A.6:

1. Implementieren Sie in PYTHON eine Funktion $inc(b)$, die eine positive Zahl in Binärdarstellung bitweise um 1 erhöht. Dabei wird b= $(b_0, \ldots, b_{k-1})$ mit k ≥ 1 und bi ∈ {0, 1} als Binärdarstellung der Zahl aufgefasst.

```python
def inc(b):
    m = len(b)                          # O(1)
    for i in range(m, 0, -1):           # höchstens O(m)
        if i-1 == 0 and b[i-1] == 1:    # O(1)
            b.append(0)
            break
        else:                           # O(1)
            if b[i-1] == 0:
                b[i-1] = 1
                break
            else:
                b[i-1] = 0
    return b
```

2. Führen Sie eine vereinfachte Laufzeitanalyse durch. Was kostet ein einzelner Aufruf im schlechtesten Fall, und was kosten n ≥ 1 Aufrufe von inc?

Im schlechtesten Fall wird die for-Schleife m-mal durchlaufen:

$$O(1) + O(m) \cdot O(1) \subseteq O(m)$$

Für den n-fachen Aufruf gilt:

$$n \cdot O(m) \subseteq O(m)$$

3. Führen Sie eine amortisierte Laufzeitanalyse durch, indem Sie n ≥ 1 Aufrufe von inc gemeinsam betrachten. Sie können vereinfachend annehmen, dass n eine 2er-Potenz ist. Bei wie vielen dieser n Aufrufe wird b_{k-1} betrachtet, und wie ist es für b_{k-2}, b_{k-3} usw.? Fassen Sie die Ergebnisse in einer Summe zusammen.

Bei n-Aufrufen wird

- b_{k-1} n-mal überprüft
- b_{k-2} $n/2$-mal überprüft
- b_{k-3} $n/4$-mal überprüft
- b_{k-4} $n/16$-mal überprüft
- ....

Anzahl der Aufrufe als Summe mit $m = len(b)$:

$$\sum_{i=1}^{m} \frac{n}{2^{i-1}}$$