Aufgabe 3 - Algorithmen

Freitag, 8. März 2024 17:37

1.2) Bestimmung Rekurrenzgleichung

$T(n) = \underbrace{T(n-2)}_{1} + \underbrace{T(n-3)}_{2} + 1$ Grundoperation

```
    System.out.println("Die " + n + "-te Zahl der Perrin-Folge ist: " + Perrin(n));
    scanner.close();
}

private static int Perrin(int n) {
    switch (n) {
        case 0:
            return 3;
        case 1:
            return 0;
        case 2:
            return 2;
        default:
            return Perrin(n - 2) + Perrin(n - 3);
    }
}
```

(Perrin(n-2): 1, Perrin(n-3): 2) → Rekursionsaufrufe

1.3) Vollständige Induktion $T(n) \leq 2^n$

- **Induktionsbasis:**

  $n \in \{0,1,2\}$: Fallunterscheidung → keine Berechnung notwendig → $O(1) = 1 \leq 2^n$ (1, 2, 4)

  $n = 3$: $T(3) = T(1) + T(0) + 1 \leq 2^3$

  $3 \leq 8$ ✓

- **Induktionsannahme:**

  Angenommen $T(n) \leq 2^n$ ist wahr für ein beliebiges, aber festes $n \geq 2$, dann gilt dies auch für $n+1$!

- **Induktionsschritt:**

1) Es gilt für $n \to n+1$:

$$T(n+1) = T(n-1) + T(n-2) + 1 = 2^{n+1}$$

---

2) Induktionsannahme „$T(n) \leq 2^n$" gilt:

$$2^{n-1} + 2^{n-2} + 1 \leq 2^{n+1}$$

---

3) 1 kann nach oben zu 2^{n-2} abgeschätzt werden:

$$2^{n-1} + 2^{n-2} + 2^{n-2} \leq 2^{n+1}$$

$$\Rightarrow 2^{n-1} + 2^{n-1} \leq 2^{n+1}$$

$$\Rightarrow 2^n \leq 2^{n+1}$$

Q.E.D

1.5) Laufzeit:

```
private static int Perrin(int n) {
    switch (n) {
        case 0:
            return 3;
        case 1:
            return 0;
        case 2:
            return 2;
        default: {
            int[] perrinValues = new int[n + 1];
            perrinValues[0] = 3; perrinValues[1] = 0; perrinValues[2] = 2;

            for (int i = 3; i <= n; i++) {
                perrinValues[i] = perrinValues[i - 2] + perrinValues[i - 3];
            }

            return perrinValues[n];
        }
    }
}
```

Fallunterscheidung (konstante Zeit) => O(1)

Array-Erstellung mit Werte (einmalig) (konstante Zeit) => O(1)

Berechnung (linear abhängig von i) => O(i)

Rückgabe (konstant) => O(1)

=> O(i)

O(1) sind vernachlässigbar, da O(i) höhere Aufwandsklasse

Speicherkomplexität:

- Eingabe: 1
- Array: n

=> O(n) (linear abhängig von Array-Größe)