# Übungsblatt 01 – Datenstrukturen und Algorithmen

Name: Taha Darende

Matrikelnummer: 3724493

Name: Öznur Gencoglu

Matrikelnummer: 3682221

Name: Jeannine Renner

Matrikelnummer: 3724419

# Aufgabe 1:

a) Auf einer sortierten Liste ist das binäre Suchverfahren am schnellsten. Bei der binären Suche ist der Aufwand geringer, vor allem wenn die Anzahl der Elemente länger ist, da bei der binären Suche erstmal halbiert und geschaut wird, ob das gesuchte Element oberhalb oder unterhalb liegt.
Bei der sequentiellen Suche hingegen dauert es länger, da es jedes Element durchläuft.

b) Bei ungeordneten Arrays bzw. verketteten Listen ist die binäre Suche nicht möglich, da man den Teile-und-herrsche-Grundsatz nicht anwenden kann. Wenn man das Array halbiert, kann nicht entschieden werden, ob das Element kleiner oder größer ist.
Die sequentielle Suche hingegen läuft jedes Element von links nach rechts und ist somit in dem Fall anwendbar.

c)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|

n = 12

Gesucht: 2

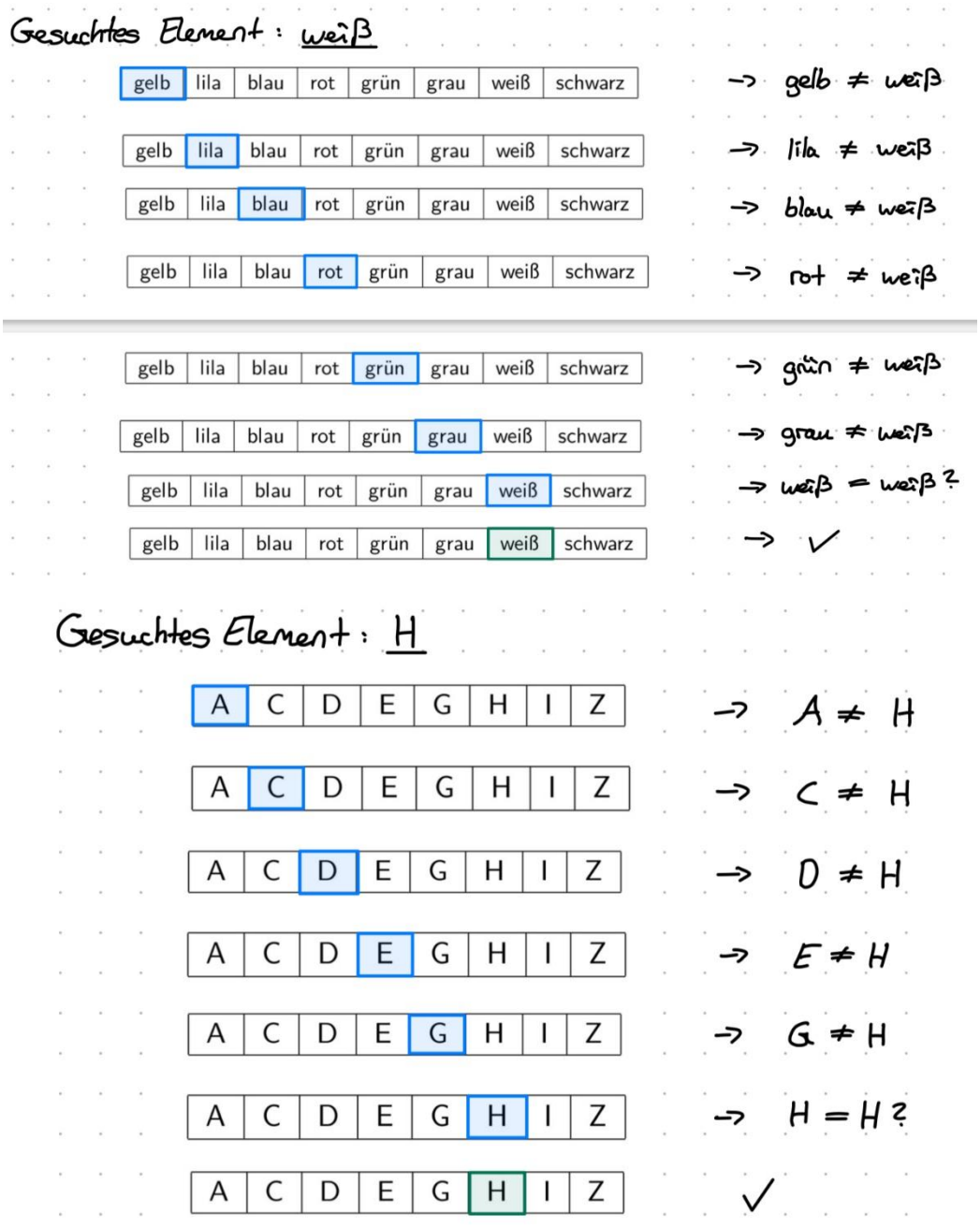
- Sequentielle Suche: Man beginnt bei Element 1 und schaut ob es das gesuchte Element ist. Wenn das nicht der Fall ist wandert es eins weiter zu Element 2. Überprüft, ob es der gesuchte Wert ist und wenn ja, dann ist die Suche abgeschlossen.
  ➔ In diesem Fall braucht die sequentielle Suche nur zwei Schritte und ist für kleine n effizienter als die binäre Suche.
- Binäre Suche: Hier teilt man zuerst durch die Hälfte und schaut, ob die gesuchte Zahl kleiner oder größer als 6 ist. Anschließend halbiert man nochmal und schaut so lange bis das gesuchte Element gefunden ist. Die binäre Suche dauert in diesem Fall länger.

d) Ungünstigste Stellen:
   - Erstes Element der Liste
   - Letztes Element der Liste

   In beiden Fällen benötigt die binäre Suche die maximale Anzahl von Schritten und muss alle Elemente durchlaufen, um das gesuchte Element zu finden, denn wenn das Element am Anfang oder am Ende befindet, kann es nicht halbiert werden.

# Aufgabe 2:

a)




b) 1. Liste: Die binäre Suche ist für die erste Liste nicht möglich, weil man die Elemente nicht aufteilen kann (anhand ihrer Attribute). Außerdem kann man die Farben nicht richtig sortieren und deshalb ist nur die sequentielle Suche möglich.
2. Liste: Bei dieser Liste könnte man sich anschauen, ob das gesuchte Element weiter vorne oder weiter hinten im Alphabet auftaucht. Allerdings gilt dies nur, wenn die Liste sortiert ist, welches hier nicht der Fall ist. Also ist auch hier die binäre Suche nicht möglich.

# Aufgabe 3:

a) Bei SelectionSort wird das größte bzw. kleinste Element an das Ende/Anfang angehängt und dies wiederholt man bei jedem Durchlauf so lange bis die Liste sortiert ist.
Bei InsertionSort wird die Liste durch Einfügen sortiert. Man sortiert die Liste von links nach rechts und fügt das kleinere Element immer an die richtige Stelle ein, während man die anderen Elemente nach rechts verschiebt.

Eigenschaften, in denen sie sich unterscheiden:

| | **SelectionSort** | **InsertionSort** |
|---|---|---|
| **Stabilität** | Instabil | Stabil |
| **Vergleiche im Mittel** | $n^2/2$ | $n^2/4$ |

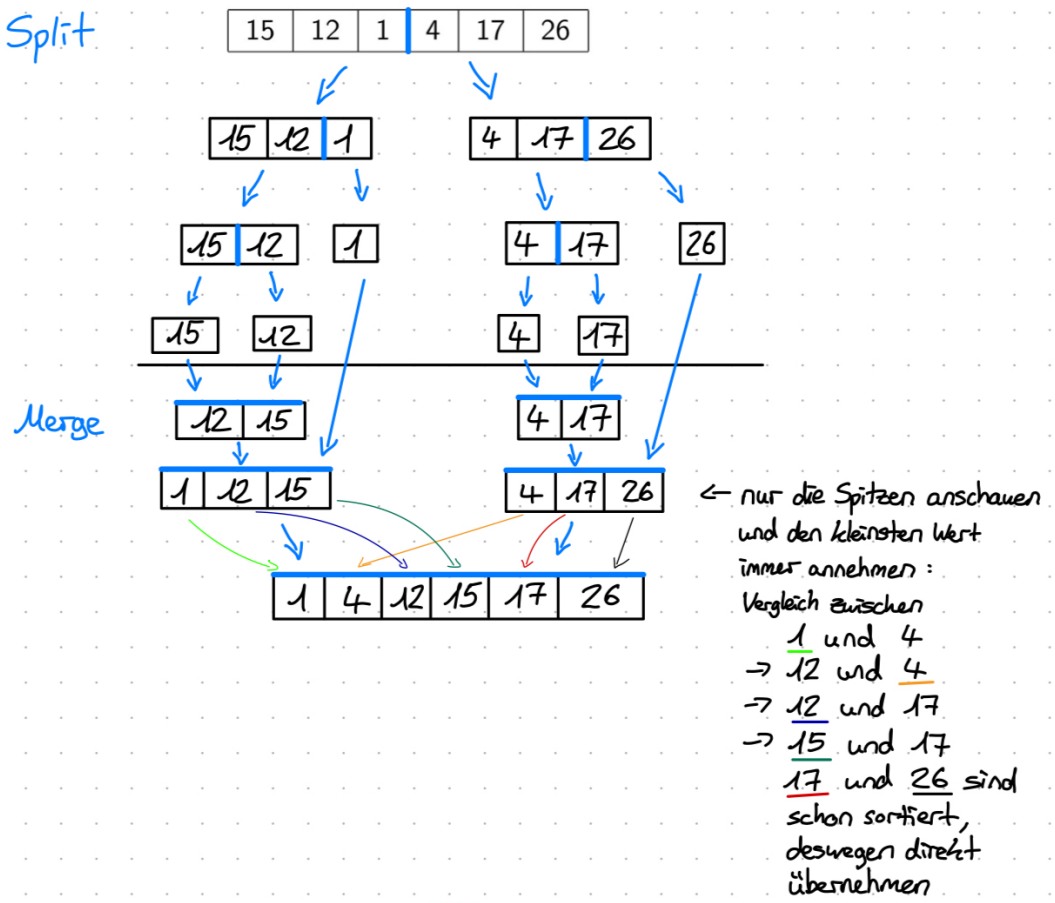
b) InsertionSort wäre vorzuziehen, da es stabiler und effizienter als SelectionSort ist, vor allem wenn die Liste schon zum Teil sortiert ist, braucht man weniger Vergleiche und Verschiebungen.

c) BubbleSort und MergeSort erfüllen diese Eigenschaft, da sie auf externen Medien große Datenmengen sortieren. Außerdem greifen sie weniger auf den Festplattenspeicher zu.

d) Da man wenig Arbeitsspeicher hat, ist Speicherverbrauch entscheidend. QuickSort oder InsertionSort wäre besser in dem Fall als MergeSort, da MergeSort mehr Speicher bewältigt.

# Aufgabe 4:

a)


Split
15 12 1 4 17 26
15 12 1
4 17 26
15 12
1
4 17
26
15
12
4
17
Merge
12 15
4 17
1 12 15
4 17 26
1 4 12 15 17 26
← nur die Spitzen anschauen und den kleinsten Wert immer annehmen:
Vergleich zwischen
1 und 4
→ 12 und 4
→ 12 und 17
→ 15 und 17
17 und 26 sind schon sortiert, deswegen direkt übernehmen

b)