1. Untersuchen Sie die Tauglichkeit zur Sperrsynchronisation der folgenden Implementierungen von `Lock/Unlock` (mit `var a, interested [2]bool; var favoured uint`, initial `a[i] == true` und `interested[i] == false` für `i < 2`, `favoured < 2` sowie `p < 2` beim Aufruf):

a)
```
func Lock (p uint) {
  for {
    a[1-p] = ! a[p]
    if a[p] && ! a[1-p] { break }
  }
}
```
```
func Unlock (p uint) {
  a[1-p] = true
}
```

b)
```
func Lock (p uint) {
  for {
    interested[p] = true
    if interested[1-p] {
      interested[p] = false
    }
    if interested[p] { break }
  }
}
```
```
func Unlock (p uint) {
  interested[p] = false
}
```

c)
```
func Lock (p uint) {
  interested[p] = true
  for interested[1-p] && favoured != p {
    Null()
  }
}
```
```
func Unlock (p uint) {
  interested[p] = false
}
```

2. Zum Algorithmus von Dekker für zwei Prozesse:
a) Beweisen Sie (in Anlehnung an den entsprechenden Beweis zum Peterson-Algorithmus im Buch), dass er deren gegenseitigen Ausschluss garantiert.
b) Begründen Sie, dass er alle anderen Anforderungen an Sperrsynchronisation erfüllt.

3. Zeigen Sie, dass die folgende „naive“ Erweiterung des Algorithmus von Peterson zur Sperrsynchronisation auf *drei* Prozesse *nicht* korrekt ist:

```
func Lock (p uint) { // p < 3
  interested[p] = true
  q, r:= (p + 1) % 3, (p + 2) % 3
  favoured = q
  for interested[q] && favoured == q ||
      interested[r] && favoured == r {
    Null()
  }
}
```
```
func Unlock (p uint) { // p < 3
  interested[p] = false
```

4. Zum Bäckerei-Algorithmus von Lamport:
a) Begründen Sie die Korrektheit der ersten Version.
b) Erklären Sie den Schritt von der ersten Version zur zweiten.