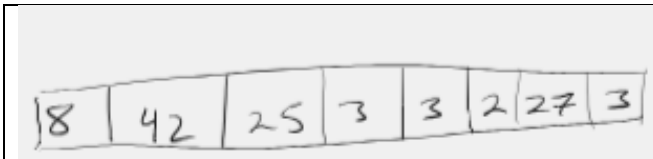
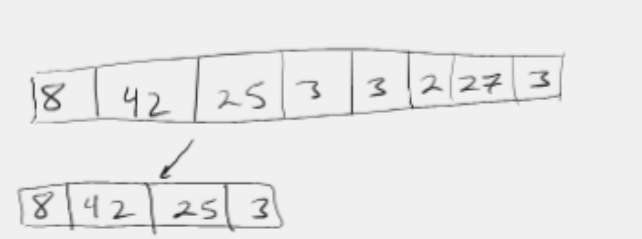
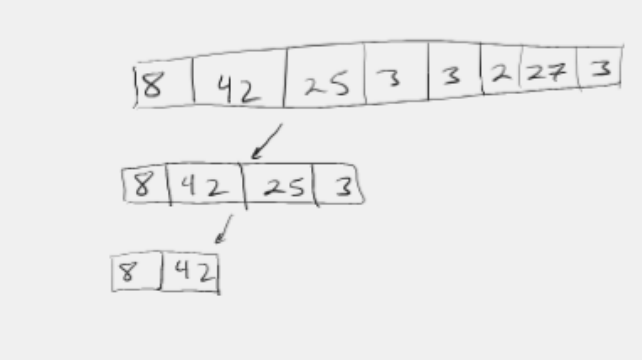
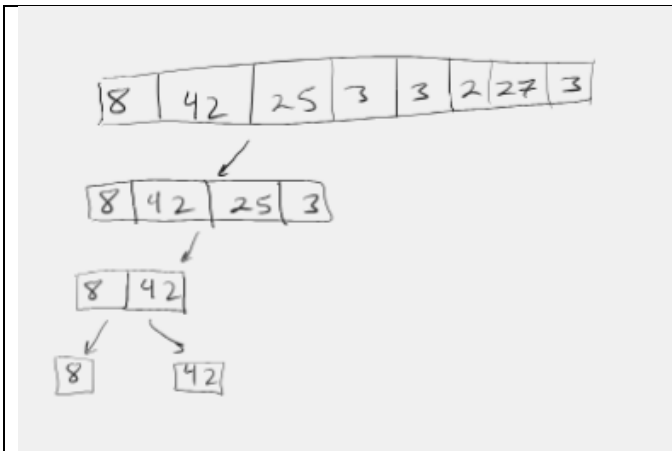
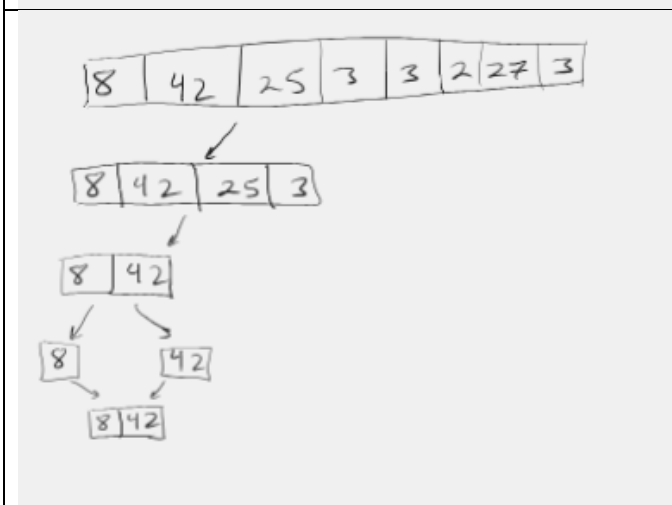
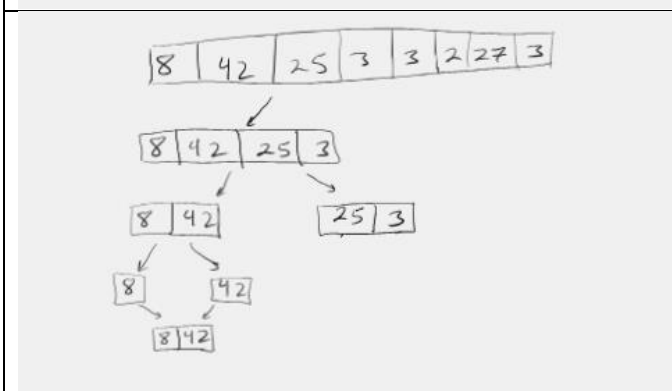
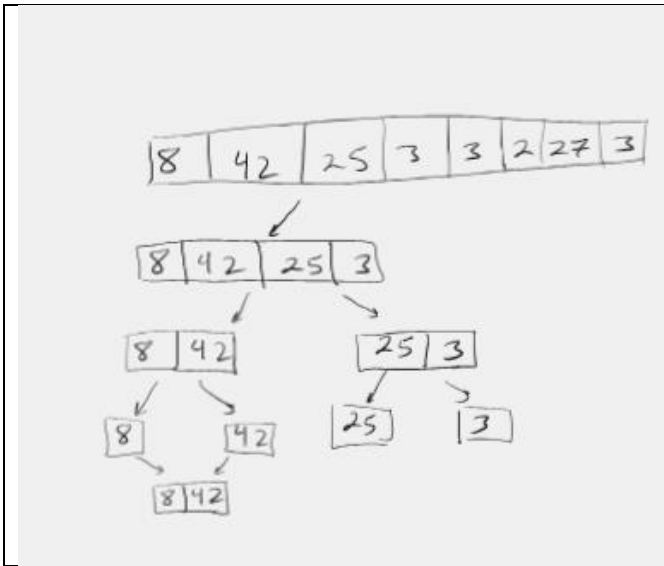
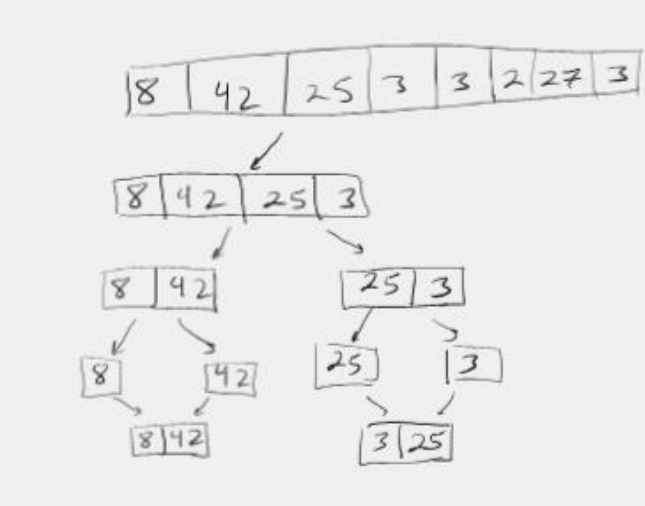
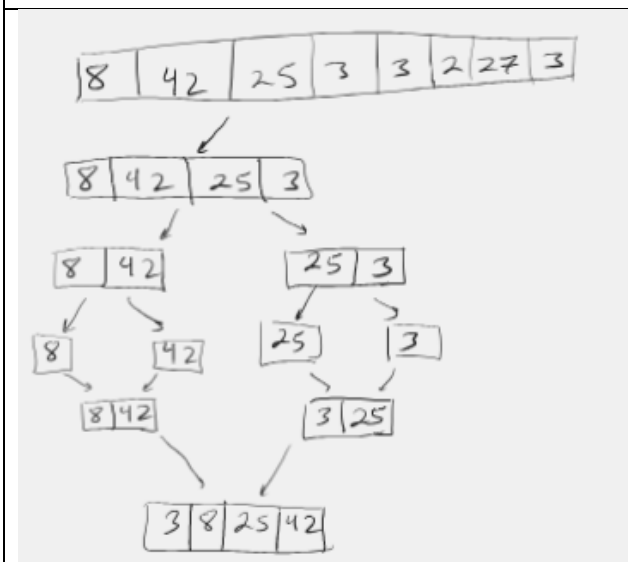
Question 2

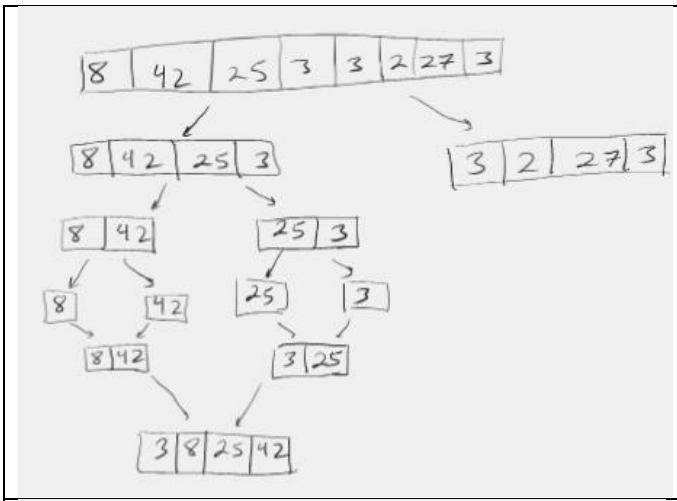
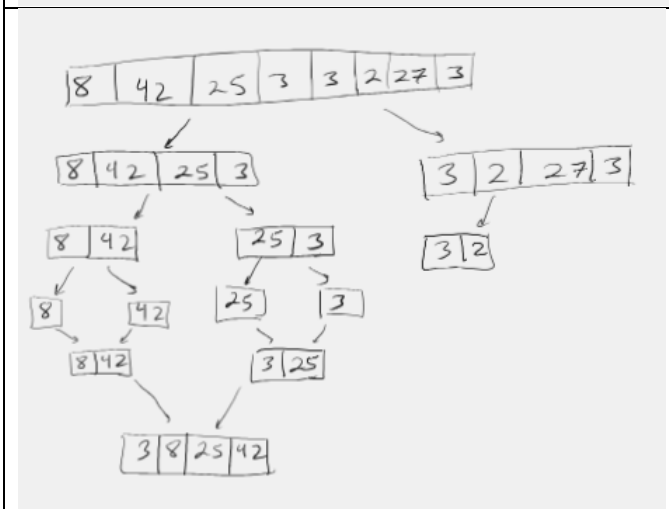
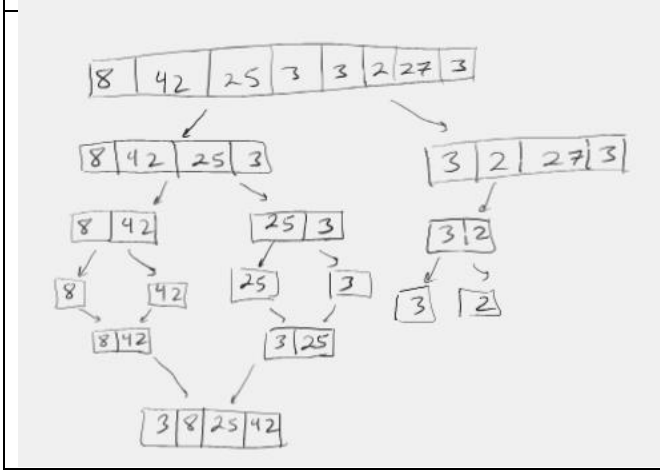
The worst-case complexity of O(n log n) for the merge algorithm. The algorithms starts with the merge sort function which just divides the code into halves regardless of the size of the array. So the complexity of the merge sort would be O(1) since it takes a constant time. However since the function is called recursively that would result into the steps takes as log(n) where in this case the n would be the size of the array. The merge function has the complexity of O(n) since it iterates over the array once to see if the low and high indexes have sorted values. Furthermore the left and right array has an O(n) complexity since it just iterates over the array once and pastes the values based on the indexes. This means that the overall time complexity of the algorithm is O(n log n) since it take O(n) time for the merge function to be called and it takes O(log n) time for the recursion to be called . And since the algorithm runs regardless of the input array given, this means that this complexity is also the worst-case complexity.

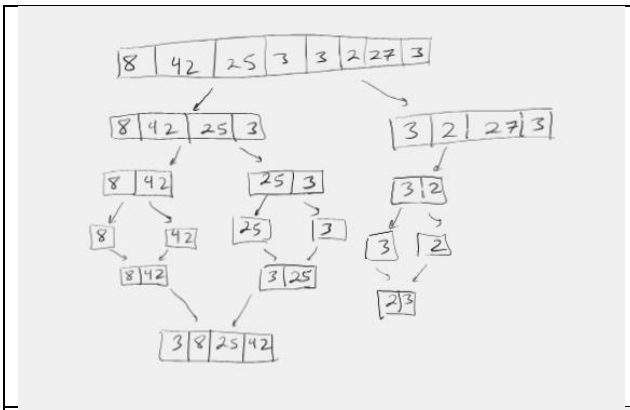
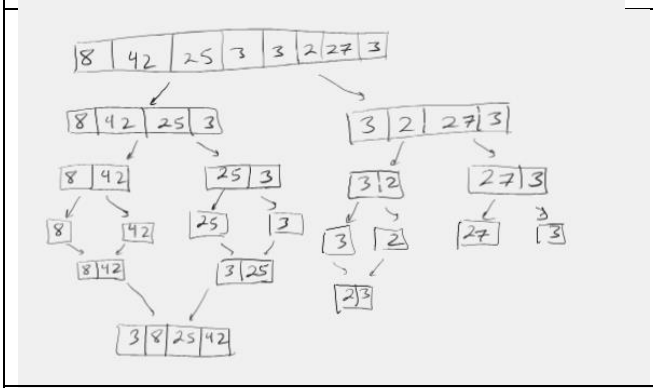
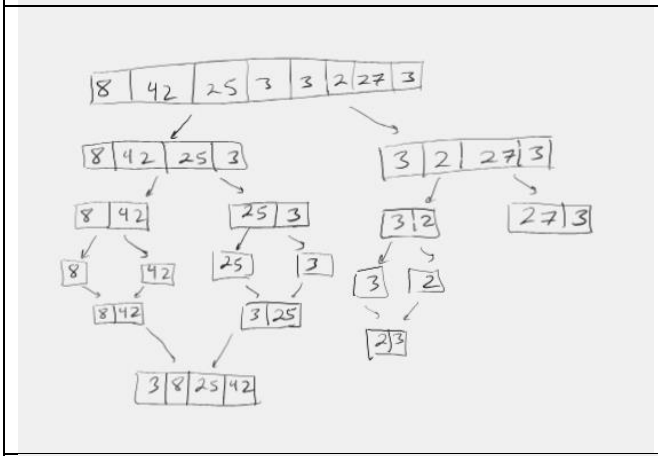
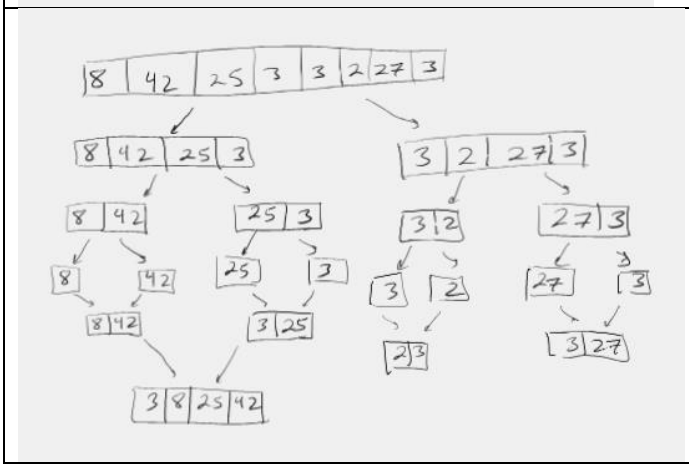
Question 3

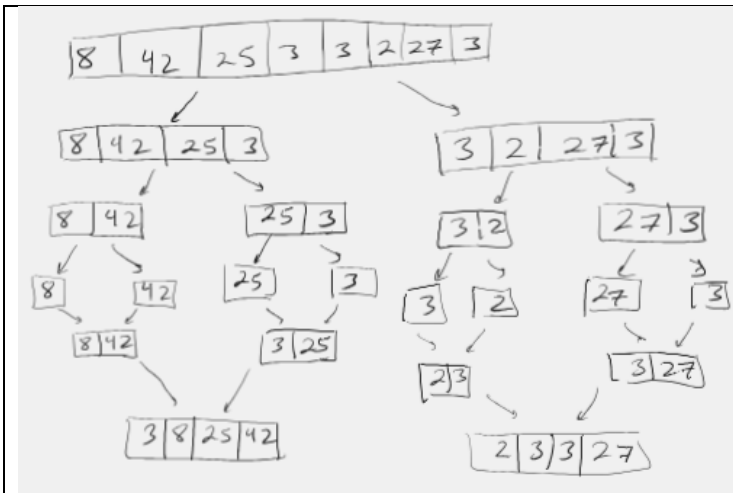
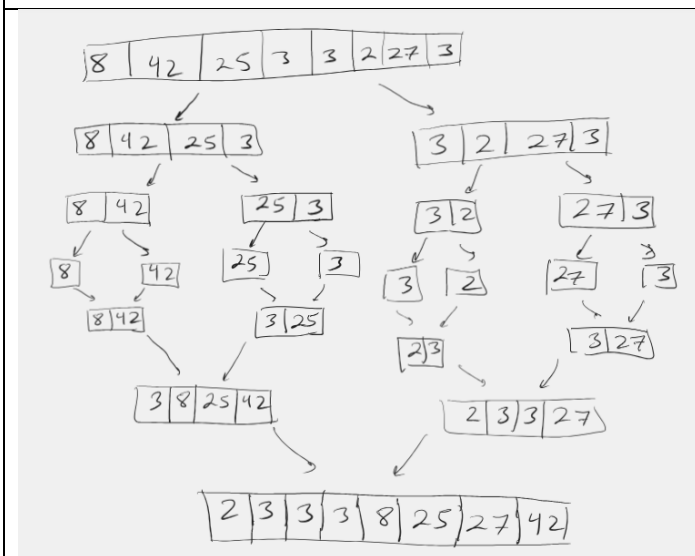
	Initial array of elements.
	Array is divided into half is and taken aside. This is where the split part of the split and merge algorithm.
	Array is split into half.

		Array is split into 2 halves where each array consist of an element.
		Now the merge function is being called. It will take the two arrays and go through the array on the left . If it is less than the element on the right it will then it will be placed before the element. In this case 8 is less than 42, so it will be placed first.
		The other half of the array is now split from the array.

	Array is split into 2 halves where each array consist of an element.
	Now the merge function is being called. It will take the two arrays and go through the array on the left . If it is less than the element on the right it will then it will be placed before the element. In this case 25 is greater than 3. So 25 will be after 3. However since the element are not the first 2 then the index of 3 and 25 are in the range they are given. So 3 is in array[2] and 25 is array[3].
	Now the 2 split of the array will merge together. The left array will be compared to the right array. If an element in the right array is greater than an element in the left side then the value of the index will be replaced. For instance 3 is lower than 8 so array[0] is now 3 instead of 8. And since the array won't be read at the same pace. When the function reaches the end of one of the list, the function will just add the all the elements of the other list right after where the index stopped.

	Second half of the array is split.
	And the split occurs again.
	Array is split into 2 halves where each array consist of an element.

	Function merges the 2 arrays, and finds 2 is less than 3.
	The second split of the array is being iterated through.
	Function merges the 2 arrays, and finds 2 is less than 3.
	Function merges the 2 arrays, and finds 3 is less than 27.

	Now the 2 split of the array will merge together. The left array will be compared to the right array. If an element in the right array is greater than an element in the left side then the value of the index will be replaced. However, in this case 3 is repeated twice so the function will take the 3 from the left array. When the function reaches the end of one of the list, the function will just add the all the elements of the other list right after where the index stopped.
	The 2 big split of the array will merge. The left array will be compared to the right array. If an element in the right array is greater than an element in the left side then the value of the index will be replaced. When the function reaches the end of one of the list, the function will just add the all the elements of the other list right after where the index stopped. In this case 2 is less than 3 so 2 will be place in the fist index.

Question 4

Yes the number of steps sems to be consistent with my complexity since the function is called recursively, so that the array is split into halves, that results into the steps takes as log(n) where in this case the n would be the size of the array. The merge function has the complexity of O(n) since it iterates over the array once to see if the low and high indexes have sorted values which happen when the merge function is called.