## Aufgabe 4.1.1

f(n) = 2, n = 0
3* f (n-1) +2, n > 0

| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| f(n) | 2 | 8 | 26 | 80 | 242 | 728 | 2186 |

Vermutetes Schema: $3^{n+1} - 1$ für n>0

## Aufgabe 4.1.2

Gegeben ist:

f(n) = 2, n = 0
3* f (n-1) +2, n > 0

Zu zeigen: f(n) = $3^{n+1} - 1$

Beweis mittels Induktion:

**Induktionsanfang.** Für n = 1 gilt f(1) = $3^{1+1} - 1$= 8

**Induktionsannahme.** Wir nehmen an, dass für ein n $\geq$ 1 die Behauptung gilt,
d.h. dass f(n) = $3^{n+1} - 1$ ist.

**Induktionsschritt.** f(n + 1) = 3* f ((n+1)-1) +2 = $3^{n+1} - 1$

Es folgt die Behauptung für alle n $\geq$ 1 gilt damit f(n) = $3^{n+1} - 1$.

## Aufgabe 4.1.3

Abbruchkriterium: n=0

f(n) = 2, n = 0
3* f (n-1) +2, n > 0

f(0) = 2 mit c = 2

Der geschlossene Ausdruck für das Abbruchkriterium n=0 lautet: f(n) = c

Für alle n > 0 gilt f(n) = 3n – 1

## Aufgabe 4.2

2)

Um die versteckten Konstanten zu ermitteln, ist die Idee den Minimal- und Maximalwert für 1 Element für ein Array beliebiger Größe zu bestimmen. Der Minimalwert für 1 Element eines beliebig großes Arrays wird dadurch ermittelt, das die Zeit der Gesamten Sortierung durch die Anzahl an Elementen geteilt wird.

$$Sort\ (n) = Sortierung\ für\ 1\ Element = \frac{Zeit\ der\ Sortierung}{Anzahl\ der\ Elemente(n)}$$

Das Minimum für die Anzahl der Elemente wird dadurch bestimmt, dass der Minimalwert von n zu n+1 übernommen wird.

$$Minimum(n, n+1) = Minimum(Sort(n), Sort(n+1))$$

Analog dazu der Maximalwert. Anschließend wird die Differenz von beiden gebildet was dem Wert der Versteckten konstante darstellen soll.

$$Konstante = Maximum(n, n+1) - Minimum(n, n+1)$$

3)

Um den Nachteil von MergeSort bei kleineren n's festzustellen wird die Zeit zum Sortieren von 10 Elementen bis 200 Elementen ermittelt und mit anderen Sortierverfahren verglichen. Ab ~140 Elementen war MergeSort schneller als die Konkurrenz.

# Aufgabe 4.3.1

## NFA

**Attribute: NFA**

Pos currentPos

Pos start

List<Pos> states

List<Pos> endStates

List<Edge> edges

**MethodenSignatur: NFA**

Public void AddState(String Name, Boolean End)

Public void AddEdge(int ID, String Value, Pos SourcePos, Pos TargetPos)

Public void DeleteState(String pos)

Public void DeleteEdge(int edge)

Public void StartPosIsEndPos(Boolean)

Public Pos EingabeSequenz(String)

**GET-Methoden**

public Pos getStart()

public List<Pos> getEndStates()

public List<Pos> getAllStates()

public List<Edge> getAllEdges()

public Liste<Edge> getEdgesOfPos(Pos)

Praktikum 2 Team 9: Julian Mechow, Christopher Wolfarth, Patrick Höling

## Edge

**Attribute: Edge**

Int id

String value

Pos target

Pos source

**MethodenSignatur: Edge**

Public Pos getSource()

Public Pos getTarget()

Public String getValue()

Public int getID()

## Pos

**Attribute: Pos**

String name

Boolean endstate

Boolean startstate

**MethodenSignatur: Pos**

Public void SetEndState(Boolean)

## Beschreibung der ADT's

Es können mit Public void AddState(String Name, Boolean End) Zustände in den Automaten hinzugefügt werden. Es wird der Name des Zustandes übergeben und ob es ein Endzustand ist.

Es wird zu Beginn genau ein Startzustand vom System erstellt, dieser kann nicht gelöscht werden und es können keine weiteren Startzustände erstellt werden.
Es ist möglich den Startzustand zu einem Endzustand zu machen und dies wieder rückgängig zu machen. Dazu dient die Methode Public void StartPosIsEndPos(Boolean).

Mit Public void AddEdge(int ID, Char Value, Pos SourcePos, Pos TargetPos) kann man Kanten in den Automaten hinzufügen, der Kante wird eine Nummer übergeben, um diese eindeutig zu identifizieren. Außerdem gibt man den Zustand an, an der die Kante beginnt und den Zustand zu dem sie führt und der Wert der für diesen Übergang gelesen werden muss.

Es ist ebenfalls möglich erstellte Kanten und Zustände wieder zu entfernen. Dafür gibt es die Methoden Public void DeleteState(String pos) und Public void DeleteEdge(int edge).

Dem Automaten können Sequenzen in Form eines Strings übergeben werden, welche er dann mit den vorhandenen Zuständen und Kanten abarbeitet und am Ende gibt er den Zustand zurück, in dem

er sich befindet. Dafür gibt es die Methode Public Pos EingabeSequenz(String).
Der Automat liest zeichenweise ein.

## Vor- und Nachbedingungen

**Public void AddState(String Name, Boolean End)**

Vorbedingung:
Der Name des Zustandes ist nicht bereits vergeben.

Nachbedingung:

Erstellt einen Zustand und Speichert ihn in der Liste aller Zustände ab.

**Public void AddEdge(int ID, Char Value, Pos SourcePos, Pos TargetPos)**

Vorbedingung:
Die angegebenen Zustände existieren in der Liste aller Zustände.
Die ID der Kante ist nicht bereits vergeben

Nachbedingung:

Erstellt eine Kante und speichert Sie in der Menge aller Kanten ab.

**Public void DeleteState(String pos)**

Vorbedingung:
Der übergebene Zustand existiert in der Liste aller Zustände.

Nachbedingung:

Der übergebene Zustand wird aus der Liste aller Zustände entfernt.

**Public void DeleteEdge(int edge)**

Vorbedingung:
Die übergebene Kante existiert in der Liste aller Kanten.

Nachbedingung:

Die übergebene Kante wird aus der Liste aller Kanten entfernt.

**Public void StartPosIsEndPos(Boolean)**

Vorbedingung:
-

Nachbedingung:

Der Startzustand wird bei „true“ zusätzlich ein Endzustand und bei „false“ ist er nur Startzustand.

**Public Pos EingabeSequenz(String)**

Vorbedingung:
Es müssen Zustände existieren.

Nachbedingung:

Der übergebene String wird abgearbeitet und der Zustand, welcher durch das abarbeiten des String erreicht wurde wird zurückgegeben.

## Tests

1. Das System gewährleistet, dass es zu jeder Zeit nur genau einen Startzustand geben kann. Dies muss überprüft werden.
2. Positiv-Tests für Automaten und Eingaben
3. Negativ-Tests für Automaten und Eingaben
4. Nach der Abarbeitung einer Sequenz muss ein Endzustand erreicht werden.