Question 1

a) The kv node points to a branch node which might only one path.

b) For 44431337a, while 444 is mapped to "d", there is no immediate 3 following this path, thus we can prove that 44431337a has no value mapped to it.

For fc and 1a3098a, we would need to provide the tree root and the array of 16 elements following the path f and then c (for fc) and until we reach 098a (for 1a309a).

c) The worst-case is when each level throughout the tree has the maximum possible number of children and equivalent nodes at each step.

d) With the state trie, all addresses are collision-resistant hashes, which are pseurandom, thus making it pseudorandomly distributed in the space. This allows for short proof sizes even in the worst case proof time.

e) The long-term storage in storage tries has pointers to all the previous blocks, which may not be randomly distributed, which means that the proof sizes could be very large. For example, the worst case would be if all 256 nodes are expanded where an expensive call would be to a data that has a very small difference. This could result in all k nodes having to be expanded and a result not being found.

Question 2

a) If t1 has not been reached, OP_IF evaluates to true and OP_CHECKLOCKTIMEVERIFY will only be executed if it exceeds the lock time to avoid a premature refund to Alice. If t1 has already passed, Bob will need to provide a valid signature and Y to Alice.

b) We would have used a third party as such with a multi sig transaction and an escrow to ensure the transaction output can be spent. However, we would also be subjected to higher counterparty risk as we would need to also find a trustable third party that can mediate these transactions.

c)
```
function AliceRedeem (uint25 x) public {
      require (hash(x) == X);
      secret = x;
      transfer (recipient, amount);
}

function BobRedeem () public {
      require (block.timestamp > t2 + lockTime);
      transfer (owner, amount);
}
```

d) Y should be a random, unique value that has never been used before. A good Y could be a random nonce with at least 256 bits to ensure that the other party would not be able to guess it in a reasonable time.

e) Alice should fund her transaction first since Bob was the one who picked Y.

f) t1 should be greater than t2 by a combined time of the transaction preparation time from the counterparty, the expected time to confirm a transaction, and a safety margin.

Question 3

Bug 1:
The constructor should only be able to be called once per contract.

```
constructor(address opponent, uint32 turnLength, bytes32
p1Commitment) public {

     if (_playerAddress [0] == msg.sender)
          revert();
     ...
}
```

Bug 2:
The joinGame function should only be called once per contract.

```
function joinGame(uint8 p2Nonce) public payable {

     if (_p2Nonce != 0)
          revert();
     ...
}
```

Bug 3:
There is a bug stopping the current player from taking a move.

```
function playMove(uint8 squareToPlay) public {

     // make sure correct player is submitting a move
          require(msg.sender == _playerAddress[_currentPlayer]);
     ...
}
```

Bug 4:

The defaultGame function is never called. Which meant that the game wouldn't have stopped a player from taking too long to submit a move.

```
function playMove(uint8 squareToPlay) public {

     defaultGame()
     ...
}
```

Bug 5:
The constructor needs to update balance with the value passed in so it can store what value the opponent must match.

```
constructor(address opponent , uint32 turnLength , bytes32
p1Commitment) public {
     if (_playerAddress [0] == msg.sender)
          revert;
     this.balance = msg.value;
     ...

}
```