Karina Zhang

The time complexity of CheckReverse is **O(n)**. For each element i that is less than the length of String one, a block of code which has a time complexity of O(1) will run. O(1) being executed a variable number of times is O(n).

The time complexity of ShortestOfStrings is **O(n)**. For each element i that is less than words.length - 2, a block of code which has a time complexity of O(1) will run. O(1) being executed a variable number of times is O(n).

Sorting Algorithm	**Worst case**	**Reasoning**	**Best case**	**Reasoning**
Bubble Sort	O(n^2)	Contains a nested for-loop.	O(n^2)	Even if presorted, it will still run the nested for-loop.
Bubble Sort (Recursive)	O(n^2)	We call bubbleSort for every element of the array, which is O(n), and within each call, we have a for-loop. Therefore, runtime is O(n*n) = O(n^2)	O(n^2)	Even if presorted, it will still call bubbleSort for every element of the array and run the for-loop located inside every call.
Selection Sort	O(n^2)	Contains a nested for-loop.	O(n^2)	Even if presorted, it will still run the nested for-loop.
Insertion Sort	O(n^2)	Contains a while-loop inside a for-loop.	O(n)	If presorted, it will never start the while-loop contained within the for-loop. Only the for-loop will run.
Merge Sort	O(nlogn)	Performs binary search which has a time complexity of O(logn) since you divide the problem space by two every time. The function "merge" runs at O(n) because it contains a while loop. Every "mergeSort" call calls "merge" inside of it, meaning Merge Sort as a whole has O(nlogn) running time.	O(nlogn)	Even if presorted, binary search will be performed which always runs in O(nlogn) and merge will also always be performed which always runs in O(n).

Bubble Sort Non Recursive

i = 0

j = 0	4	77	98	30	20	50	77	22	49	2
j = 1	4	77	98	30	20	50	77	22	49	2
j = 2	4	77	98	30	20	50	77	22	49	2
j = 3	4	77	30	98	20	50	77	22	49	2
j = 4	4	77	30	20	98	50	77	22	49	2
j = 5	4	77	30	20	50	98	77	22	49	2
j = 6	4	77	30	20	50	77	98	22	49	2
j = 7	4	77	30	20	50	77	22	98	49	2
j = 8	4	77	30	20	50	77	22	49	98	2
j = 9	4	77	30	20	50	77	22	49	2	98

i = 1

j = 0	4	77	30	20	50	77	22	49	2	98
j = 1	4	77	30	20	50	77	22	49	2	98
j = 2	4	30	77	20	50	77	22	49	2	98
j = 3	4	30	20	77	50	77	22	49	2	98
j = 4	4	30	20	50	77	77	22	49	2	98
j = 5	4	30	20	50	77	77	22	49	2	98
j = 6	4	30	20	50	77	22	77	49	2	98
j = 7	4	30	20	50	77	22	49	77	2	98
j = 8	4	30	20	50	77	22	49	2	77	98

i = 2

j = 0	4	30	20	50	77	22	49	2	77	98
j = 1	4	30	20	50	77	22	49	2	77	98
j = 2	4	20	30	50	77	22	49	2	77	98
j = 3	4	20	30	50	77	22	49	2	77	98
j = 4	4	20	30	50	77	22	49	2	77	98
j = 5	4	20	30	50	22	77	49	2	77	98
j = 6	4	20	30	50	22	49	77	2	77	98
j = 7	4	20	30	50	22	49	2	77	77	98

i = 3

j = 0	4	20	30	50	22	49	2	77	77	98
j = 1	4	20	30	50	22	49	2	77	77	98
j = 2	4	20	30	50	22	49	2	77	77	98
j = 3	4	20	30	50	22	49	2	77	77	98
j = 4	4	20	30	22	50	49	2	77	77	98
j = 5	4	20	30	22	49	50	2	77	77	98
j = 6	4	20	30	22	49	2	50	77	77	98

i = 4

j = 0	4	20	30	22	49	2	50	77	77	98
j = 1	4	20	30	22	49	2	50	77	77	98
j = 2	4	20	30	22	49	2	50	77	77	98
j = 3	4	20	22	30	49	2	50	77	77	98
j = 4	4	20	22	30	49	2	50	77	77	98
j = 5	4	20	22	30	2	49	50	77	77	98

i = 5

j = 0	4	20	22	30	2	49	50	77	77	98
j = 1	4	20	22	30	2	49	50	77	77	98
j = 2	4	20	22	30	2	49	50	77	77	98
j = 3	4	20	22	30	2	49	50	77	77	98
j = 4	4	20	22	2	30	49	50	77	77	98

i = 6

j = 0	4	20	22	2	30	49	50	77	77	98
j = 1	4	20	22	2	30	49	50	77	77	98
j = 2	4	20	22	2	30	49	50	77	77	98
j = 3	4	20	2	22	30	49	50	77	77	98

i = 7

j = 0	4	20	2	22	30	49	50	77	77	98
j = 1	4	20	2	22	30	49	50	77	77	98
j = 2	4	2	20	22	30	49	50	77	77	98

i = 8

j = 0	4	2	20	22	30	49	50	77	77	98
j = 1	2	4	20	22	30	49	50	77	77	98

Bubble Sort Recursive

arr = {4, 77, 98, 30, 20, 50, 77, 22, 49, 2}
n = 10

bubbleSort(arr, 10)	compares every adjacent pair when i < 9 (see i = 0 section of Bubble Sort Non Recursive)
return bubbleSort(arr, 9)	compares every adjacent pair when i < 8 (see i = 1 section of Bubble Sort Non Recursive)
return bubbleSort(arr, 8)	compares every adjacent pair when i < 7 (see i = 2 section of Bubble Sort Non Recursive)
return bubbleSort(arr, 7)	compares every adjacent pair when i < 6 (see i = 3 section of Bubble Sort Non Recursive)
return bubbleSort(arr, 6)	compares every adjacent pair when i < 5 (see i = 4 section of Bubble Sort Non Recursive)
return bubbleSort(arr, 5)	compares every adjacent pair when i < 4 (see i = 5 section of Bubble Sort Non Recursive)
return bubbleSort(arr, 4)	compares every adjacent pair when i < 3 (see i = 6 section of Bubble Sort Non Recursive)
return bubbleSort(arr, 3)	compares every adjacent pair when i < 2 (see i = 7 section of Bubble Sort Non Recursive)
return bubbleSort(arr, 2)	compares every adjacent pair when i < 1 (see i = 8 section of Bubble Sort Non Recursive)
return bubbleSort(arr, 1)	return arr

Selection Sort

i = 0

j = 1	**4**	**77**	98	30	20	50	77	22	49	2	
j = 2	**4**	77	**98**	30	20	50	77	22	49	2	
j = 3	**4**	77	98	**30**	20	50	77	22	49	2	
j = 4	**4**	77	98	30	**20**	50	77	22	49	2	
j = 5	**4**	77	98	30	20	**50**	77	22	49	2	
j = 6	**4**	77	98	30	20	50	**77**	22	49	2	
j = 7	**4**	77	98	30	20	50	77	**22**	49	2	
j = 8	**4**	77	98	30	20	50	77	22	**49**	2	
j = 9	**4**	77	98	30	20	50	77	22	49	**2**	swap!
	2	77	98	30	20	50	77	22	49	4	

i = 1

j = 1	2	**77**	**98**	30	20	50	77	22	49	4	
j = 2	2	**77**	98	**30**	20	50	77	22	49	4	swap!
j = 3	2	**30**	98	77	**20**	50	77	22	49	4	swap!
j = 4	2	**20**	98	77	30	**50**	77	22	49	4	
j = 5	2	**20**	98	77	30	50	**77**	22	49	4	
j = 6	2	**20**	98	77	30	50	77	**22**	49	4	
j = 7	2	**20**	98	77	30	50	77	22	**49**	4	
j = 8	2	**20**	98	77	30	50	77	22	49	**4**	swap!
	2	**4**	98	77	30	50	77	22	49	20	

i = 2

j = 1	2	4	**98**	**77**	30	50	77	22	49	20	swap!
j = 2	2	4	**77**	98	**30**	50	77	22	49	4	swap!
j = 3	2	4	**30**	98	77	**50**	77	22	49	4	
j = 4	2	4	**30**	98	77	50	**77**	22	49	4	
j = 5	2	4	**30**	98	77	50	77	**22**	49	4	swap!
j = 6	2	4	**22**	98	77	50	77	30	**49**	4	
j = 7	2	4	**22**	98	77	50	77	30	49	**20**	swap!
	2	4	**20**	98	77	50	77	30	49	22	

i = 3

j = 1	2	4	20	**98**	**77**	50	77	30	49	22	swap!
j = 2	2	4	20	**77**	98	**50**	77	30	49	22	swap!
j = 3	2	4	20	**50**	98	77	**77**	30	49	22	
j = 4	2	4	20	**50**	98	77	77	**30**	49	22	
j = 5	2	4	20	**30**	98	77	77	50	**49**	22	
j = 6	2	4	20	**30**	98	77	77	50	49	**22**	swap!
	2	4	20	**22**	98	77	77	50	49	30	

i = 4

j = 1	2	4	20	22	**98**	**77**	77	50	49	30	swap!
j = 2	2	4	20	22	**77**	98	**77**	50	49	30	
j = 3	2	4	20	22	**77**	98	77	**50**	49	30	swap!
j = 4	2	4	20	22	**50**	98	77	77	**49**	30	swap!
j = 5	2	4	20	22	**49**	98	77	77	50	**30**	swap!
	2	4	20	22	**30**	98	77	77	50	49	

i = 5

j = 1	2	4	20	22	30	**98**	**77**	77	50	49	swap!
j = 2	2	4	20	22	30	**77**	98	**77**	50	49	
j = 3	2	4	20	22	30	**77**	98	77	**50**	49	swap!
j = 4	2	4	20	22	30	**50**	98	77	77	**49**	swap!
	2	4	20	22	30	**49**	98	77	77	50	

i = 6

j = 1	2	4	20	22	30	49	**98**	**77**	77	50	swap!
j = 2	2	4	20	22	30	49	**77**	98	**77**	50	
j = 3	2	4	20	22	30	49	**77**	98	77	**50**	swap!
	2	4	20	22	30	49	**50**	98	77	77	

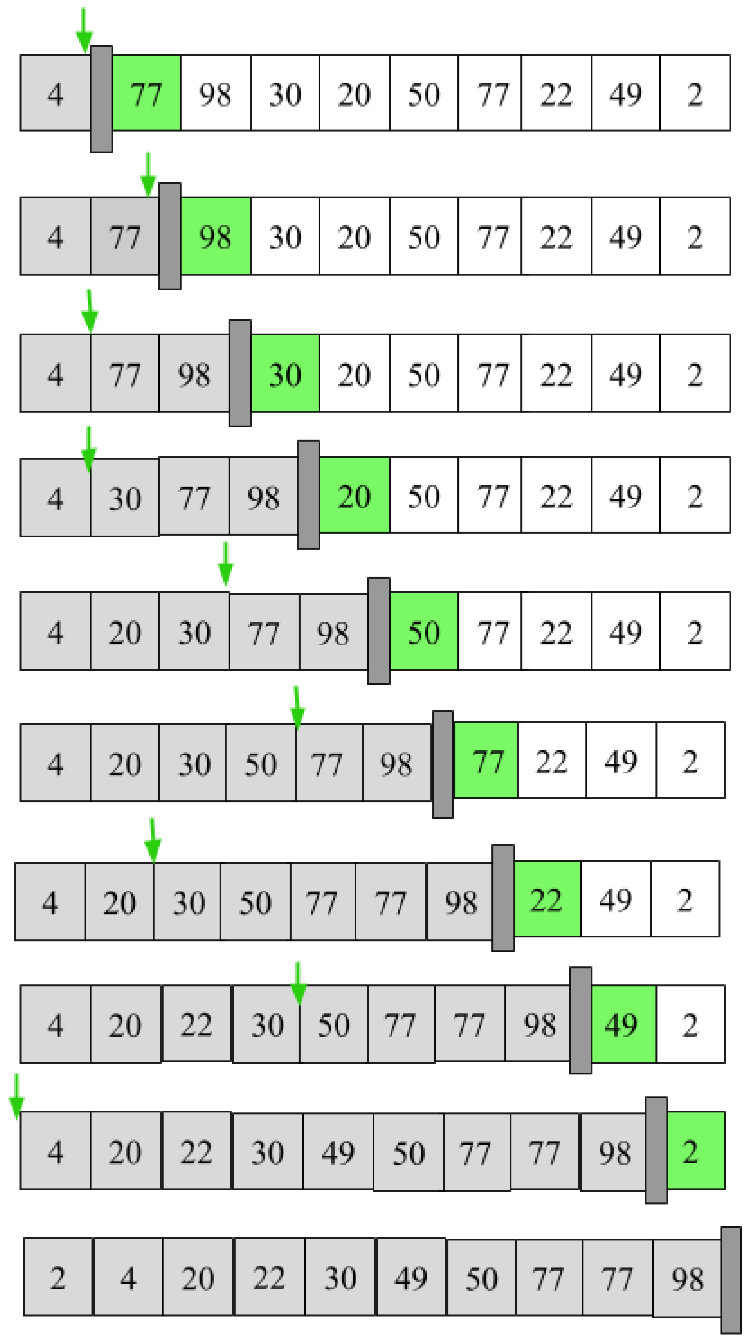
i = 7

j = 1	2	4	20	22	30	49	50	**98**	**77**	77	swap!
j = 2	2	4	20	22	30	49	50	**77**	98	**77**	
	2	4	20	22	30	49	50	**77**	98	77	

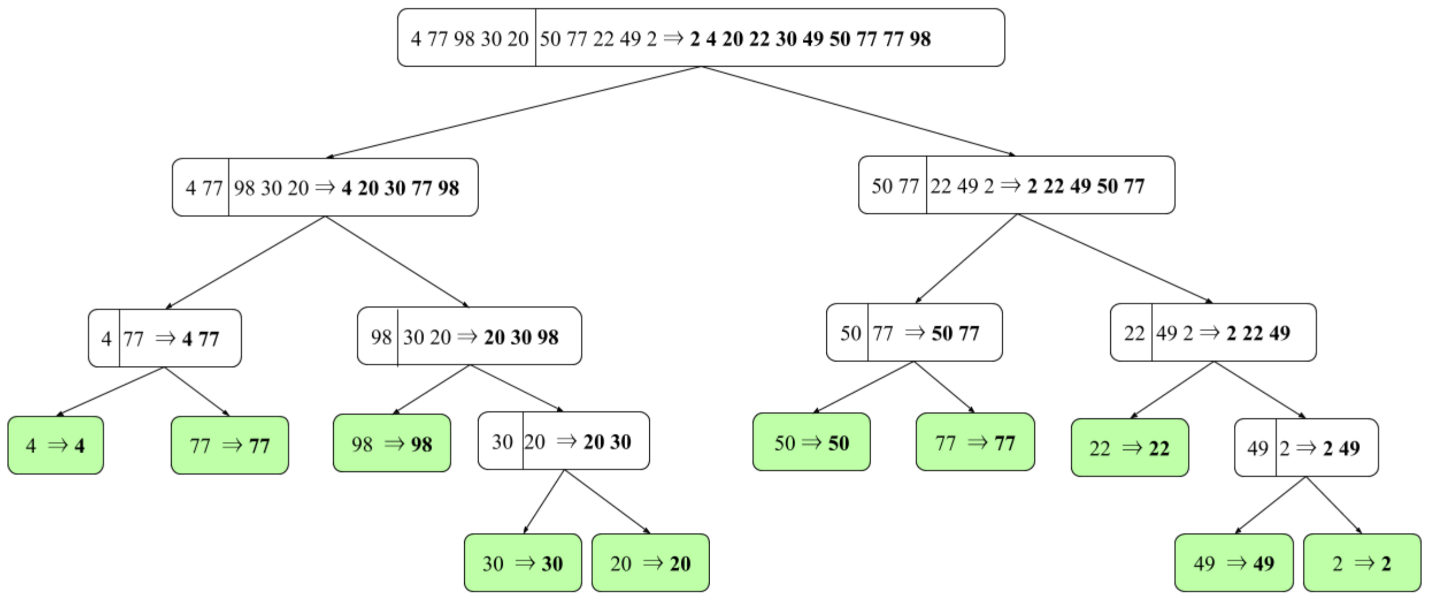
i = 8

j = 1	2	4	20	22	30	49	50	77	**98**	**77**	swap!
	2	4	20	22	30	49	50	77	**77**	98	

Insertion Sort



Merge Sort


4 77 98 30 20 | 50 77 22 49 2 ⇒ 2 4 20 22 30 49 50 77 77 98
4 77 | 98 30 20 ⇒ 4 20 30 77 98
50 77 | 22 49 2 ⇒ 2 22 49 50 77
4 | 77 ⇒ 4 77
98 | 30 20 ⇒ 20 30 98
50 | 77 ⇒ 50 77
22 | 49 2 ⇒ 2 22 49
4 ⇒ 4
77 ⇒ 77
98 ⇒ 98
30 | 20 ⇒ 20 30
50 ⇒ 50
77 ⇒ 77
22 ⇒ 22
49 | 2 ⇒ 2 49
30 ⇒ 30
20 ⇒ 20
49 ⇒ 49
2 ⇒ 2