recommended background capabilities
general aptitude: recognising patterns, and interpolation and extrapolation

I0. Computation

1. In sequential programming, "`IF`" lets us specify code that may or may not run, depending on whether a condition is true. For example,

```
IF (counter < 5) THEN
 PRINT "The count is less than five."
END IF
```

will only print "`The count is less than five.`" if the "`counter`" variable does indeed have a value less than 5. We can use logical operators like "`AND`", "`OR`", and "`NOT`" to combine the conditions.

```
IF (counter < 5) AND (target > 10) THEN
 PRINT "Counter is less than five, and target is more than ten."
END IF
```

The code above will only print the text shown, if both of the conditions are true.

We can use "truth tables" to represent the way that logical operators combine conditions. The truth tables for `NOT`, `OR` and `XOR` are shown below. `XOR` means "exclusive `OR`" (which we can think of as "either-or"), so notice how it differs from the conventional "inclusive `OR`" ("any of the conditions").

p	NOT *p*
true	false
false	true

p	*q*	*p* OR *q*
true	true	true
true	false	true
false	true	true
false	false	false

p	*q*	*p* XOR *q*
true	true	false
true	false	true
false	true	true
false	false	false

1. Fill in the truth table for the `AND` operator (you may abbreviate "true" as T, and "false" as F).

p	*q*	*p* AND *q*
true	true	
true	false	
false	true	
false	false	

Confidence: 0 | 1 | 2 | 3 | 4

2. Consider the following code, and determine whether it behaves as it "should".

```
IF (counter > 3) OR (counter < 5) THEN
 PRINT "The count is between 3 and 5."
END IF
```

When will the above code print "`The count is between 3 and 5.`"?

- ☐ Every time.
- ☐ When `counter` has a value between 3 and 5.
- ☐ When `counter` has a value outside 3 and 5.
- ☐ Never.
- ☐ The code cannot be compiled or interpreted.

Confidence: 0 | 1 | 2 | 3 | 4

2. We can work with binary (base 2) numbers in a similar way to decimal (base 10) numbers.

Decimal: $4729_{10} = (4 \times 10 \times 10 \times 10) + (7 \times 10 \times 10) + (2 \times 10) + 9$

Binary: $1101_2 = (1 \times 2 \times 2 \times 2) + (1 \times 2 \times 2) + (0 \times 2) + 1$

Hence the binary number $1101_2 = 8 + 4 + 1 = 13_{10}$ (converted to base 10).

1. Convert the binary number 10101_2 to decimal.

Confidence: 0 | 1 | 2 | 3 | 4

2. Convert the decimal number 22 to binary.

Confidence: 0 | 1 | 2 | 3 | 4

I1. Sequential Programming

1. The FOR statement in C has the following syntax.

```
for (start_expression; test_expression; update_expression) {
  // comment: internal code here
}
```

This FOR loop behaves as follows.

- Evaluate `start_expression` on starting the loop.
- Evaluate `test_expression` before every iteration, and terminate the loop if it is false.
- Execute the internal code.
- Evaluate `update_expression` after every iteration.

1. Consider the following code, which generates a countdown sequence.

```
for (n = 10; n > 0; n--) {
  printf("%i\n", n);     // print number n on a new line
}
```

The start expression `n = 10` sets the value of `n` to 10.

The test expression `n > 0` is true if `n` is more than 0, and false otherwise.

The update expression `n--` decreases the value of `n` by 1.

What sequence of numbers does this code generate as output?

Confidence: 0 | 1 | 2 | 3 | 4

2. How does the behaviour of the following code differ from the example above?

```
for (n = 10; n > 0; /* comment: do nothing here */) {
  n--;
  printf("%i\n", n);
}
```

Confidence: 0 | 1 | 2 | 3 | 4

2. Higher-level programming languages allow us to define “functions” (or subroutines), which are blocks of code that we can “call” as shorthand, instead of having to make a copy of the same code every time we want to run it.

A function can receive “arguments”, information that we “pass” to it (usually in parentheses after the function name). We can use this to easily perform the same computations on different information.

Recursion occurs if a function calls itself. For example, the following JavaScript function will count down by calling itself with successively lower values of `counter` until it reaches 0.

```
function countdown(counter) {
  if (counter > 0) {
    countdown(counter - 1);
  }
}
```

1. The following recursive JavaScript function calls itself twice, branching out by a factor of 2 for the given number of levels.

```
function branch2(level) {
  if (level > 0) {
    branch2(level - 1);
    branch2(level - 1);
  }
}
```

How many times will the `branch2()` function be called, in total, in the recursion chain (or tree) that begins with a call to `branch2(3)`?

Confidence: 0 | 1 | 2 | 3 | 4

I2. Data Types

When we write whole numbers (integers) in decimal, the rightmost digit is the smallest, and each successive digit to the left has a place value ten times greater. In binary, each digit to the left has a place value two times greater. Each binary digit is called a "bit".

1. The binary number 1101_2 is equal to the decimal number 13_{10}. What is 11010_2 in decimal?

Confidence: 0 | 1 | 2 | 3 | 4

2. The bit shift operators << and >> in C have the exact effect they are named for: they shift the bits in a binary integer to the left, or to the right. For example, the expression `x >> 2` gives the result of shifting binary integer `x` by 2 bits to the right. What is the mathematical effect of such an operation?

Confidence: 0 | 1 | 2 | 3 | 4

3. Historically, an unsigned short integer represents any positive integer from 0 to its maximum possible value, using 16 bits. What is the maximum possible value of an unsigned short integer?

Confidence: 0 | 1 | 2 | 3 | 4

I3. Practical Software Engineering

Good programmers tend to develop a lot of discipline in the way they write their code, to prevent the occurrence of "bugs", where software does not behave as intended. This discipline usually involves consistency, careful indenting, the use of meaningful labels, and using extensive comments, among other techniques. Nevertheless, bugs inevitably arise, and hence we need the practice of "debugging", to find bugs and correct them.

1. Imagine that we have developed a new sorting algorithm, and written some code that should sort a list of up to 10000 integers, into ascending order. We cannot test "all possible lists" for correct program behaviour. Instead, describe a variety of test cases that we should use, to check that the program behaves correctly.

Confidence: 0 | 1 | 2 | 3 | 4

I4. Advanced / Abstract Data Structures

1. We can describe a "queue" as a data structure that has a "first in, first out" mode of operation. The `push()` operation adds an object to the queue; the `pop()` operation removes an object from the queue. We can say that we add objects to the end of the queue; we remove objects from the start of the queue.

Consider a queue of integers. After the following sequence of operations, what is the next number that will `pop()` off the queue?

```
push(5);  push(3);  pop();  push(3);  push(9);  pop();
```

Confidence: 0 | 1 | 2 | 3 | 4

2. A "stack" is a data structure that has a "last in, first out" mode of operation. The `push()` operation adds an object to the stack; the `pop()` operation removes an object from the stack. We can say that we add objects and remove objects from the top of the stack.

Consider a stack of integers, which we are pushing successive integers onto (in order, 1, 2, 3, and so forth). In between some of the `push()` operations, we perform some `pop()` operations.

Propose a sequence of push() and pop() operations that would result in the state of the stack, as represented by the table below.

10
9
6
4
3
1

Confidence: 0 | 1 | 2 | 3 | 4

I5. Algorithms

1. The classic "guess the number" game is a prototype for searching through an ordered list.

   In the number-guessing game, the computer selects a target number, without the player knowing what number it is. The number will usually be an integer in some range, say 0 to 100 inclusive.

   The player then proceeds to try to guess the number, by typing in successive guesses, one at a time. The computer will respond to each guess as it is made, indicating whether the target number is higher than, lower than, or equal to the guess.

   The game ends when the player successfully guesses the target number. We can measure the performance (in this case, efficiency) of our guessing strategy, as the number of guesses required to finish the game: the fewer guesses it takes, the more efficient our strategy is.

   1. In the "brute force" approach, the player guesses successive numbers in turn until reaching the target: first 0, then 1, then 2, and so forth.

      # 1. What is the "best-case" number of guesses for this strategy? (What is the least number of guesses required, to successfully guess the number, if the computer happened to pick the most favourable target number for our strategy?)

      Confidence: 0 | 1 | 2 | 3 | 4

      # 2. What is the "worst-case" number of guesses for this strategy? (What is the most number of guesses required, to successfully guess the number, if the computer happened to pick the least favourable target number for our strategy?)

      Confidence: 0 | 1 | 2 | 3 | 4

3. What is the "average-case" number of guesses for this strategy? (If the computer picked a target number at random, how many guesses would it require, on average, to successfully guess the number? Assume that the random target numbers are uniformly distributed.)

Confidence: 0 | 1 | 2 | 3 | 4

2. In the "binary search" approach, the player chooses guesses in a more intelligent fashion. Each guess will split the remaining possible numbers into approximately half.

1. Using this strategy, what is the worst-case number of guesses required?

Confidence: 0 | 1 | 2 | 3 | 4

2. Instead of guessing integers between 0 and 100, consider a version of this game where the player guesses integers between 0 and 1000.

What is the worst-case number of guesses required for this, more difficult, version?

Confidence: 0 | 1 | 2 | 3 | 4

2. We cannot wrap up this introduction to algorithms without some work on sorting algorithms.

1. Sort the following integers into order, from lowest to highest, arranged top to bottom in the table.

299	
144	
396	
570	
388	
720	
989	
466	

Confidence: 0 | 1 | 2 | 3 | 4

2. Briefly describe the strategy or strategies used to sort the integers above.

Confidence: 0 | 1 | 2 | 3 | 4

END