COMP10001 Foundations of Computing
Semester 2, 2016

Tutorial Questions: Week 12

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?

   A: *One test would be to iterate over all elements in the list, and check that the desired inequality (e.g. $\leq$) holds for each adjacent pairing of items.*
   One possible solution (there are many) to the "generate" part of the problem would be to exhaustively generate all permutations of the elements of the list. The best-case efficiency of the method would be achieved in the case that the first permutation was sorted, in which case the efficiency would simply be the length of the list (n); the worst-case efficiency would be achieved in the case that the last permutation generated was sorted, which would take $n \times n!$ to compute (where n is the length of the original list) – a very, very long time for a reasonably-sized list!

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?

   A: *There are various ways of doing this. One would be to iteratively build up a sorted sub-list, using binary search to insert one element at a time into the position i in the sub-list such that the value of the element lies between the values of the ith and $i+1$th elements. This algorithm is popularly called "insertion sort".*
   The best-case efficiency of the algorithm is n, in the case that the mid-point of the sub-list is the insertion point for all elements (you might like to think about what the sort-order of the original list would like to achieve this); the worst-case efficiency is roughly equivalent to $n \times \log_2(n)$, i.e. the case of each of the n elements taking the worst-case time for binary search ($= \log_2(n)$) to insert, which occurs when binary search gets down to a one-element sub-list before it can finally insert the element.

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

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

   A:

   - *á*
   - *(the Thai vowel Mai na)*
   - *á*

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`.

   A:

```
def conv(infile, infile_enc, outfile, outfile_enc):
    open(outfile, "w", encoding=outfile_enc).write(open(infile, encoding=infile_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`

A:

```
0x30c0: False
0x303c: True (2)
0xe0adaa: True (1)
0x3aa: False
```

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
```

A:

```python
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
    trailing = 0
    for byte in bits:
        if len(byte) != 8:
            return False
        if byte[0] == '0':
            if trailing:
                return False
        elif byte[:2] == '10' and trailing:
            trailing -= 1
        elif byte[:3] == '110' and not trailing:
            trailing = 1
        elif byte[:4] == '1110' and not trailing:
            trailing = 2
        elif byte[:5] == '11110' and not trailing:
            trailing = 3
        else:
            return False
    if trailing:
        return False
    return True
```