Joao Pereira
19354106

Q1

Q1(a)
A function type in Haskell is a type within a function (E.g. Char, Bool, Int) that operates with the arguments in a function in order to produce a result. An example of a function type could be:

```
isPalindrome :: (Eq a) => [a] -> Bool
```

An implication could be the fact that function types must be precise to the types that will be used within the function. Another issue will be how you order the types with the last type being the return type. This can lead to complications with ordering and then proceeding to use them in function.

Q1(b)
Class constraints are all the types before the "=>" symbol in a function. They declare the types that will be used in the function that belongs to that constraint class.

Example of a constraint:

```
isPalindrome :: (Eq a) => [a]
```

Q1(c)

```
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]

isPalindrome :: (Eq a) => [a] -> Bool
isPalindrome [] = error"Empty list"
isPalindrome x = if reverse x == x
                 then True
                 otherwise False
```

Q2

Q2(a)
How do guards operate and provide an example:
Guards are implemented by using pipes which are the "|" symbol and they operate by performing the first guard and if it outcomes as True it then performs the function and if it does not outcome true it then moves on to the next guard and so on. They operate is well within a function as well as work with parameters they provide.

An example of guards in operation:

```
insert :: Ord t => t -> BinTree t -> BinTree t
insert x Empty = Node x Empty Empty
insert x (Node y left right)
    | x == y            = Node y left right
    | x < y             = Node y (insert x left) right
    | otherwise         = Node y left (insert x right)
```

This is code I have written for a bst in Comparative languages and it displays the use of guards. It evaluates through them all until it reaches the last one where it performs an otherwise

Q2(b)

```
evalPoly :: Int -> [Int] -> Int
evalPoly x [] = 0 -- if empty return 0
evalPoly x (value:values) = value + x * (evalPoly x values)
```

Q2(c)

```
shortest :: [[a]] -> [a]
shortest [] = []
shortest [x] = x
shortest (first:second) = if length first > length (shortest second)
                          then shortest second
                          else first
```

Q3

Q3(a)

Complexity Class P is defined to be a set of problems that can be solved through polynomial time in deterministic algorithms, in other words, O(nk) where k is constant. They can be solved in the worst cases. An example would be let's the multiplication of two numbers which would therefore be polynomial.

Q3(b)
Similar to Class P however it is defined to be a set of problems that can be solved in polynomial time through a non-deterministic algorithm. They're defined in the polynomial-time of O(nk) as well.

The functional difference between both would be that Class NP can come off as more optimized than Class P. Since non-deterministic Turing machines are known to be more powerful and efficient than deterministic Turing machines, it determines that NP is more dominant as it is non-determinant based.

Q3(c)
The Cook-Levin theorem states that satisfiability problem or SAT is NP-Complete in the sense that any problem can be reduced in NP. Essentially SAP is therefore NP-Complete.

The proof of the theorem operates by :
Requiring that all clauses are to be satisfied where each single claus will have one condition.

1 - Firstly, with all steps, the machine is at least in one state.
This intends that at least one of the variables K is True. With this in mind, it then proceeds to a set of clauses, one for each step of i {Qi,1, Qi,2, . . . , Qi,K }.

2 - AThe machine is not in more than one single state at each step.
This intends that for each step i and each pair in that step (i, j), the clauses {Qi,j ', Qi,j ''} must be true.
3 - Within each step, all the tape squares consist of one symbol from the machine's set alphabet.
This then proceeds to two clauses that need at least one symbol per square at each step and that there are no two symbols in each square at each step.
The clauses that perform this are {Si,j,1, Si,j,2, . . . , Si,j,A} and {Si,j,k ', Si,j,k ''} where A is the number of letters, and where for i (each step), j (square) and the the distinct symbols pairs (k' and k").

4- Within each step, the head of the tape is positioned over a single square
We define this by if the tape head is positioned over the single square and if for each step the tape is not positioned on two or more squares.
At least one square: {Ti,1,Ti,2, . . . ,Ti,P(n)}
not positioned on two or more squares: {Ti,j ',Ti,j ''}
This results in providing the clause: O(P(n) 2) + O(P(n) 3) = O(P(n) 3)

5 - At first, the machine is in a state of 0, the head of the tape is in square 1 and x which is the input string is in square 1 to n.

6 - Within the step "less than or equal to" P(n), the machine is in a state of {Q1,qY , Q2,qY , . . . , QP(n),qY }.

7 - Within each step, the machine proceeds onto the next global state which can be as discussed, a state, symbol or head while corresponding to the previous state or symbol. In order to result in a clause that will permit this, we are greeted with a condition that states the square (j) over the tape is not possible to change at step (i) of the computation if the tape head is not in a position at that current moment. This condition correlates into {Ti,j , Si,j,k, Si+1,j,k} clauses.

With the theorem in mind and all of its properties and aspects, it proves that every language in NP can be reduced to SAT.

https://www.computing.dcu.ie/~davids/courses/CA320/CA320_Complexity_2p.pdf

Q5

Q5(a)
In simple terms
Computability is regarding what can be computed and what cannot.
Complexity is a term based on what is required to compute the problems that are computable and how efficient can be computed.

Q5(b)
Posible with Computing. The term or quote itself has a lot of meanings behind it. Within computability, possibilities can sometimes come across as limitless and endless. Of course with Computing there have been people who have demonstrated how even some things that we have to come out as impossible, possible. However, not every possibility is a solvable possibility. Like the Turing machine, where once applied on any program or algorithm, any task that is not possible is then solvable but then the halting problem comes in proving that you can write a program which will only finish when it is solved, now to know whether the program will ever terminate, you have to answers questions that anyone can come to a thought of but you're unable to that, which is the halting problem. This means in regards to the halting problem, to develop a program to get another program to terminate or halt, is logically impossible.

We like to think of everything possible within computing whether that is with developing unthinkable algorithms or branching a new idea that no know has ever thought about. We think of everything with boundaries that are avoidable however in some cases they are not possible until further time comes or a new concept regarding what is possible is introduced. Every problem has its limits and some limits are not possible to overcome, some programs will run forever and as the halting problem stated, it is logically impossible to come to a halt.

Future technologies could impact this phrase in several ways. Within the technology world, it is continuously growing to be bigger and bigger. Every day we notice new technologies and applications being developed and every day someone is surprised by how technology has surpassed a limitation. What we thought was impossible five years

ago has already been developed recently or soon in the future it will be developed in one way or another. Of course, not everything is feasible or possible, and although we have thesis' doubting possibilities of the computing world, there will always be another thesis' or proof that impossible is possible. It all just takes time and determination. In my opinion, in the coming future, thousands of new algorithms, products, applications and programs will be developed that I now think would not be possible. With time, limitations will be reduced and the term of what is possible will gradually be slimmer and slimmer as technology advances.

In science, everything is possible, but in reality, not everything is possible in today's day.