Dmtiri Stanchevici
Unit 3
Module 5

Ex. 5.6

Printed is: [I@1dbd16a6

Ex. 5.10

Without typing and executing the problem, I evaluated that the program prints out:
5
7
4

Ex. 5.11

The program should print out
[1, 2, 3, 4, 5, 6, 4, 5, 6]
[1, 2, 3, 4, 5, 6, 4, 5, 6]

Ex. 5.20

Add pseudocode for bottom-out cases

Algorithm: mergeSort (data)

Input: an array of length n called data

    // The first call to the recursive method is with the whole array:

1. mergeSortRecursive (data, 0, n-1)

Algorithm: mergeSortRecursive (data, L, E)

Input: data array, with a specified range from L to E

    // Base cases

if L == E OR L == E-1
return;
// Compute where the middle element lies in the array:

1. middle = (L+E) / 2;

// Recursively sort the left half.

2. mergeSortRecursive (data, L, middle)

// Recursively sort the right half

3. mergeSortRecursive (data, middle+1, E)

// Now merge the two subarrays:

4. mergeRange (data, L, middle+1, E)

Ex. 5.22

Trace mergeSortRecursive for [51, 24, 63, 73, 42, 85, 71, 41, 87, 32].

Base cases:
if L == E return
OR
if L == E-1, sort/swap if necessary and then return;

DATA					
[51, 24, 63, 73, 42, 85, 71, 41, 87, 32]	Recursion 0 (enter from main) L = 0 E = 9 middle = 4 left mergeSort (data, L, middle)				
		Recursion 1 L = 0 E = 4 middle = 2 left mergeSort (data, L, middle)			

			Recursion 2 L = 0 E = 2 middle = 1 left mergeSort (data, L, middle)		
				Recursion 3 L = 0 E = 1 base case L == E-1 Return	
			Back in Recursion 2 L = 0 E = 2 middle = 1 left mergeSort (data, L, middle) right mergeSort (data, middle+1, E)		
				Recursion 4 L = 2 E = 2 base case L == E Return	
[24, 51, 63, 73, 42, 85, 71, 41, 87, 32]			Back in Recursion 2 L = 0 E = 2 middle = 1 left mergeSort (data, L, middle) right mergeSort (data, middle+1, E) mergeRange (data, L=0, middle+1=2, E=2)		
		Back in Recursion 1 L = 0 E = 4 middle = 2 left mergeSort (data, L, middle) right mergeSort (data, middle+1, E)			
			Recursion 5 L = 3 E = 4		

			base case L == E-1 return		
[24, 42, 51, 63, 73, 85, 71, 41, 87, 32]		Back in Recursion 1 L = 0 E = 4 middle = 2 left mergeSort (data, L, middle) right mergeSort (data, middle+1, E) mergeRange (data, L=0, middle+1=3, E=4) Return			
	Back in Recursion 0 (enter from main) L = 0 E = 9 middle = 4 left mergeSort (data, L, middle) right mergeSort (data, middle+1, E)				
		Recursion 6 L = 5 E = 9 middle = 7 left mergeSort (data, L, middle)			
			Recursion 7 L = 5 E = 7 middle = 6 left mergeSort (data, L, middle)		
				Recursion 8 L = 5 E = 6 base case L == E-1 return	
			Back in Recursion 7 L = 5		

			E = 7 middle = 6 left mergeSort (data, L, middle) right mergeSort (data, middle+1, E)		
				Recursion 9 L = 7 E = 7 base case L == E return	
[24, 42, 51, 63, 73, 41, 71, 85, 87, 32]			Back in Recursion 7 L = 5 E = 7 middle = 6 left mergeSort (data, L, middle) right mergeSort (data, middle+1, E) mergeRange (data, L=5, middle+1=7, E=7)		
		Back in Recursion 6 L = 5 E = 9 middle = 7 left mergeSort (data, L, middle) right mergeSort (data, middle+1, E)			
			Recursion 10 L = 8 E = 9 base case return		
[24, 42, 51, 63, 73, 32, 41, 71, 85, 87]		Back in Recursion 6 L = 5 E = 9 middle = 7 left mergeSort (data, L, middle) right mergeSort (data, middle+1, E)			

		mergeRange (data, L=5, middle+1=8, E=9)			
[24, 32, 41, 42, 51, 63, 71, 73, 85, 87]	Back in Recursion 0 (enter from main) L = 0 E = 9 middle = 4 left mergeSort (data, L, middle) right mergeSort (data, middle+1, E) mergeRange (data, L=0, middle+1=5, E=9)				