Part 1

1. Two types of Proofs in SNARKS:
    a. PLONK
    b. GROTH16

2. SNARKS rely on an initial set of configuration parameters which are then used to construct proofs and hence get encoded into the protocol. These configuration parameters are one of the necessary factors in proving validity of a transaction. SNARKS require a trusted setup between a verifier and a prover to generate these initial config parameters. STARKS on the other hand rely on collision-resistant hash functions and hence do not require these initial config parameters.

3. Two more differences:
    a. SNARKS are not resistant to attacks by Quantum Computers. STARKS are resistant to attacks by quantum computers.
    b. Transacting with SNARKS is much cheaper. They only require 24% of gas required by STARKS.

Part 2

1.

2. Q2 Helloworld.circom
    1. Helloworld.circom defines a Multiplier2 circuit that multiplies two private input signals and produces one output signal. The true reationship between the input and output signals are captured using the constraints.
    2. Powers of Tau ceremony is used to generate an initial set of configuration parameters called Common Reference String(CRS). These are essential to the construction of any zk-SNARKS proof and the validation of transactions.
    3. Phase 1 of the ceremony typically deals with the initial construction of a Common Reference String(CRS) using a Multi-Party-Computation(MPC) setup. Phase 2 is where CRS is evolved to include an individual cicuit's arithmetic. It basically adds to the randomness of the CRS.

3. Q2 Multiplier3
    1. See script in commit
    2. Error is as follows:

```
error[T3001]: Non quadratic constraints are not allowed!
    ┌─ "Multiplier3.circom":14:4
    │
14  │      d <== a * b * c;
    │      ^^^^^^^^^^^^^^^ found here
    │
    = call trace:
      ->Multiplier3

previous errors were found
```

Circom does not allow non quadratic constraints. Which means as per there documentation "*All constraints must be of the form A*B + C = 0, where A, B and C are linear combinations of signals.*"

3. See code in commit.

4. Multiplier3 Plonk:
    1. There is no phase 2 of powersofTau ceremony required for plonk. The initial "snarkjs plonk setup" command directly geenerates the final verification keys needed to construct the proof.
    2. Practical differences between Groth16 and Plonk:
        i. Groth16 seems to have a slightly shorter runtime as compared to Plonk.
        ii. Solidity verifier from plonk is more straightforward in terms of expected inputs. You can directly pass the proof and public signals onto the PlonkVerifier.verifyProof() function. The solidity verifier in Groth16 is more complex with abstract inputs(i.e. a,b,c, Input).

5. Test Circuit:
    1. Look at code
    2. Look at code
    3.

```
divyam@LAPTOP-GUBNU639:~/harmony_zku/zku-c3-week1/Q2$ npm run test

> test
> node scripts/bump-solidity.js && npx hardhat test



  HelloWorld
1x2 = 2
    ✔ Should return true for correct proof (3520ms)
    ✔ Should return false for invalid proof (432ms)

  Multiplier3 with Groth16
1x2x3 = 6
    ✔ Should return true for correct proof (2269ms)
    ✔ Should return false for invalid proof (392ms)

  Multiplier3 with PLONK
1x2x3 = 6
    ✔ Should return true for correct proof (2886ms)
    ✔ Should return false for invalid proof


  6 passing (11s)
```

Part 3

1. Comparators.circom:
   1. 32 stands for the number of bits the input can have.
   2. True if first element of input array(i.e. in[0]) is less than second element of input array (i.e. in[1]).
   3. See code.

2. Sudoku
   1. Look at code
   2. Error is as follows:

```
[INFO]  snarkJS: Plonk constraints: 97750
[ERROR] snarkJS: circuit too big for this power of tau ceremony. 97750 > 2**16
[ERROR] snarkJS: Error: build/sudoku/circuit_final.zkey: Invalid File format
    at Object.readBinFile (/home/divyam/harmony_zku/zku-c3-week1/Q3/projects/zkPuzzles/node_modules/@iden3/binfileutils/
build/main.cjs:36:35)
    at async zkeyExportVerificationKey (/home/divyam/harmony_zku/zku-c3-week1/Q3/projects/zkPuzzles/node_modules/snarkjs
/build/cli.cjs:5283:28)
    at async Object.zkeyExportVKey [as action] (/home/divyam/harmony_zku/zku-c3-week1/Q3/projects/zkPuzzles/node_modules
/snarkjs/build/cli.cjs:8357:18)
    at async clProcessor (/home/divyam/harmony_zku/zku-c3-week1/Q3/projects/zkPuzzles/node_modules/snarkjs/build/cli.cjs
:303:27)
[ERROR] snarkJS: Error: build/sudoku/circuit_final.zkey: Invalid File format
    at Object.readBinFile (/home/divyam/harmony_zku/zku-c3-week1/Q3/projects/zkPuzzles/node_modules/@iden3/binfileutils/
build/main.cjs:36:35)
    at async zkeyExportVerificationKey (/home/divyam/harmony_zku/zku-c3-week1/Q3/projects/zkPuzzles/node_modules/snarkjs
/build/cli.cjs:5283:28)
    at async exportSolidityVerifier (/home/divyam/harmony_zku/zku-c3-week1/Q3/projects/zkPuzzles/node_modules/snarkjs/bu
ild/cli.cjs:5376:29)
    at async Object.zkeyExportSolidityVerifier [as action] (/home/divyam/harmony_zku/zku-c3-week1/Q3/projects/zkPuzzles/
node_modules/snarkjs/build/cli.cjs:8410:26)
    at async clProcessor (/home/divyam/harmony_zku/zku-c3-week1/Q3/projects/zkPuzzles/node_modules/snarkjs/build/cli.cjs
:303:27)
```

This error can be fixed by changing the number of constraints from 2^16 to 2^17 by changing the file from “powersOfTau28_hez_final_16.ptau” to “powersOfTau28_hez_final_17.ptau”.

3. Look at code

4. The brute force implementation is a O(n^4). The existing “sum and sum of squares of each row” is a O(n^2) solution making it more efficient(i.e. faster) by a factor of n^2.

3. [Bonus]

4. [Bonus] More libraries to foster growth of ZKapps:
   a. Tensor implementation to perform tensor operations like backpropagation. This can go a long way in enabling of Deep Learning based Dapps.
   b. OpenAIGym(https://gym.openai.com/) style toolkits implementation to work on Reinforcement Learning Problems.
   c. Circom wrappers around Social Media APIs to use private data for analytics/ training ml models without compromising on privacy.