How do we know an algorithm solves a particular problem?

1. Give two different ways in which a software bug can arise and give two different effects of a software bug [2]

Solution:

- Programming Error
- Secondary Program Error like a compiler

2. Give the different aspects of the software engineering software design process known as the Waterfall model and very briefly explain what they are [5]

Solution:

- Requirements Phase - Produce a specification of what a program is supposed to do
- Design Phase - Compose a plan for the solution involving algorithms, data structures etc
- Implementation phase - Write the code
- Verification Phase - Test and Debug
- Maintenance - Continual modification/updating

3. When it comes to establishing program correctness, briefly compare and contrast the different approaches taken in software testing and in formal methods [2]

Solution:

Formal Methods - Mathematically based techniques for the formal specification and verification of software and hardware systems
Software Testing - Evaluating an attribute or capability of a program or system and determine that it meets its required results

4. What is a software metric and what is one used for [2]

Solution:

Software Metric - A measure of testing-hardness
It is used to evaluate the performance of software

5. What is the difference between an algorithm being totally correct and being partially correct? [2]

Solution:

Totally Correct - Correct with respect to some specification and terminates on every input
Partially Correct - Only correct with respect to the specification

6. What is the lexicographical order on pairs of natural numbers? How would you undertake an induction with a property indexed by pairs of natural numbers? [4]

Solution:

- $(a, b) = (c, d)$ iff $a = c$ and $b = d$
- $(a, b) > (c, d)$ iff either $a > c$ or ($a = c$ and $b > d$)

Base Case: Show that $P(0, 0)$ holds (or, in general $P(a_0, b_0)$ for some a_0 and b_0 where $a_0 \geqslant b_0 \geqslant 0$)
Inductive Step: Assume that $P(a, b)$ is true for some (a, b) where $a \geqslant b \geqslant 0$ (in general, where $(a, b) \geqslant (a_0, b_0)$ in the lexicographic ordering). We then prove that if this is then case then $P(a_+, b_+)$ must be true, where (a_+, b_+) is the next pair in the lexicographic ordering after (a, b)
Principle of induction: Allows us to state that $P(a, b)$ holds for all $a \geqslant b \geqslant 0$ (in general, when $(a, b) \geqslant (a_0, b_0)$)

7. Outline how the mathematical principle of induction and the algorithmic construction of recursion are often related in the realm of program correctness [4]

Solution:
Induction has 3 steps:

- Formulate a property P(n)
- Decide upon a base case
- The inductive step of proving for k+1

With recursion we have a base case and a recursive call
It is often easier to inductively prove a recursive algorithm as the base case and recursive call mirrors the inductive step

8. Consider the following algorithm where the input is a positive integer supplied by the variable y [8]

```
algorithm: f(y)
if y==0:
    return 1
else:
    x=f(y-1)
    return y*x
```

What does this algorithm do? Use induction to prove that it is totally correct. What would this algorithm do if given a negative integer as input?

Solution:
This recursively computes factorials.

In the case when $0 \leqslant y \leqslant n$.
Base case - When y takes the value 0 as input, the algorithm terminates and the value $1 = 0!$ is

output, so P(0) holds
Inductive step - Assume that P(n) holds for some $n \geqslant 0$; that is, f(y) terminates and is correct whenever it holds that $0 \leqslant y \leqslant n$. In order to prove that P(n+1) holds, we need to show that

- Whenever we have the input value for y is $n + 1$, the algorithm terminates; and
- the value $(n + 1)!$ is output

For n+1 the if statement fails and so the else will be computed, so recursively call f(y) with y having the value n.
By the IH, we know that this recursive call will terminate and that the value returned will be $n!$. Hence, our (original) algorithm will terminate and the value $(n+1) \cdot n! = (n+1)!$ will be output. So, P(n+1) holds.
By induction, we have that out algorithm is totally correct; that is, it always terminates and correctly computes $n!$ when given $n \geqslant 0$ as input.

9. Consider the following algorithm where the input is a positive integer supplied by the variable n: [7]

```
algorithm: f(n):
(x,y)=(1,n)
while y!= 0:
    x=x*(n-y+1)
    y=y-1
output x
```

What does this algorithm do? Prove that it is totally correct

Solution:

This algorithm iteratively computes factorials.
By observation, the variable y decreases by 1 at the end of every iteration of the while loop. So there will be exactly n iterations of the while loop; consequently, our algorithm always terminates.

For correctness. We can also see that at the end of each iteration of the while loop, the value of n-y+1 increases by 1. Prior to the first execution of the while-loop, the value of n-y+1 is 1. Thus, by observation, during the ith iteration of the while loop, x is multiplied by i, having been initialized at 1. Consequently, after the nth iteration of the while-loop, x has the value n!.

10. Let $\Pi = \{(a,b) : a,b \in \mathbb{N}, a \geq 0, b \geq 0, a \geq b\}$. What is the lexicographic ordering on Π? If P(a,b) is a property indexed by $(a,b) \in \Pi$, outline how one might undertake a proof by induction to prove that P always hold [4]

Solution:

$$(0,0) < (1,0) < (1,1) < (2,0) < (2,1) < ...$$

Base case P(0,0) holds as it is the lowest entry
Inductive step, assume $P(a_k, b_k)$ is greater than its predecessors
Therefore $P(a_{k+1}, b_{k+1})$ will be larger than $P(a_k, b_k)$

11. Here is an iterative version of Euclid's algorithm where we assume that the inputs are positive integers supplied via the variables m and n, and that on input $m \geqslant n$ with (m,n)$\neq$ (0,0) [6]

```
algorithm: EuclidIt (m,n)
(x,y)=(m,n)
while y!=0:
    x=x-y
    if y>x:
        swap x and y
output x
```

Outline how we might use a check-point and an invariant in order to prove that this algorithm correctly outputs the GCD of the two input numbers m and n (You should only outline how we might do this; there is no need to provide a full proof)

Solution:

Set a check point prior to the execution of the WHILE test, we can show that the values are decreasing within the lexicographical order and thus it will eventually terminate
An invariant is a checkpoint in a loop that will always simplify to true, if it does then it will show the program terminates and gives the correct output

12. Here is a recursive version of Euclid's algorithm where we assume that the inputs are positive integers and are supplied via the variables x and y, and on that input $x \geq y$ with (x,y)$\neq 0$ [8]

```
algorithm: Euclid(x,y)
IF y == 0:
    return x
ELSE:
    set x' = y and y' = x mod y (*)
    set x = x' and y = y'  (*)
Euclid(x, y)
```

Prove that if m and n are the values of the variables x and y, respectively, immediately prior to executing the lines labelled (*) and that m′ and n′ are the values of the variables x and y, respectively, immediately after executing the lines labelled (*) then the greatest common divisor of m and n is equal to the greatest common divisor of m′ and n′. How does this help you to prove that if Euclid terminates then it outputs the greatest common divisor of the two input values for x and y?

Solution:

The lines labelled (*) decreases (x,y) in the lexicographical order, but it remains part of the order and so will have the same GCD. Because it is decremented in the lexicographical order, it will eventually hit y=0 and so terminate, because (x′,y′) is always GCD (as shown by the invariant), when y does hit 0, x will be the GCD

13. What are the three fundamental aspects of a successful proof of total correctness when using invariants? (You should provide brief explanations of these aspects.) [3]

Solution:

- Initialisation - Show that the invariant holds prior to the first iteration of the loop
- Maintainance - Show that the invariant is maintained
- Termination - Show that the invariant holds when the loop stops

14. What are the research areas of formal specification and formal verification? Give two applications where formal specification and formal verification techniques are used in industry. [4]

Solution:

Formal specification - The study of the specification of a program's properties using a specification language defined by logic
Formal Verification - Mathematical techniques are used to ensure that a design conforms to some precisely expressed notion of functional correctness.

Integrated circuit design by ARM and space systems by NASA