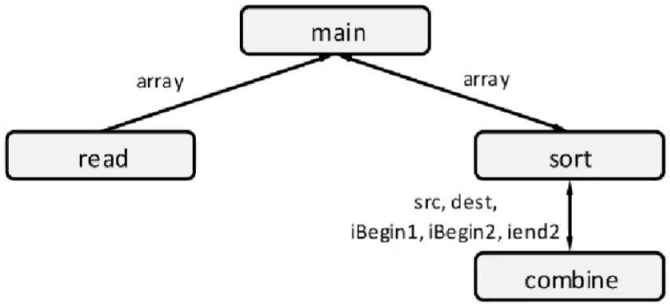
Braeden Pope
Modularization Design
Sub-list Sort Analysis

Modularization Metrics



Function	Cohesion	Coupling
read	Strong. This function only reads the information from a give file.	Simple. Is performed only once and very little information is exchanged or validated.
sort	Extraneous. It not only retrieves the two subarrays for sorting, but also keeps track of the indices of the source array.	Simple. The information exchange is clear and straightforward, giving an unsorted array and returns a sorted array.
combine	Strong. Only combines the two subarrays onto the destination array, therefore having a single functionality.	Interactive. It consistently is called by the sort function, with a constant exchange of the array and indices.

Algorithmic Metrics

```
    sort(array)
1       size ← array.length
2       src ← array
3       des ← new array of same size as source
4       num ← 2

5       WHILE num > 1
6           num ← 0
7           begin1 ← 0

8           WHILE begin1 < size
9               end1 ← begin1 + 1
10              WHILE end1 < size AND src[end1 - 1] ≤ src[end1]
11                  end1 ++

12              begin2 ← end1
13              IF begin2 < size
14                  end2 ← begin2 + 1
15              ELSE
16                  end2 ← begin2
17              WHILE end2 < size AND src[end2 - 1] ≤ src[end2]
18                  end2 ++

19              num ++
20              combine(src, des, begin1, begin2, end2)
21              begin1 ← end2
22          SWAP src and des pointers

23      RETURN src
```

Algorithmic Efficiency

The Algorithmic Efficiency of this algorithm is O(n log n). This is the efficiency because it takes longer according to the size of the input array, but each loop of sorting becomes shorter as more of the array becomes sorted, and the subarrays become larger.

Program Trace

Line/Variable	src	des	begin1	begin2	end1	end2	num	size
1	//	//	//	//	//	//	//	3
2	[31,72,32]	[0,0,0]	//	//	//	//	//	3
3	[31,72,32]	[0,0,0]	//	//	//	//	//	3
4	[31,72,32]	[0,0,0]	//	//	//	//	2	3
5	[31,72,32]	[0,0,0]	//	//	//	//	2	3
6	[31,72,32]	[0,0,0]	//	//	//	//	0	3
7	[31,72,32]	[0,0,0]	0	//	//	//	0	3
8	[31,72,32]	[0,0,0]	0	//	//	//	0	3
9	[31,72,32]	[0,0,0]	0	//	1	//	0	3
10	[31,72,32]	[0,0,0]	0	//	1	//	0	3
11	[31,72,32]	[0,0,0]	0	//	2	//	0	3
12	[31,72,32]	[0,0,0]	0	2	2	//	0	3
13	[31,72,32]	[0,0,0]	0	2	2	//	0	3
14	[31,72,32]	[0,0,0]	0	2	2	3	0	3
17	[31,72,32]	[0,0,0]	0	2	2	3	0	3
19	[31,72,32]	[0,0,0]	0	2	2	3	1	3
20	[31,72,32]	[31,32,72]	0	2	2	3	1	3
21	[31,72,32]	[31,32,72]	0	2	2	3	1	3
22	[31,72,32]	[31,32,72]	0	2	2	3	1	3
23	[31,72,32]	[31,32,72]	0	2	2	3	1	3