COMP10001 Foundations of Computing
Semester 2, 2016

Tutorial Questions: Week 12

Recall that "hexadecimal" numbers are base 16 numbers, with the digits $\{0, 1, 2, ..., 9, A, B, ..., F\}$, with A, e.g., representing the decimal value 10. The hexadecimal number $A0$ thus represents the decimal number $10 \times 16^1 + 0 \times 16^0 = 160$. Recall also that "binary" numbers are base 2 numbers, with the digits $\{0, 1\}$, and that the binary number 11001, e.g., represents the decimal number $1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 25$. To convert a binary number to a hexadecimal number, it is simplest to take each 4-bit sequence (representing a decimal number between 0 and 15) and translate it directly into a single hexadecimal digit (you might like to ponder over why we can do this), and simply concatenate the hexadecimal digits together. E.g., in the case of 11001, it "segments" into the 4-digit binary numbers 1 and 1001, which translate into the hexadecimal digits 1 and 9 respectively, meaning its hexadecimal equivalent is 19.

In Python, you flag a number as binary by prefixing it with `0b` (e.g. `0b11001`), and flag a number as hexadecimal by prefixing it with `0x` (e.g. `0x19`). For example:

```
>>> 0x19 == 0b11001 == 25
True
```

You can also convert an integer into a binary number in string form (prefixed with `'0b'`) using the command `bin()`:

```
>>> bin(0xcc)
'0b11001100'
```

and similarly, you can convert an integer into a hexadecimal number in string form (prefixed with `'0x'`) using the command `hex()`:

```
>>> hex(78)
'0x4e'
```

1. For the problem of sorting a list of integers, design: (a) a test to determine whether a list is in sort order, and (b) a "brute-force" algorithm to solve the problem. What is the best- and worst-case "runtime efficiency" of your algorithm?

2. For the problem of sorting a list of integers, design a "divide-and-conquer" algorithm to solve the problem (e.g. by applying the same basic logic that was used in the binary search algorithm). What is the best- and worst-case "runtime efficiency" of your algorithm?

3. What character does the byte represented by `0xE0` translate into in the following character encodings:

- ISO-8859-1
- ISO-8859-11
- UTF-8

4. Write a function `conv(infile, infile_enc, outfile, outfile_enc)` that reads in `infile` in character encoding `infile_enc`, and writes it out to `outfile` in character encoding `outfile_enc`.

5. Which of the following bit sequences (presented as hexadecimal numbers) represent valid UTF-8 strings, and in the case they are valid UTF-8 strings, how many code points does the bit sequence correspond to?

   - `0x30c0`
   - `0x303C`
   - `0xE0ADAA`
   - `0x3AA`

6. Complete the following code to write a function which checks whether a given hexadecimal number input is valid UTF-8 or not:

```
def is_valid_utf8(val):
    bits = []
    start = True
    for digit in hex(val)[2:]:
        four_bits = bin(int(digit,16))[2:].zfill(4)
        if start:
            bits.append(four_bits)
            start = False
        else:
            bits[-1] += four_bits
            start = True
```