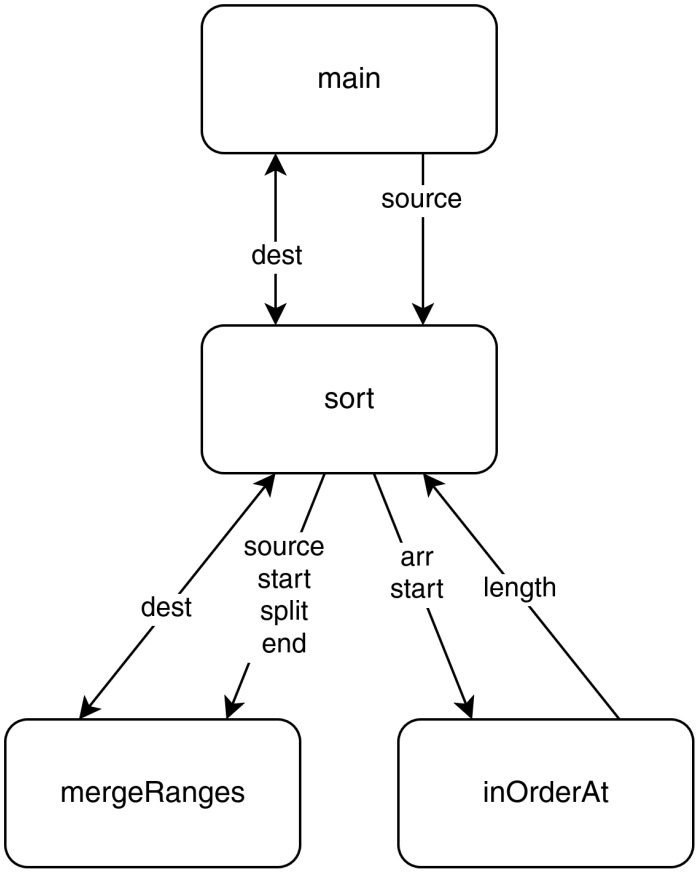
2025-11-08
Eric Petersen

LAB 08: SUB-LIST SORT ANALYSIS

Step 1: Modularization Metrics



main - The main function has **strong** cohesion by definition, as it performs one task completely and only that task. The main function has **trivial** coupling as no values are passed to or from main.

sort - The sort function has **strong** cohesion, as it sorts the input *source* list into the *dest* list, and performs no other actions. The sort function has **simple** coupling as the *source* and *dest* parameters are easy to select and and interpret, with the only requirement keeping this from being **encapsulated** being the requirement of matching lengths between the two array parameters.

mergeRanges - The mergeRanges function has **strong** cohesion, as it merges the two ranges denoted by [*start*, *split*) and [*split*, *end*) from *source* into *dest*. The mergeRanges function has **simple** coupling as each parameter is easy to supply and interpret, although each parameter value must require validation.

inOrderAt - The inOrderAt function has **strong** cohesion, as it returns the length of the in-order sublist at a certain index, and only performs that task - nothing else. The inOrderAt function has **simple** coupling as each parameter is easy to select and interpret, yet the *index* parameter must be validated to be less than or equal to the length of the *arr* parameter.

Step 2: Algorithmic Metrics

```
PROCEDURE mergeRanges(source, dest, start, split, end) // The mergeRanges function is O(N)
    // ASSERT 0 <= LENGTH(source) == LENGTH(dest)
    // ASSERT start <= split <= end <= LENGTH(source)

    IF split == start or split == end THEN // O(1)
        dest[start:end] <- source[start:end] // O(1) * O(N)
        RETURN // O(1)
    ENDIF

    left <- start // O(1)
    right <- split // O(1)
    index <- start // O(1)

    // While loop runs at most O(N) times. This is because one of left or right is incremented
    // each loop, and the loop terminates when left == split or right == end, which is at most
    // end - start iterations, which is N.
    WHILE left < split and right < end DO
        IF source[left] <= source[right] THEN // O(N) * O(1)
            dest[index] <- source[left] // O(N) * O(1)
            left <- left + 1 // O(N) * O(1)
        ELSE
            dest[index] <- source[right] // O(N) * O(1)
            right <- right + 1 // O(N) * O(1)
        ENDIF
        index <- index + 1 // O(N) * O(1)
    ENDWHILE

    WHILE right < end DO // O(N) as right is incremented until == end, which is <= N
        dest[index] <- source[right] // O(N) * O(1)
        index <- index + 1 // O(N) * O(1)
        right <- right + 1 // O(N) * O(1)
    ENDWHILE

    WHILE left < split DO // O(N) as left is incremented until == split, which is <= N
        dest[index] <- source[left] // O(N) * O(1)
        index <- index + 1 // O(N) * O(1)
        left <- left + 1 // O(N) * O(1)
    ENDWHILE
ENDPROCEDURE
```

```
PROCEDURE inOrderAt(arr, start) // inOrderAt is O(N)
    // ASSERT 0 <= start <= LENGTH(arr)

    index <- start + 1 // O(1)
    WHILE index < LENGTH(arr) DO // While loop is O(N) as index is repeatedly incremented until N
        IF arr[index] < arr[index - 1] THEN // O(1)
            RETURN index - start // O(N) * O(1)
        ENDIF
        index <- index + 1 // O(N) * O(1)
    ENDWHILE
    RETURN LENGTH(arr) - start // O(1)
ENDPROCEDURE
```

```
PROCEDURE sort(source, dest) // sort is O(N * LOG(N))
    // ASSERT LENGTH(source) == LENGTH(dest)

    WHILE TRUE DO // At most O(Log(N)) overall because each iteration cuts the problem in half
        IF inOrderAt(source, 0) == LENGTH(source) THEN // O(N) because of inOrderAt
            dest[:] <- source // O(Log(N)) * O(N)
            RETURN // O(Log(N)) * O(1)
        ENDIF

        start <- 0 // O(Log(N)) * O(1)
        // While loop is overall O(N) bcause O(A) * O(B) is O(N)
        WHILE start < LENGTH(source) DO // O(Log(N)) * O(B) because start is set to end until N
            split <- start + inOrderAt(source, start) // O(Log(N)) * O(B) * O(A) because inOrderAt
            end <- split + inOrderAt(source, split) // O(Log(N)) * O(B) * O(A) because inOrderAt
            mergeRanges(source, dest, start, split, end) // O(Log(N)) * O(B) * O(A)
            start <- end // O(Log(N)) * O(B) * O(1)
        ENDWHILE

        source <- dest // O(Log(N)) * O(1)
    ENDWHILE
ENDPROCEDURE
```

Step 3. Test

Testing Array:

Function	Test Type	Provided Input	Expected Output
mergeRanges	normal	([1,2,3,4,5], [0,0,0,0,0], 0, 0, 3)	dest is now [1,2,3,0,0]
mergeRanges	normal	([5,4,3,2,1], [0,0,0,0,0], 0, 1, 2)	dest is now [4,5,0,0,0]
mergeRanges	normal	([3,4,5,1,2], [0,0,0,0,0], 0, 3, 5)	dest is now [1,2,3,4,5]
mergeRanges	error	([1,2,3,4,5], [], 0, 0, 0)	assertion error
mergeRanges	error	([1,2,3,4,5], [0,0,0,0,0], 0, 0, 100)	assertion error
mergeRanges	error	([], [0,0,0,0,0], 0, 0, 0)	assertion error
mergeRanges	boundary	([1,2,3,4,5], [0,0,0,0,0], 0, 0, 0)	dest is now [0,0,0,0,0]
mergeRanges	boundary	([], [], 0, 0, 0)	dest is now []
mergeRanges	boundary	([1], [0], 0, 0, 1)	dest is now [1]
inOrderAt	normal	([1,2,3,4,5], 0)	5
inOrderAt	normal	([1,2,3,4,5], 3)	2
inOrderAt	normal	([1,2,5,0,0], 0)	3
inOrderAt	error	([1,2,3,4,5], -10)	assertion error
inOrderAt	error	([1,2,3,4,5], 10)	assertion error
inOrderAt	boundary	([1,2,3,4,5], 5)	0
inOrderAt	boundary	([], 0)	0
sort	normal	([1,2,3,4,5], [0,0,0,0,0])	dest is [1,2,3,4,5]
sort	normal	([5,4,3,2,1], [0,0,0,0,0])	dest is [1,2,3,4,5]
sort	normal	([5,1,4,3,2], [0,0,0,0,0])	dest is [1,2,3,4,5]
sort	error	([1,2,3,4,5], [])	assertion error
sort	error	([], [0, 0, 0, 0, 0])	assertion error
sort	boundary	([], [])	dest is []
sort	boundary	([0], [0])	dest is [0]
sort	boundary	([5,5,5,5,5], [0,0,0,0,0])	dest is [5,5,5,5,5]

Automation Drivers:

```
def test_mergeRanges() -> None:
    tests = [
        ([1,2,3,4,5], [0,0,0,0,0], 0, 0,   3,         [1,2,3,0,0]),
        ([5,4,3,2,1], [0,0,0,0,0], 0, 1,   2,         [4,5,0,0,0]),
        ([3,4,5,1,2], [0,0,0,0,0], 0, 3,   5,         [1,2,3,4,5]),
        ([1,2,3,4,5],          [], 0, 0,   0,   "assertion error"),
        ([1,2,3,4,5], [0,0,0,0,0], 0, 0, 100,   "assertion error"),
        (          [], [0,0,0,0,0], 0, 0,   0,   "assertion error"),
        ([1,2,3,4,5], [0,0,0,0,0], 0, 0,   0,         [0,0,0,0,0]),
        (          [],          [], 0, 0,   0,                 []),
        (         [1],         [0], 0, 0,   1,                [1]),
    ]

    for test in tests:
        source, dest, start, split, end, expected = test
        try:
            mergeRanges(source, dest, start, split, end)
            assert dest == expected
        except AssertionError:
            assert expected == "assertion error"

def test_inOrderAt() -> None:
    tests = [
        ([1,2,3,4,5],   0,                5),
        ([1,2,3,4,5],   3,                2),
```

```
        ([1,2,5,0,0],   0,                3),
        ([1,2,3,4,5], -10, "assertion error"),
        ([1,2,3,4,5],  10, "assertion error"),
        ([1,2,3,4,5],   5,                0),
        (         [],   0,                0),
    ]

    for test in tests:
        arr, start, expected = test
        try:
            result = inOrderAt(arr, start)
            assert result == expected
        except AssertionError:
            assert expected == "assertion error"

def test_sort() -> None:
    tests = [
        ([1,2,3,4,5], [0,0,0,0,0],       [1,2,3,4,5]),
        ([5,4,3,2,1], [0,0,0,0,0],       [1,2,3,4,5]),
        ([5,1,4,3,2], [0,0,0,0,0],       [1,2,3,4,5]),
        ([1,2,3,4,5],          [], "assertion error"),
        (         [], [0,0,0,0,0], "assertion error"),
        (         [],          [],                []),
        (        [0],         [0],               [0]),
        ([5,5,5,5,5], [0,0,0,0,0],       [5,5,5,5,5]),
    ]

    for test in tests:
        source, dest, expected = test
        try:
            sort(source, dest)
            assert dest == expected
        except AssertionError:
            assert expected == "assertion error"

def runTests() -> None:
    test_inOrderAt()
    test_mergeRanges()
    test_sort()

if __name__ == "__main__":
    runTests()
```