Übungsblatt 02 – Datenstrukturen und Algorithmen

Name: Taha Darende

Matrikelnummer: 3724493

Name: Öznur Gencoglu

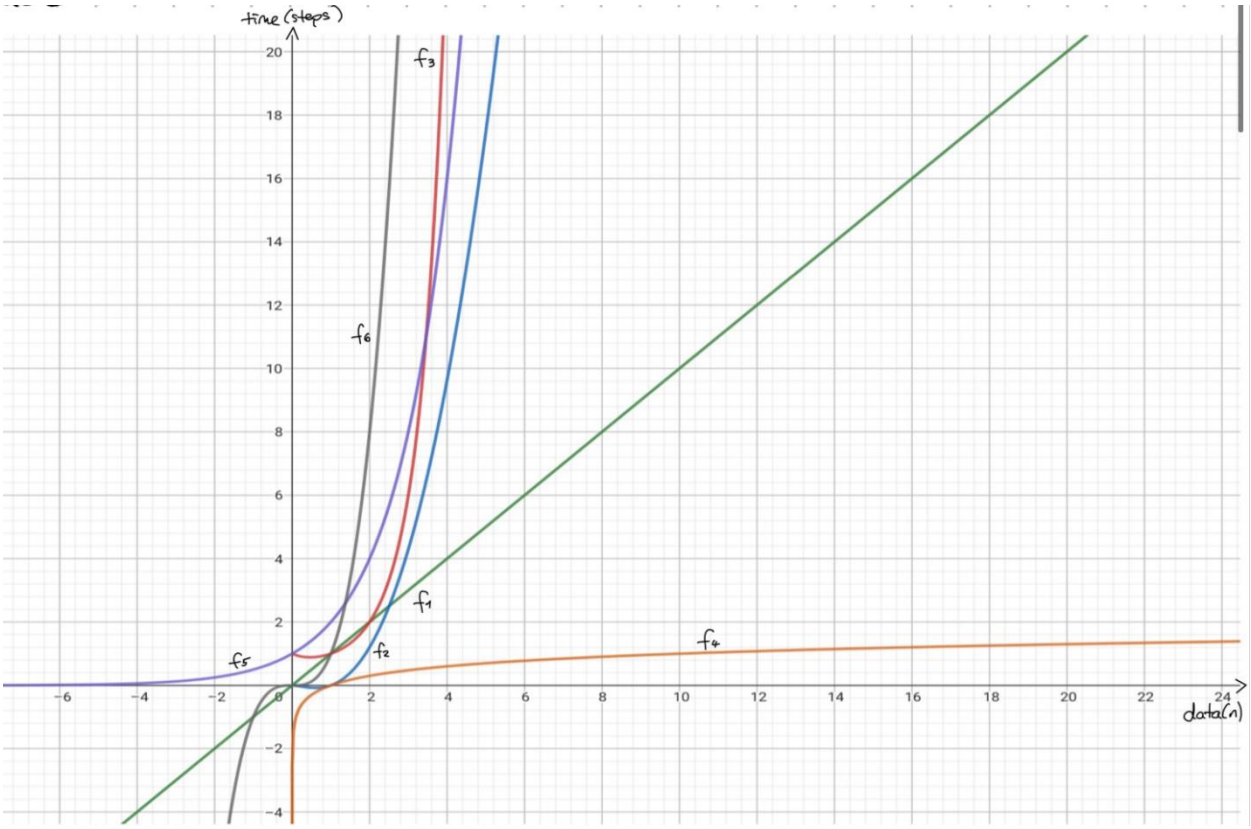
Matrikelnummer: 3682221

Name: Jeannine Renner

Matrikelnummer: 3724419

Aufgabe 1:

a)



b)

Funktion f	***Asymptotische Komplexität f(n) = O(g(n))***	***Komplexitätsklasse***
$f_3(n) = n!$	$O(n!)$	faktoriell
$f_5(n) = 2^n$	$O(2^n)$	exponentiell
$f_6(n) = n^3$	$O(n^3)$	polynomiell
$f_2(n) = n^2\log(n)$	$O(n^2 \log n)$	quadratisch logarithmisch
$f_1(n) = n$	$O(n)$	linear
$f_4(n) = \log(n)$	$O(\log n)$	logarithmisch

Aufgabe 2:

a) Fall 1: Gegeben sind absteigend sortierte Daten.

- ◯ MergeSort
- ◯ QuickSort
- ◯ InsertionSort
- ◯ SelectionSort
- ◯ BubbleSort

➔ Da die Zeitkomplexität ***O(n log n)*** in allen drei Fällen (bester Fall, mittlerer Fall, schlechtester Fall) beträgt und bei absteigend sortierten Daten am besten funktioniert.

b) Fall 2: Gegeben sind diese sortierten Daten → [5, 6, 8, 9, 11, 23, 32, 34, 58, 59, 123, 233, 436, 543].

- ◯ MergeSort
- ◯ QuickSort
- ◯ InsertionSort
- ◯ SelectionSort
- ◯ BubbleSort

➔ Da die Daten bereits sortiert sind und somit keine Verschiebungen notwendig sind.

➔ Zeitkomplexität von ***O(n)***

c) Fall 3: Gegeben sind unsortierte ("chaotische") Daten.

- ◯ MergeSort
- ◯ QuickSort
- ◯ InsertionSort
- ◯ SelectionSort
- ◯ BubbleSort

➔ Da beide eine Komplexität im Durchschnitt von ***O(n log n)*** haben und nach dem Teile-und-Herrsche-Grundsatz funktionieren, sind sie im Vergleich zu den anderen Verfahren für unsortierte Daten gut geeignet.

Aufgabe 3:

a) Sowohl die 1. Schleife als auch die 2. Schleife haben eine Zeitkomplexität von O(n). Daraus folgt: O(n) + O(n) = 2n.
Da der Faktor weggeht bleibt O(n) übrig.
➔ Der Algorithmus hat eine asymptotische Komplexität von ***O(n).***

b) Die 1. Schleife (äußere Schleife) läuft a mal durch (a = 1000) und hat somit eine Komplexität von O(n).
Die innere Schleife hat eine Komplexität von O(log n), deswegen werden die beiden multipliziert und daraus folgt O(n log n).
Die 2. Schleife ist O(n) und dies addiert man mit der ersten for-Schleife.
➔ Der Algorithmus hat eine asymptotische Komplexität von ***O(n log n).***

c) Die äußere Schleife läuft a-mal, in dem Fall a=100 und die innere Schleife läuft n-1-mal
➔ Der Algorithmus hat eine asymptotische Komplexität von ***O(n).***

d) In der 1.Schleife wird *result* in jeder Iteration verdoppelt und hat somit eine Zeitkomplexität von O(log n).
Darauf folgt eine verschachtelte Schleife und die äußere wird „result"-mal durchlaufen, also O(n) und die innere die Hälfte von result.
➔ Der Algorithmus hat eine asymptotische Komplexität von ***O(n log n).***

e) Bei jedem Aufruf werden zwei weitere Aufrufe ausgeführt, was dazu führt, dass die Anzahl an Komplexität exponentiell steigt.
➔ Der Algorithmus hat eine asymptotische Komplexität von ***$O(2^n)$.***

f) Die Verzweigung wird ausgeführt, wenn n größer als 1 ist und n wird dann um 1 verringert.
Bei der for-Schleife läuft es n/2-mal durch, daraus folgt die Komplexität O(n).
➔ Der Algorithmus hat eine asymptotische Komplexität von ***O(n).***

Aufgabe 4:

a) Die asymptotische Komplexität von couldBeBetter1 beträgt O(n) und die von isDoneBetter1 nun O(1).
Da die Anzahl der Operationen konstant und unabhängig von der Eingabe ist, ist diese schneller.

b) Die asymptotische Komplexität von couldBeBetter2 beträgt $O(n^2)$, da es eine verschachtelte Schleife ist und die äußere/innere Schleife jeweils n-mal durchlaufen wird.
Der verbesserte Algorithmus hat eine Komplexität von O(n), da für die Fakultät von n man n-mal durchlaufen muss. Somit ist dies schneller.

c) CouldBeBetter3 zeigt die Fibonacci-Folge rekursiv, wodurch die Schleife n-mal durchlaufen wird und die Fibonacci-Zahlen bei jeder Iteration berechnet werden müssen, führt es zu einer exponentiellen Anzahl an Funktionsaufrufen.
Die Zeitkomplexität beträgt somit $O(2^n)$.
IsDoneBetter3 hingegen stellt die Fibonacci-Folge iterativ dar, wodurch die Zahlen gespeichert werden und man die Schleife somit n-mal durchläuft. Die Komplexität beträgt hier O(n) und ist dementsprechend schneller.

Aufgabe 5:

a) Da die BubbleSort Implementierung auf einer verketteten Liste sich von der Implementierung auf einem Array unterscheidet.
Bei einer verketteten Liste gibt es einen Zeiger, welches jeweils auf das nächste Element zeigt. Dadurch muss man erstmal jedes Element der Liste durchlaufen, um das gewünschte Element zu erreichen. Bei Arrays hingegen kann man direkt auf das gewünschte Element zugreifen.
Außerdem kann man bei verketteten Listen die Elemente nicht einfach, wie bei Arrays vertauschen. Dafür muss man auch jeweils die Zeiger immer ändern, was dazu führt, dass die Implementierung in einer höheren Komplexitätsklasse liegt.