QUESTION 1. [10 MARKS]

Suppose your (ancient) computer can store 2 million non-negative whole numbers as 8-bit binary numbers, without any need for a "sign" bit.

PART (A) [2 MARKS]

How many DIFFERENT non-negative whole numbers can your machine store? Explain your answer

SOLUTION: Each non-negative whole number from 00000000 through 1111111 (0 through 127, in decimal) can be stored, or 128 different numbers.

PART (B) [2 MARKS]

Give an example of a non-negative whole number that could not be stored on your machine. Explain why.

SOLUTION 100000000 (256 in base 10), since this requires 9 bits.

PART (C) [2 MARKS]

Give an example of some arithmetic operations that takes numbers that can be stored on your machine, but produces a number that cannot be stored on your machine. Explain.

SOLUTION: 128 + 128. In binary, 128 is 10000000, and hence can fit into 8 bits of storage. However, 128 + 128 = 256, which we've just shown cannot be stored on my machine.

PART (D) [4 MARKS]

Write 6 and 7 as binary numbers, and multiply these binary numbers. Show your work.

SOLUTION: 6 is 110 in binary, and 7 is 111 in binary. To multiply them:

```
   111
 x 110
 -----
   000
  111
 111
------
101010
```

QUESTION 2. [26 MARKS]

Explain as much as you can of what's going on in the following Python sessions.

PART (A) [3 MARKS]

```
>>> 11/3 > 9/3
False
```

SOLUTION: 11/3 is 3 (integer division discards the remainder), and so is 9/3, and so 11/3 is not ¿ 9/3, so the statement is False.

PART (B) [2 MARKS]

```
>>> 'okkee' in 'bookkeeper'
True
```

SOLUTION: String 'okkee' is in string 'bookkeeper', so the statement is True.

PART (C) [6 MARKS]

```
>>> number1 = 5
>>> string1 = 'five'
>>> number1 = string1
>>> string1 = number1
>>> type(number1)
<type 'str'>
>>> type(string1)
<type 'str'>
```

SOLUTION: The number 5 is stored at *number1*. The string 'five' is stored at *string1*. The object at *string1* (string 'five') is stored at *number1*. The object at *number1* (string 'five') is stored at *string1*. The object at *number1* (string 'five') is tested to see its type, which is string. The object at *string1* (string 'five') is tested to see its type, which is string.

PART (D) [6 MARKS]

```
>>> numList = [2,3,5,7,11]
>>> wordList = ['two', 'three', 'five', 'seven', 'eleven']
>>> numList[1] = wordList[2]
>>> wordList[2] = numList[2]
>>> numList
[2, 'five', 5, 7, 11]
>>> wordList
['two', 'three', 5, 'seven', 'eleven']
```

SOLUTION: A list of numbers is stored at the location labelled *numList*. A list of strings is stored at the location labelled *wordList*. Entry 2 (the string 'five') of *wordList* is stored at position 1 of *numList* (replacing the number 3). Entry 2 of *numList* (the number 5) is stored at position 2 of *wordList* (replacing the string 'five'). *numList* is displayed, and then *wordList* is displayed.

PART (E) [9 MARKS]

```
>>> list1 = ['basil', 'olive oil', 'pine nuts', 'garlic']
>>> list2 = ['lentils', 'salt', 'tumeric', 'water']
>>> recipe = {'pesto': list1, 'dal' : list2}
>>> recipe['pesto'].append(recipe['dal'][3])
>>> recipe['pesto']
['basil', 'olive oil', 'pine nuts', 'garlic', 'water']
```

SOLUTION: The list *['basil', 'olive oil', 'pine nuts', 'garlic']* is stored at *list1*. The list *['lentils', 'salt', 'tumeric', 'water']* is stored at *list2*. *list1* is stored at key 'pesto' and *list2* is stored at 'dal' in look-up table *recipe*. The entry at position 3 of the entry at 'dal' in lookup table *recipe* (the string 'water') is appended to the entry at position 'pesto' of look-up table *recipe*. The entry at 'pesto' of look-up table *recipe* is displayed, and it has a new entry.

QUESTION 3. [7 MARKS]

PART (A) [4 MARKS]

What is the maximum number of regions you can divide a sheet of paper into with 7 straight lines? Explain your answer (a formula doesn't count as an explanation).

SOLUTION: One horizontal line can divide the page into two regions (the region above and the region below the line). Each line we add after that can be drawn so that it meets each of the previous lines at a different point. That means that each line that is added to the diagram creates a new region for each of the previous lines it meets, from left to right, (by slicing an old region in two), plus one region after it leaves the last of the previous lines and proceeds to the edge of the paper. That makes, 2 regions for one line, 2+2 regions for two lines, 2+2+3 regions for three lines, 2+2+3+4 regions for four lines, 2+2+3+4+5 regions for five lines, 2+2+3+4+5+6 regions for six lines, and 2+2+3+4+5+6+7 = 29 regions for seven lines.

PART (B) [3 MARKS]

Suppose that you correctly filled in the code for *lazyLit.py* from Assignment 2, but inadvertently replaced the contents of the file *alice30.txt* with a list of all the different surnames in the Toronto phonebook. The names occur in alphabetical order, and there are no duplicate names. What is the result of the following Python session? Explain why.

```
>>> import lazyLit
>>> bookTable = {}
>>> lazyLit.buildTable('alice30.txt', 3, bookTable)
>>> lazyLit.useTable(100, 3, bookTable)
```

SOLUTION: The output is the first 100 surnames from the list. Since each surname occurs just once, each prefix of 3 surnames occurs just once, and there is only one possible name to follow each prefix. This removes any random choice.

Total Marks = 43