Time and Space Complexity Analysis

Samuel Navia Quiceno (A00405006)
Simón García (A00371828)
Juan Camilo Criollo Cifuentes(A00402515)

Algorithm 1:

Algorithm	Time Complexity	Space Complexity
Queue<Order> tempQueue = new Queue<>();	1	n
boolean customerFound = false;	1	1
while (!customerOrders.isEmpty()) {	n +1	
Order currentOrder = customerOrders.front();	n	1
if (currentOrder.getCostumer().equals(newOrder.getCostumer())) {	n	
customerFound = true; } end-if	m	1
currentOrder = customerOrders.dequeue();	n	1
tempQueue.enqueue(currentOrder);	n	1
if (customerFound && !customerOrders.isEmpty() && !customerOrders.front().getCostumer().equals(newOrder.getCostumer()))	n	
tempQueue.enqueue(newOrder);	1	1
customerFound = false;	1	1
} //Here while ends		
if (customerFound) {	1	
tempQueue.enqueue(newOrder); //Ends if	1	1
if(customerOrders.isEmpty())	1	
tempQueue.enqueue(newOrder); //Ends if	1	1
customerOrders = tempQueue;	1	n
Total	6n + m + 10 = 7n + 10 *	9+2n

Worst Case	O(n)	O(n)

*In time complexity, n = number of nodes in customerOrders, m = number of orders of the customer which value is added. In the worst case, the entire length of orders is from one customer so n = m

*In space complexity, n = size of queue of costumberOrders

Algorithm 2:

Algorithm	Time Complexity	Space Complexity
String message;	1	1
if(!actionHistory.isEmpty()){	1	
String lastAction = actionHistory.top();	1	1
actionHistory.pop();	1	
String[] lastActionSplit = lastAction.split("-");	7	1
if(lastActionSplit[0].equals("customer")){	1	
customers.delete(lastActionSplit[1]);	1	
message = "The last created customer was deleted.";	1	1
} else if(lastActionSplit[0].equals("product")){	1	
if(lastActionSplit.length == 2){	1	1
products.delete(lastActionSplit[1]);	1	
message = "The last created product was deleted.\nEven though discontinued, the orders with this product will still be dispatched.";	1	1
} else if(lastActionSplit.length == 7){	1	1
if(!lastActionSplit[1].equals("delete")){	1	
products.delete(lastActionSplit[2]); //END IF	1	
String code = lastActionSplit[1].equals("update") ? lastActionSplit[2] : lastActionSplit[1];	1	1
String name= lastActionSplit[3];	1	1
String description = lastActionSplit[4];	1	1

double price = Double.parseDouble(lastActionSplit[5].replace(",",".")); 	1	1
int priority = Integer.parseInt(lastActionSplit[6]); 	1	1
Product product = new Product(code, name, description, price, priority); 	1	1
products.add(code, product); 	1	
if(lastActionSplit[1].equals("priority")){ 	1	
List<Order> reorderOrders = orders.getValues(); 	n	n
for(Order order: reorderOrders){ 	n+1	
order.reHeapOrders(); // END FOR && END IF 	n*(n+1)	
message = "The last updated product was restored to it's original values."; 	1	1
} else { message = "Unexpected error product action split length."; 	1	1
} else if(lastActionSplit[0].equals("order")){ 	1	
orders.delete(lastActionSplit[1]); 	1	
customerOrders.dequeue(); 	1	
message = "The last added order has been dequeued."; 	1	1

} else {
message = "Unexpected error customer/product/order.";
}
 	
	1	1

} else {
message = "No actions are left to undo.";
}
 	
	1	1

return message; 	1	
Total	38+3n+n^2	18+n

Worst Case	$O(n^2)$	O(n)

*In time complexity, n = number of nodes in Order,
*In space complexity, n = size of the List Orders