: Object can be inserted
- Actions
    - A1: Object at index is set
- Table

Test Case ID	C1	C2	C3	Expected Output
TC1	Y/T	-	-	-
TC2	N/F	Y/T	Y/T	Object at index is set
TC3	N/F	Y/T	N/F	Error
TC4	N/F	N/F	Y/T	Error
TC5	N/F	N/F	N/F	Error

testSublist

- Conditions
    - C1:L<=0?
    - C2: Start index in range?
    - C3: End index in range?
- Actions
    - A1: Sublist is made
- Table

Test Case ID	C1	C2	C3	Expected Output
TC1	Y/T	-	-	-
TC2	N/F	Y/T	Y/T	Sublist
TC3	N/F	Y/T	N/F	Error
TC4	N/F	N/F	Y/T	Error
TC5	N/F	N/F	N/F	Error