**Problem 2:**

1. Given P, B, Q, and R, to choose an invariant I that satisfies Brown's rule and implies Hoare's rule:

we can define I as:

I = P ∧ R

This choice of invariant ensures that both P and R are satisfied throughout the execution of the loop. The conjunction of P and R captures the necessary conditions for Hoare's rule to hold: P implies the initial condition before the loop, and R captures the intermediate state when the loop condition B is true.

1. To prove Hoare's rule assuming Brown's rule, we need to show that Brown's rule implies Hoare's rule.

Here is the proof:

**Assume Brown's rule:**

P ∧ B → R

{R} C {(B ∧ R) ∨ (¬B ∧ Q)}

P ∧ ¬B → Q

{P} while B do C {Q}

**We need to prove Hoare's rule:**

P → I

{I ∧ B} C {I}

I ∧ ¬B → Q

{P} while B do C {Q}

**Proof:**

1. Assume P.
2. Since P ∧ B → R (from Brown's rule) and P holds, it follows that B → R.
3. Let's choose I = P ∧ R.
4. From B → R (step 2) and I = P ∧ R, we have I ∧ B → R.
5. From {R} C {(B ∧ R) ∨ (¬B ∧ Q)} (from Brown's rule), we have {I} C {(B ∧ I) ∨ (¬B ∧ Q)} since I = P ∧ R.
6. Since I = P ∧ R and {I} C {(B ∧ I) ∨ (¬B ∧ Q)}, we have {I ∧ B} C {I} (by substituting I for P ∧ R).
7. From P ∧ ¬B → Q (from Brown's rule) and P holds, it follows that ¬B → Q.
8. From I = P ∧ R and ¬B → Q, we have I ∧ ¬B → Q.
   1. 9. Finally, from P (step 1), {I ∧ B} C {I} (step 6), and I ∧ ¬B → Q (step 8), we can conclude {P} while B do C {Q} (Hoare's rule).

Therefore, assuming Brown's rule, we have proven Hoare's rule.

**Problem 3:**

Calculate all veriﬁcation conditions generated by the following annotated speciﬁcation

{x = n}

y:=1;

while x!= 0 do

invariant x! ∗ y = n!

y:=y\*x;

x:=x-1

{x = 0 ∧ y = n!}

Precondition {x=n}

* This specifies the initial condition where ‘x’ is equal to ‘n’

Postcondition {x = 0 ∧ y = n!}

* This specifies the final state of the programm where ‘x=0’ and ‘y=n’

The Loop variant hold true at the beginning of the Programm and after each while-loop

1. After the first while loop

* verification condition: because ‘x = n’ -> x!\*y = n!\*1=n!

1. Inside the while loop

* verification condition: x!\*y = n! ∧ x != 0 -> (x-1)!\*(y\*x) = (x-1)!\*n = (n-1)!\*n = n!

1. After the while loop

* Verification condition: x!\*y = n! ∧ x = 0 -> y =n!
* This condition is valid because it ensures that the loop terminates (‘x’ become 0)

and ‘y’ is equal to ‘n!’

* All the conditions generate by the annotated specification are valid.