Exercise 2 – Sorting Algorithms (41 Marks)

This is a simple program that provides two functions that, when given an integer list, return the list sorted into ascending order. The first function uses a bubble sort algorithm to sort the given list, while the second function uses a merge sort algorithm.

A program designed to test these functions is shown below (and is provided electronically). Study the code and try to understand what is happening in the program, before attempting the questions that follow.

```
1  def bubbleSort(sortList):
2      sorted = False
3      length = len(sortList)
4      while !sorted:
5          for i in range(length - 2):
6              if sortList[i] > sortList[i+1]:
7                  sortList[i] = sortList[i+1]
8                  sortList[i+1] = sortList[i]
9                  sorted = False
10     return sortList
-
12 def mergeSort(sortList):
13     mid = len(sortList) // 2
14     leftHalf = sortList[:mid]
15     rightHalf = sortList[mid:]
16     if len(sortList) > 1:
18         mergeSort(leftHalf)
19         mergeSort(rightHalf)
-
21 numList = []
22 for i in range(6):
23     print("Add an integer number to the list: ")
24     numList.append(int(input()))
25 print("Bubble sort given:")
26 print(numList)
27 print("Bubble sort returns")
28 print(bubbleSort(numList))
29 input()
```

Section A

A 1 Give a line number from the program that contains a parameter. **[1]**

.....

A 2 Give a line number from the program that contains recursion. **[1]**

.....

A 3 Define 'recursion'. **[1]**

.....

.....

A 4 The program does not run and produces a syntax error.
Explain the cause of this syntax error. **[1]**

.....

.....

A 5 When the `bubbleSort` function is called, the program gets stuck in an infinite loop.
Explain the cause of this logic error. **[1]**

.....

.....

A 6 Instead of swapping elements that are in the wrong order, the `bubbleSort` function copies the value of the second element in the pair to the first element in the pair.
State the type of error this is, and explain the cause of the error. **[2]**

.....

.....

.....

A 7 Currently, the program crashes if the user enters a non-integer value when prompted to add a number to the list. This could be prevented by implementing exception handling.
Explain what exception handling is and why it is necessary. **[2]**

.....

.....

.....

.....

A 8 The merge sort algorithm is an example of a divide-and-conquer algorithm.
Explain what a 'divide-and-conquer' algorithm is. **[2]**

.....

.....

.....

.....

A 9 State the time complexity of the bubble sort and merge sort algorithms using Big O notation. **[2]**

.....

.....

.....

.....

A 10 As well as taking different amounts of time, different algorithms also require different amounts of space in memory. Explain which of the two sorting algorithms has the larger space complexity. **[2]**

.....

.....

.....

.....

Section A: /15

SECTION B

B 1 Modify the program to remove the syntax error. **[1]**

Program updated ☐

B 2 Modify the program so that the `bubbleSort` function does not get stuck in an infinite loop. **[1]**

Program updated ☐

B 3 Modify the program so that `bubbleSort` correctly swaps elements that are in the wrong order. **[1]**

Program updated ☐

B 4 Modify the program so that the program does not crash if the user enters a non-integer value when prompted to add a number to the list. Your solution should display a message to the user telling them when they have entered a non-integer number and keep asking the user for input until they give a valid integer. The input should terminate once they enter a blank (just press return). **[4]**

Program updated ☐

B 5 Currently, the code that asks the user to enter the numbers in the list is hard-coded into the main program procedure, and so cannot be easily reused.

Modify the program so that this code is moved into a new `getList` function that gets the user's input and returns the resulting list. `getList` should be called in the main program procedure. The function should stop asking for input when a 'blank' number is entered (i.e. the user just presses return). **[4]**

Program updated ☐

B 6 Modify the `getList` function so that the user can enter any number of integers between square brackets separated by commas (e.g. [1, 4, 2, 17, 14, 12]) to give their entire list at once instead of each number individually. The user should still have the option to enter numbers individually if they choose, again ending with a blank. **[3]**

Program updated ☐

B 7 The `bubbleSort` function checks every element of the list for each pass it makes through the list – even the elements that it knows have been correctly sorted in the previous passes.

Modify the `bubbleSort` function so that, after each pass, the number of elements that are checked is reduced so that elements that will not need to be swapped again are not checked again. **[1]**

Program updated ☐

B 8 The `mergeSort` function is currently incomplete.

Complete the `mergeSort` function so that it performs a full merge sort on a given list and returns the sorted list. The `mergeSort` function should be tested in the main program procedure instead of the `bubbleSort` function. **[4]**

Program updated ☐

B 9 Modify the program to compare the time efficiency of the `bubbleSort` and `mergeSort` functions. The `bubbleSort` and `mergeSort` functions should be modified to include a `swaps` variable that counts the number of swaps that are made, and returns `swaps` once the list is sorted.

A `test` function should be added that takes an integer value, `n`, for the length of list, and displays the number of swaps made by `bubbleSort` in comparison to `mergeSort` when they sort a list of randomly generated integers (between 1 and 100) of length `n`. The main program procedure should be modified to call `test` three times, using 10, 100 and 1,000 as the values of `n`. **[7]**

Program updated ☐

Section B: /26

