Training für Programmierwettbewerbe - Exam Questions

Hanno Barschel (174761)

1 1 Introduction

(1) Zeit Komplexität von drei Funktionen:

- **foo(.)** hat eine Zeitkomplexität von $\mathcal{O}(n)$ (wobei n die Länge des gegebenen Vektors ist) und eine Platzkomplexität von $\mathcal{O}(n)$.

  Das folgt direkt daraus, dass wir genau 2 Vorschleifen die jeweils den gesamten Vektor durchgehen haben also $\mathcal{O}(2n) = \mathcal{O}(n)$.

- **print_ pairs(.)** hat eine Zeitkomplexität von $\mathcal{O}(n^2)$ und eine Platzkomplexität von $\mathcal{O}(n)$.

  Für jedes Element des Vektors ruft die Funktion nochmal jedes Element des Vektors auf und printed beide als Paar aus. Somit kommt eine Zeitkomplexität von $\mathcal{O}(n^2)$ zustande.

- **print_ unordered_ pairs(.)** hat eine Laufzeitkomplexität von $\mathcal{O}(n^2)$ und eine Platzkomplexität von $\mathcal{O}(n)$.

  Für jedes Element des Vektors rufen wir alle Folgeelemente auf. Das sind es $n - (i + 1)$ Aufrufe für Element i (i startet bei 0). Damit kommen wir auf eine Laufzeit von

$$\sum_{i=0}^{n-1}(n - (i + 1)) = n^2 - \sum_{i=1}^{n} i = n^2 - \frac{n \cdot (n + 1)}{2} = n^2 - \frac{n^2}{2} - \frac{1}{2} \in \mathcal{O}(n^2).$$

- **all_ fib(.)** hat denke eine Zeitkomplexität von $\mathcal{O}(n)$ und eine Platzkomplexität von $\mathcal{O}(n)$.

  Die for-Schleife printed uns im Prinzip alle Fibonacci Zahlen bis n aus. Sobald wir eine Null in $memo[i]$ finden sind aber bereits $memo[i-1]$ und $memo[i-2]$ berechnet, somit müssen wir diese nur addieren und haben keine wirklich tiefe Rekursion.