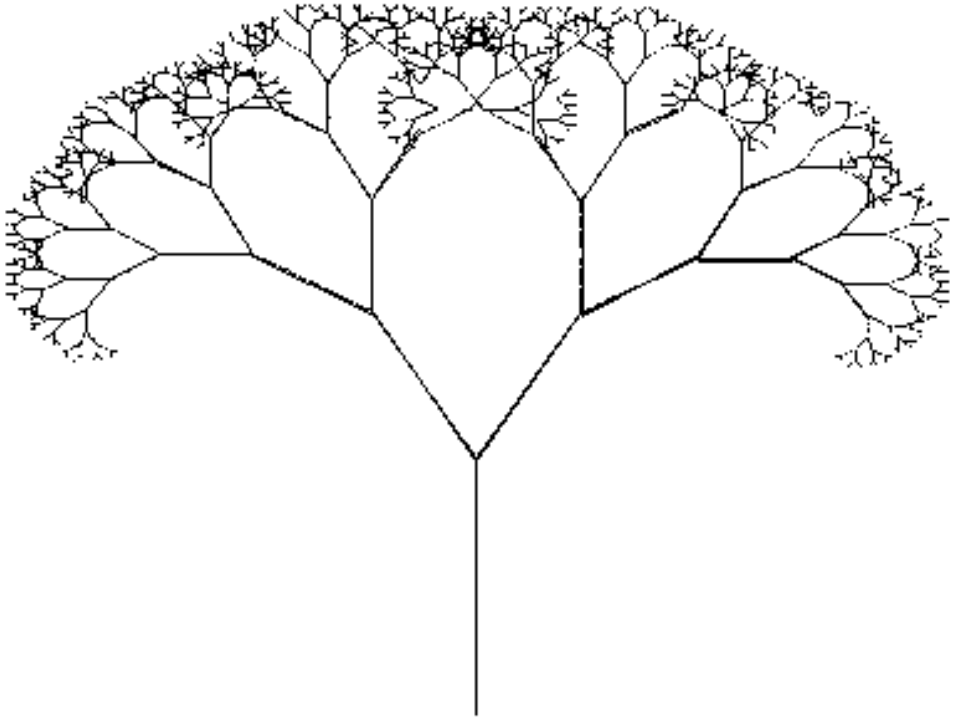
Zeichnen von Graphen

graph drawing

WS 2006 / 2007



Gruppe: **D_rot_Ala0607**

Christian Becker	11042315
Eugen Plischke	11042351
Vadim Filippov	11042026

Problemstellung

Gegeben sei ein Graph G = (V; E)

V = Knotenmenge
E = Kantenmenge

Gesucht wird eine Zeichnung d(G), so dass eine leicht verständliche bildliche Darstellung den Graph optimal visualisiert.

Durch die Visualisierung will man die sehr guten kognitiven menschlichen Fähigkeiten ausnutzen, denn Bilder kann man sich leichter merken als Formeln und formale Beschreibungen.

Folgende Anforderungen werden an die Zeichnung gestellt:

- "ästhetisch" schön
- übersichtlich
- verständlich (für den Betrachter)
- regelmäßig (*Symmetrien, Ähnlichkeiten)*

Es existieren also zwei Sichten:

1. geometrische Sicht
2. graphische Sicht

Wie erreicht man die Anforderungen, welche an die Zeichnung von G gestellt werden?

Viele Algorithmen beschäftigen sich mit genau dieser Fragestellung.

Im Folgenden werden wir zwei Algorithmen zum zeichnen von Binärbäumen genauer untersuchen und klären, in wie fern die gegebenen Anforderungen erfüllt werden.
Als erstes betrachten wir einen weniger komplexen Algorithmus und im Anschluss werden wir den **Reingold-Tilford** Algorithmus ausführlich behandeln.

Wir betrachten immer binäre Bäume.

Ein einfacher Algorithmus zum zeichnen eines Binären Baumes

"How shall we draw a tree" fragt D.E. Knuth schon 1970 in seiner "Computerbibel".

D.E. Knuth
The Art of Computer Programming, Vol .1 Fundamental Algorithms
p. 306, Addison-Wesley, Reading, 1970

Knuth war seiner Zeit (dem Graph Drawing) voraus. Damals gab es keine Graphikbildschirme und Zeichnungen wurden im Textmodus gemacht mit Buchstaben (Punkte) für die Knoten "/" und "\" zur Andeutung von Kanten.

Knuth entschied:

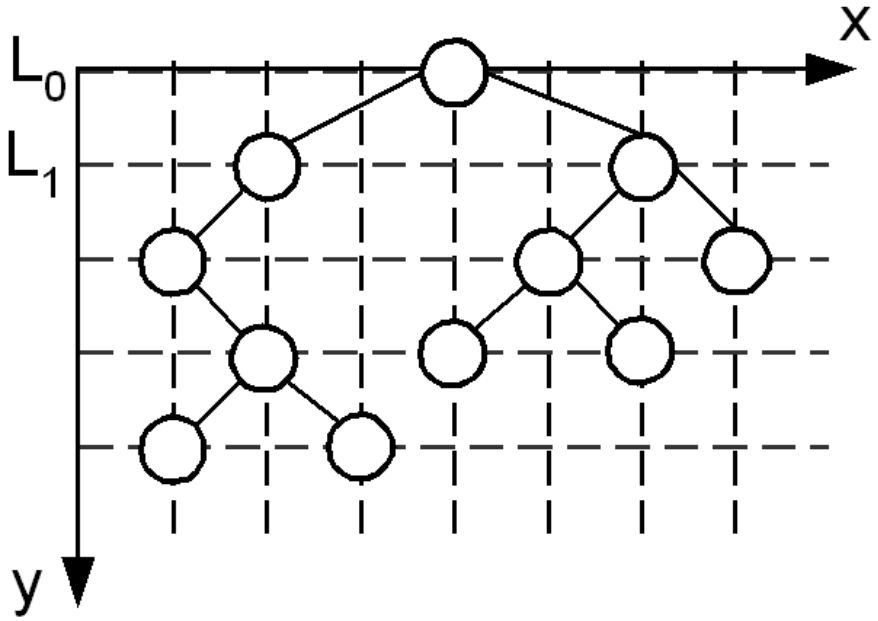
Bäume zeichnet man von oben nach unten ...weil wir so schreiben.

und entwickelte primitive Algorithmen zum Zeichnen von Bäumen.

Unser erster Algorithmus folgt ebenso dem Grundsatz, dass Bäume von oben nach unten gezeichnet werden. Die Bestimmung der y- Komponente eines jeden Knotens ist daher einfach:

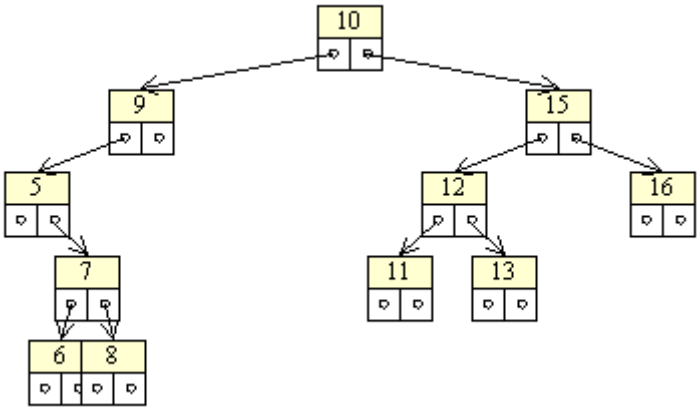
Jeder Knoten v der Tiefe i erhält den y-Wert y(v) = -i

Die Zeichnung ist also strikt nach unten gerichtet.



Da die y- Koodinaten durch die Baumstruktur trivial sind, müssen „nur“ noch die entsprechenden x- Koordinaten bestimmt werden.

Hierfür benutzen wir im Folgendem **den Rang** von v der Inorder-Travesierung.
Als Beispiel nehmen wir folgenden Baum:



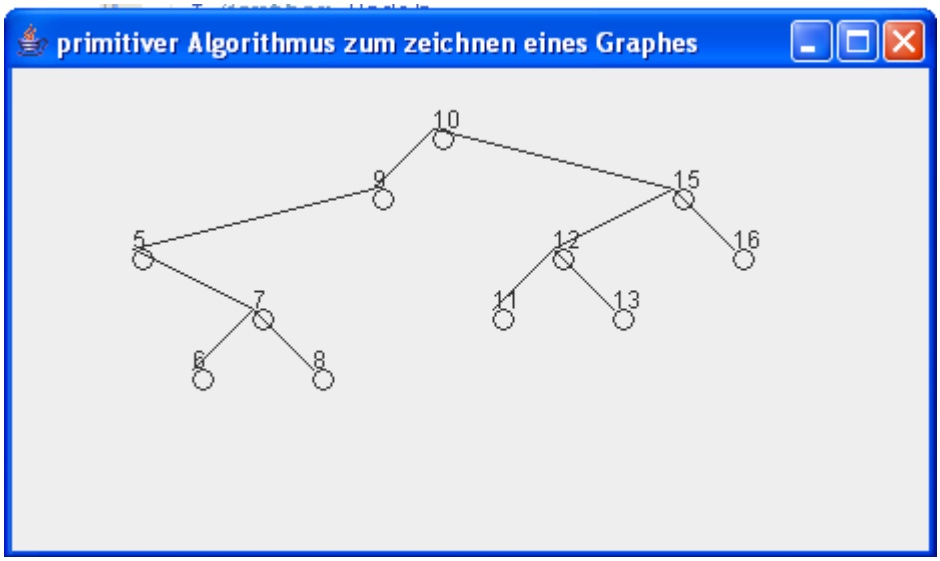
Die Inorder-Travesierung hatte das Ergebnis:

5 – 6 – 7 – 8 – 9 – 10 – 11 – 12 – 13 – 15 – 16

Die x-Koordinaten sind daher:

X(5) = 1
X(6) = 2
X(7) = 3
..... usw.

Dieser Algorithmus wurde von uns in Java implementiert.



Fazit:
Die Methode führt zu Ästhetisch nicht ansprechenden Bäumen, da die Kanten „viel zu lang“ sind und die Väter nicht „mittig“ über ihren Kindern platziert sind. Jedoch kommt es bei diesem einfachen Algorithmus nicht zu Kollisionen und man kann die Struktur des Baumes erkennen.

Aussicht:
Als nächstes betrachten wir einen Algorithmus, der mit Hilfe von Divide & Conquere den Baum von oben nach unten zeichnet. Dabei erreichen wir, dass die Breite des Baumes minimiert wird und ästhetisch ansprechende Bäume entstehen.

Reingold-Tilford-Algorithmus

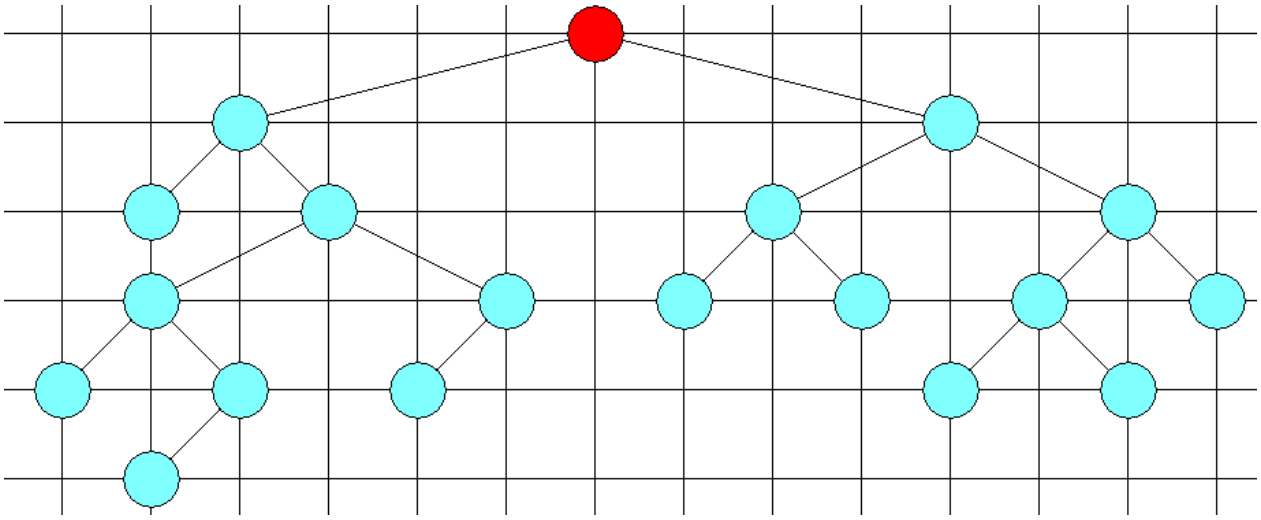
Der Reingold-Tilford-Algorithmus (RT) ist ein typischer Divide & Conquer Algorithmus.
Als Eingabe erhält er einen binären Baum T und als Ausgabe erhält man eine Zeichnung von T, welche bereits sehr übersichtlich ist und viele der gegebenen Ästhetikkriterien erfüllt.

Wir können zwei Fälle unterscheiden:

1. Es gibt nur einen Koten: Die Zeichnung ist trivial definiert.
2. Wenn es mehr als einen Knoten gibt: Wende Divide & Conquer an.

Um das Verfahren, womit die x-Koordinaten berechnet werden, besser verstehen zu können betrachten wir die Einzelnen Schritte an einem Beispiel.

Gegeben sei folgender, noch nicht gezeichneter Baum:



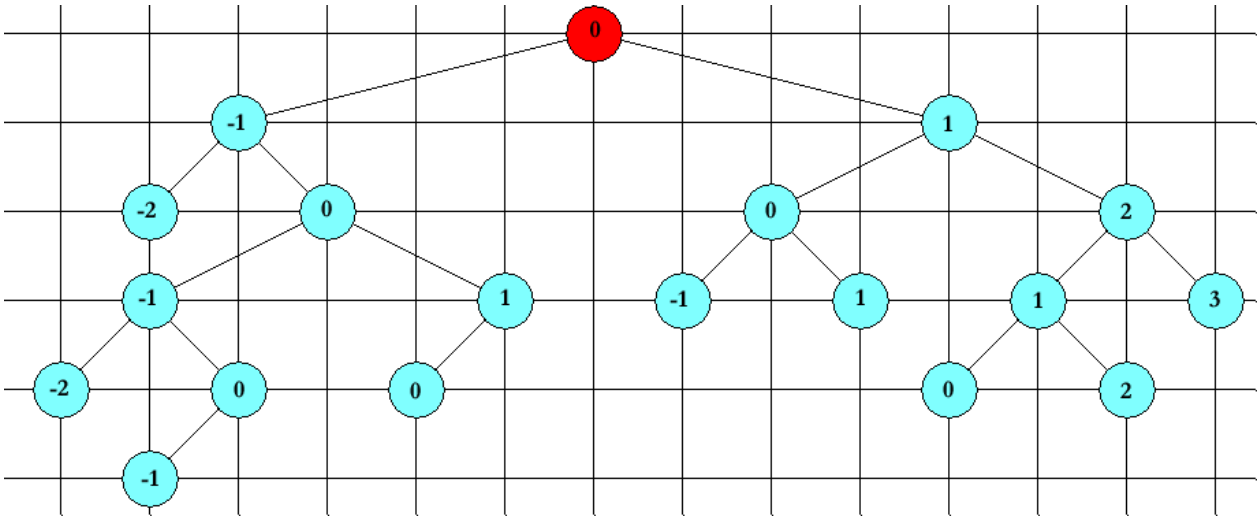
Dieser binäre Baum ist zu dem jetzigen Zeitpunkt lediglich eine **Speicherstruktur**.
Die y- Koordinaten eines jeden Knotens werden wie in den vorherigen Beispielen berechnet (die jeweilige Ebene des Knotens ist der entsprechende y- Wert).

Um mit dem RT- Algorithmus die X- Werte bestimmen zu können, werden für jeden Knoten Startwerte als X-Wert vergeben.

Dies geschieht nach der Regel:

1. Beginne beim Wurzelknoten und bewerte diesen mit 0
2. Der linke Sohn bekommt den Wert -1 des Vaters und der rechte Sohn +1 des Vaters
3. Wiederhole Schritt 2 bis keine Kinder mehr da sind.

Jeder Knoten hat nun einen x- Wert zugewiesen bekommen. Für unser Beispiel erhalten wir als Ergebnis:



Das dies noch nicht die Endgültigen Koordinaten bzw. x- Komponenten sind, ist trivial, da es so zu unzähligen Kollisionen kommen würde. Die x- Koordinate 0 wurde zum Beispiel mehrmals vergeben.

Bevor wir jetzt die nächsten Schritte machen können, bedarf es noch einigen Definitionen:

Definition Linksumriss:

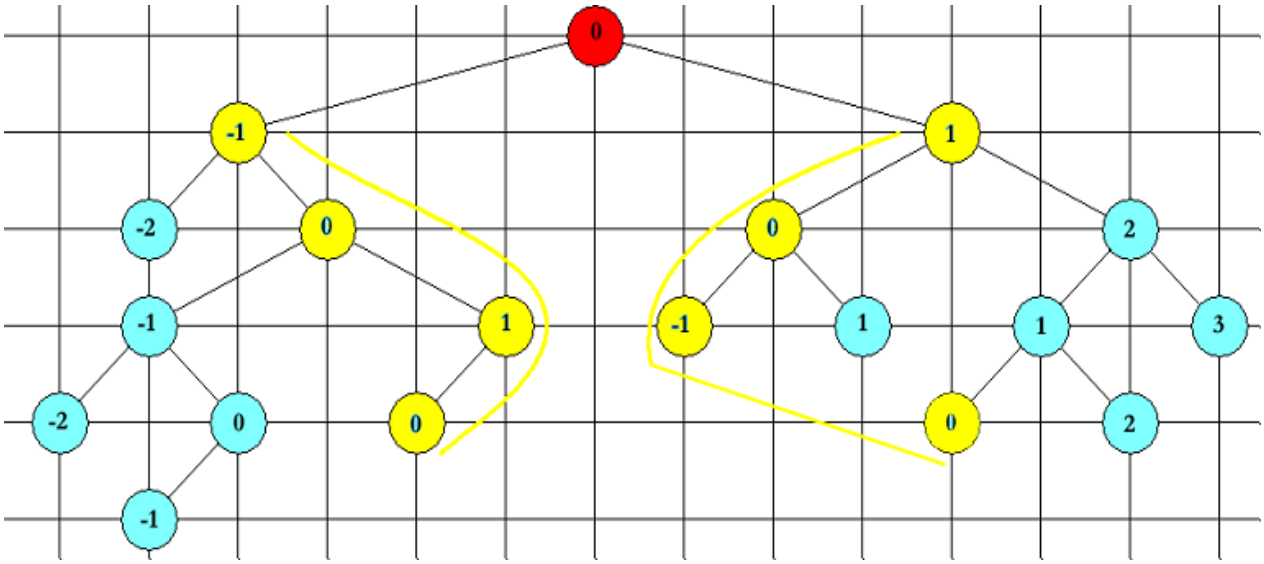
Die linken äußeren Knoten eines Teilbaumes heißen Linksumriss, dabei muss folgende Ausnahme beachtet werden: Ist die Höhe des linken Teilbaumes größer als die Höhe des rechten Teilbaumes, dann besteht der Linksumriß des linken Teilbaumes zusätzlich aus dem Teil des Linksumrisses des rechten Teilbaumes, welcher bei der Höhe des linken Teilbaumes **+ 1** beginnt.

Die Definition des Rechtsumrisses ist analog zu der des Linksumrisses.

Diese Definition ist von sehr großer Bedeutung für den RT-Algorithmus, da später andauernd mit den Umrisslisten gerechnet wird.

Zum besserem Verständnis wenden wir die Definition an einem Beispielbaum an:

Der Linksumriss des rechten Teilbaumes bzw. der rechts Umriss des linken Teilbaumes des Wurzelknotens ist in unserem Beispiel:

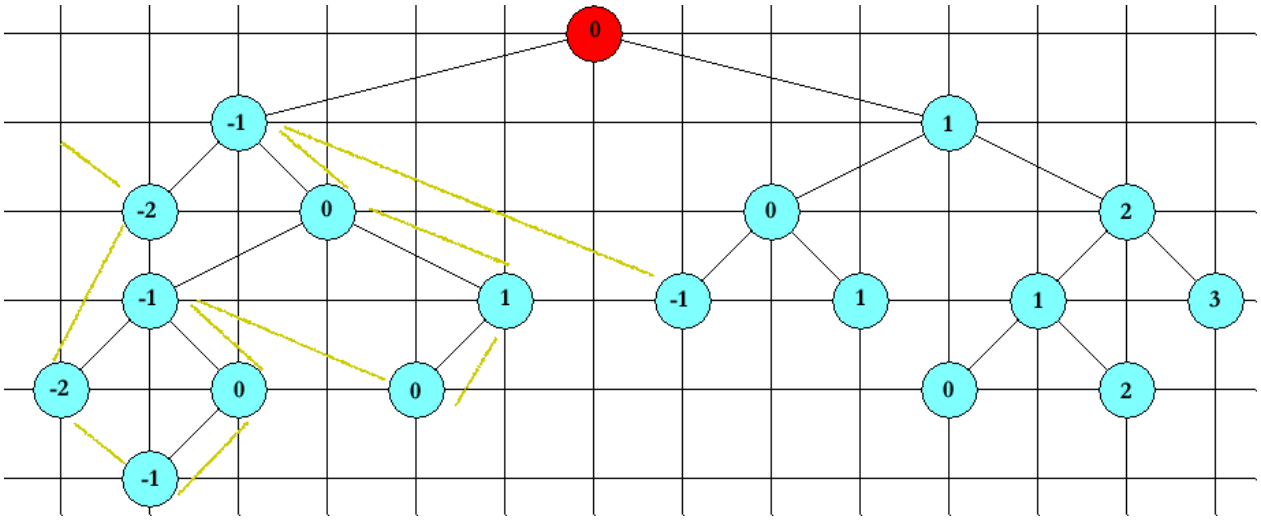


Die Bewertung der einzelnen Knoten ist für das Beispiel nicht von Interesse. Es wird lediglich gezeigt nach welchem Schema die Umrisse gebildet werden.

Wiederholung Postorder- travesierung:

Es gibt verschiedene Möglichkeiten, die Knoten von Binärbäumen zu durchlaufen. Wird der Baum post-order travesiert, so wird zuerst der linke (L), dann der rechte (R) Teilbaum durchlaufen und anschließend wird die Wurzel (W) betrachtet.

Beispielsweise wäre der Postorder durchlauf für unseren Baum:



Wir benötigen noch eine Formel, mit der wir bestimmen können, um welchen Faktor zwei Teilbäume verschoben werden.

Diese Formel lautet:

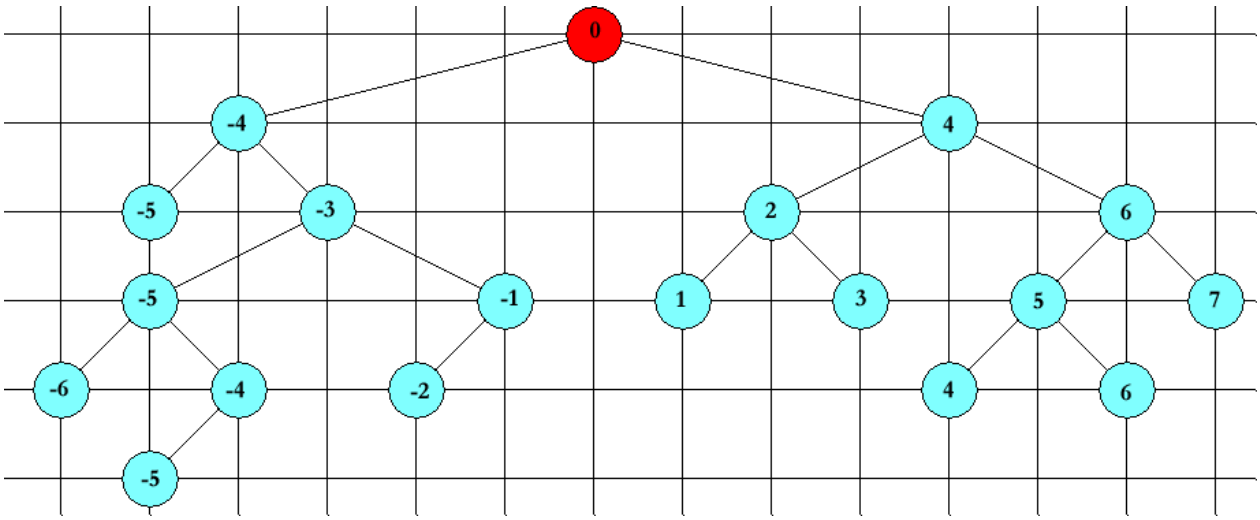
$$\mathbf{Diff_{lr} = [WERT(l) - WERT(r)] + 2}$$

Ist zum Beispiel WERT(l) = 1 und WERT(r) = -1 ,dann ist $\mathbf{Diff_{lr} =[1 – (-1)] + 2 = 4}$.
Jeder der beiden Teilbäume (**l, r**) muss um **2** Einheiten nach links bzw. rechts verschoben werden. Wenn die Differenz gleich **0** ist, ist keine Anpassung notwendig. Wenn die Differenz einen ungeraden Wert hat, wird sie inkrementiert, damit der rechte/linke Kollisions-Knoten jeweils um den gleichen Betrag (½ $\mathbf{Diff_{lr}}$) nach rechts/links verschoben werden kann.

Nachdem wir nun alle Begriffe erläutert haben, ist es nicht so kompliziert den RT- Algorithmus zu verstehen:

Für jeden Knoten (in Postorder Reihenfolge) berechnen man den Links- und Rechtsumriss.
Wir gehen solange alle Knoten der Umrisse ebenenweise durch, bis auf beiden Seiten keine Knoten mehr vorhanden sind.
Für jede Ebene berechnen wir nach der obigen Formel die Differenz und wählen das Maximum der Ebenen aus. Das ist nun der Wert, um die Hälfte dessen wir die jeweiligen Teilbäume nach rechts bzw. links verschieben.

Wenden wir dieses Verfahren auf unseren Beispielbaum an, so erhalten wir für die x-Koordinaten:

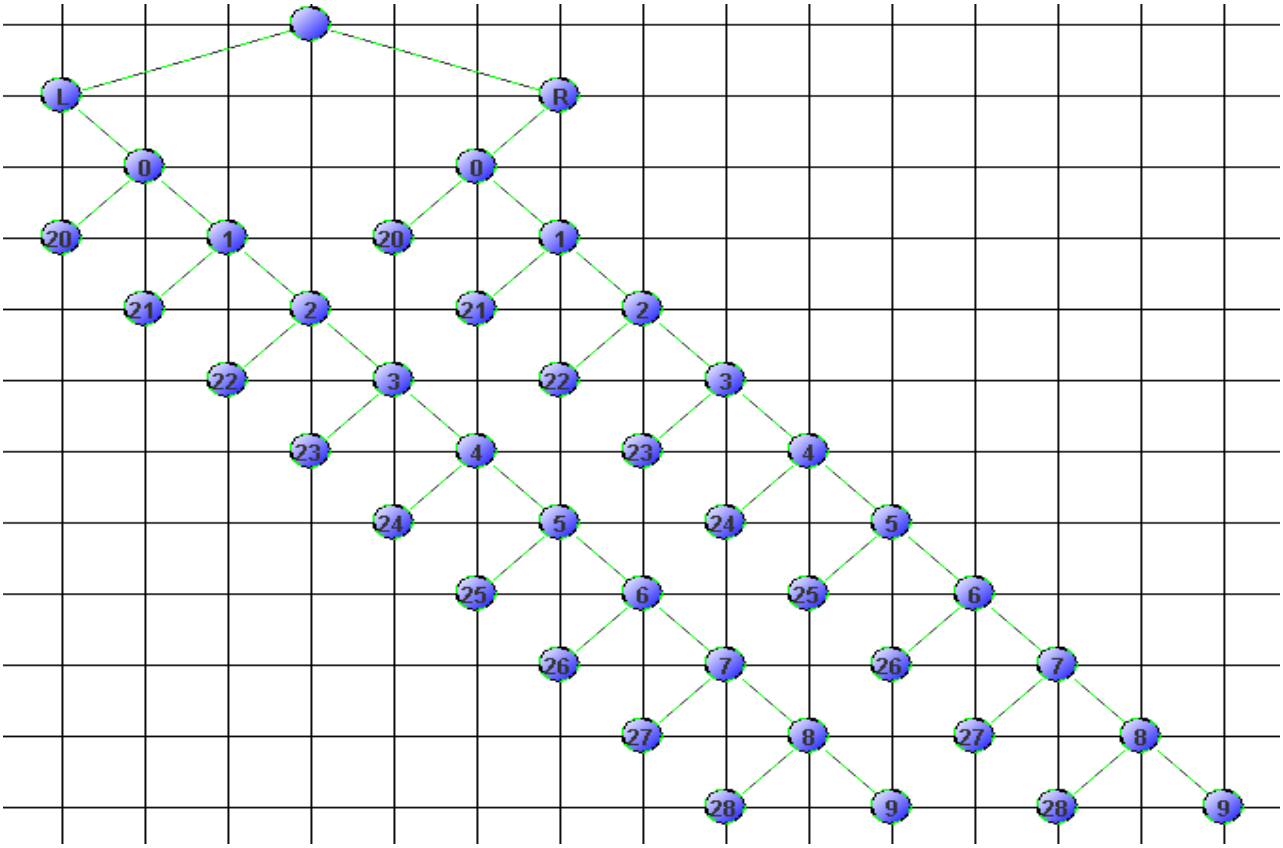


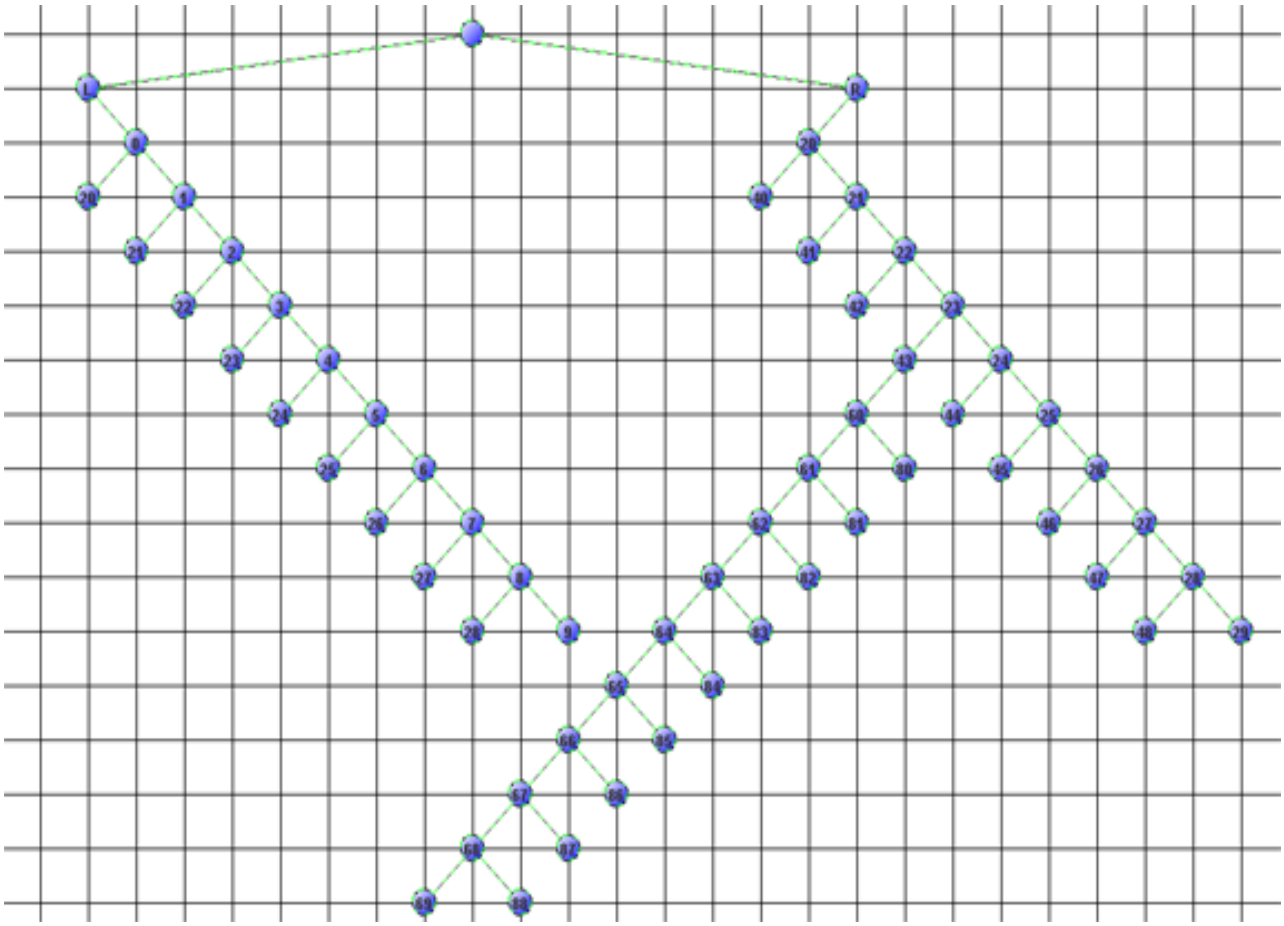
Man erkennt deutlich, dass dieses Verfahren einen kollisionsfreien Baum erstellt und das die Ästhetikkriterien erfüllt werden.

Um den Baum jetzt zeichnen zu können, bestimmen wir noch den linksten Knoten (-6)und bestimmen seinen Wert dem Betrage nach und addieren diesen Wert zu allen anderen Knoten dazu.
Dies hat lediglich den Vorteil, dass keine negativen x-Werte vorhanden sind.

Implementierung

Wir haben den Algorithmus in Java vollständig implementiert und können nun verschiedene binäre Bäume zeichnen.
Im Folgendem einige mit unserem Programm gezeichnete Bäume:





Wir gehen an dieser Stelle **nicht** genauer auf die Implementierung ein. Bei Interesse schicken wir gerne den gesamten gut dokumentierten Quellcode zu. Hierzu bitte einfache eine E-Mail an: hangall@gmx.de schicken.

Betrachtung der Laufzeit

Eine wichtige Beobachtung für die Effizienz des Algorithmus ist die, dass die Umriss-Listen von T‘ und T“ nur bis zur Höhe des jeweils kleineren Teilbaumes verfolgt werden müssen. Damit ist die Zeit für die Verarbeitung des Knotens v beim Postorder-Traversieren proportional zum Minimum der beiden Höhen von T‘ und T“.

Daher gilt für den jeden einzelnen Knoten:

$$(1 + \min\{h'(v), h''(v)\})$$

Betrachtet man alle Knoten, so erhält man folgende Formel:

$$\sum_{v \in \Omega} (1 + \min\{h'(v), h''(v)\}) = n + \sum \min\{h'(v), h''(v)\}$$

Laufzeit für Zusammenfügungsoperationen (n Anzahl Knoten in T(v)):

h(v) := Höhe(T(v)) + 1
hl(v) := Höhe(Tl) + 1
hr(v) := Höhe(Tr) + 1

F(T(v)): Zeit für Baum T(v).

Es gilt $F(T(v)) = F(T_l) + F(T_r) + \min\{(v_r); h_l(v)\}$

Behauptung:

F(T(v)) = n - h(v).

Beweis durch Induktion:

n = 0: F(T(v)) = 0 - 0 = 0
n = 1: F(T(v)) = 1 - 1 = 0

Sei die Behauptung korrekt für Bäume mit k < n Knoten. Für einen Baum T(v) mit n Knoten hat T_l k < n Knoten und T_r n - k - 1 < n Knoten.

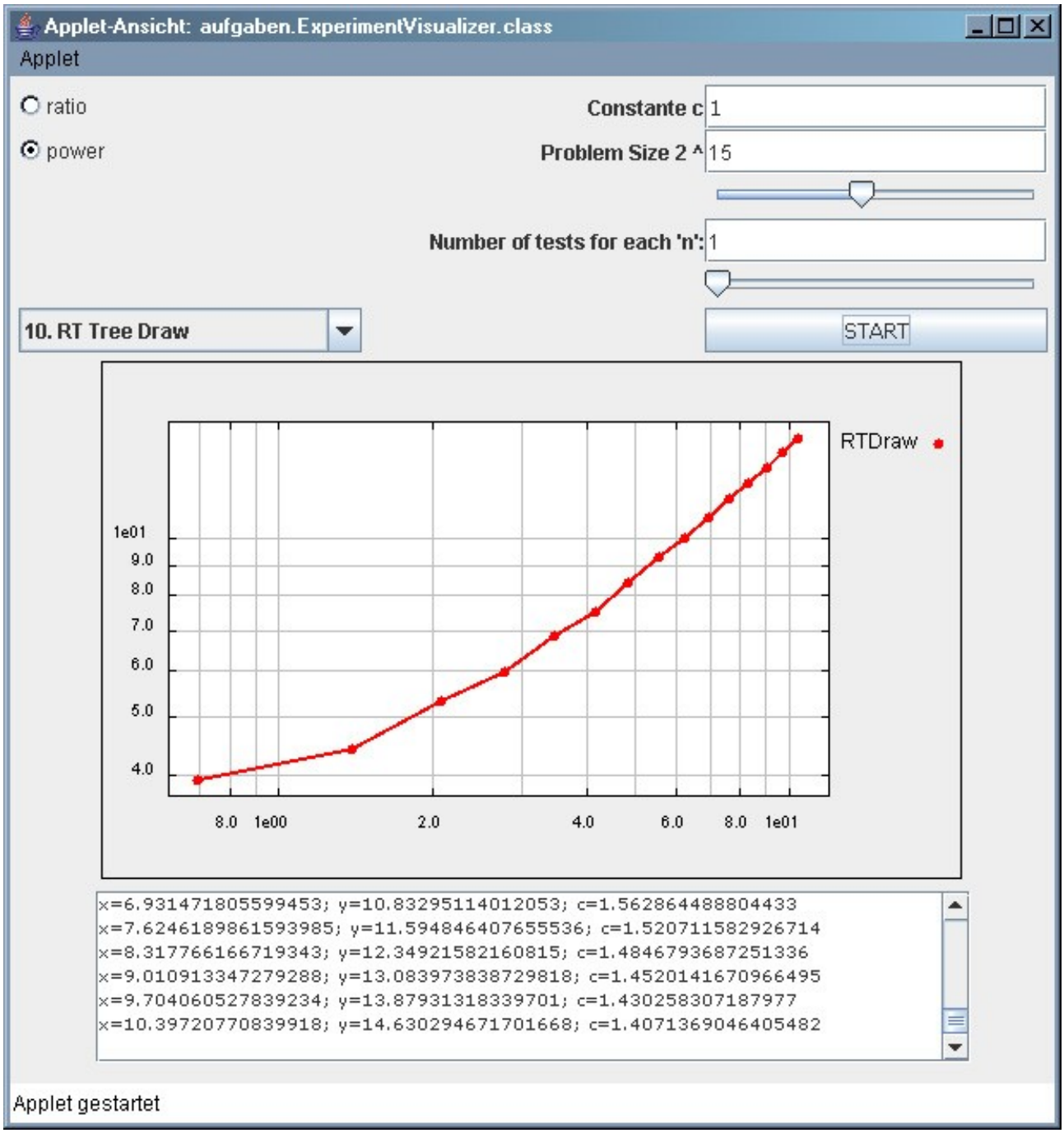
$$
\begin{aligned}
F(T(v)) &= F(T_l) + F(T_r) + \min\{h_l(v), h_r(v)\} \\
&= k - h_l(v) + (n - k - 1) - h_r(v) + \min\{h_l(v), h_r(v)\} \\
&= n - 1 - h_l(v) - h_r(v) + \min\{h_l(v), h_r(v)\} \\
&= n - (\max\{h_l(v), h_r(v)\} + 1) \\
&= n - h(v)
\end{aligned}
$$

Postorder, Preorder: O(n) Insgesamt Zeit O(n).

Experimentelle Analyse

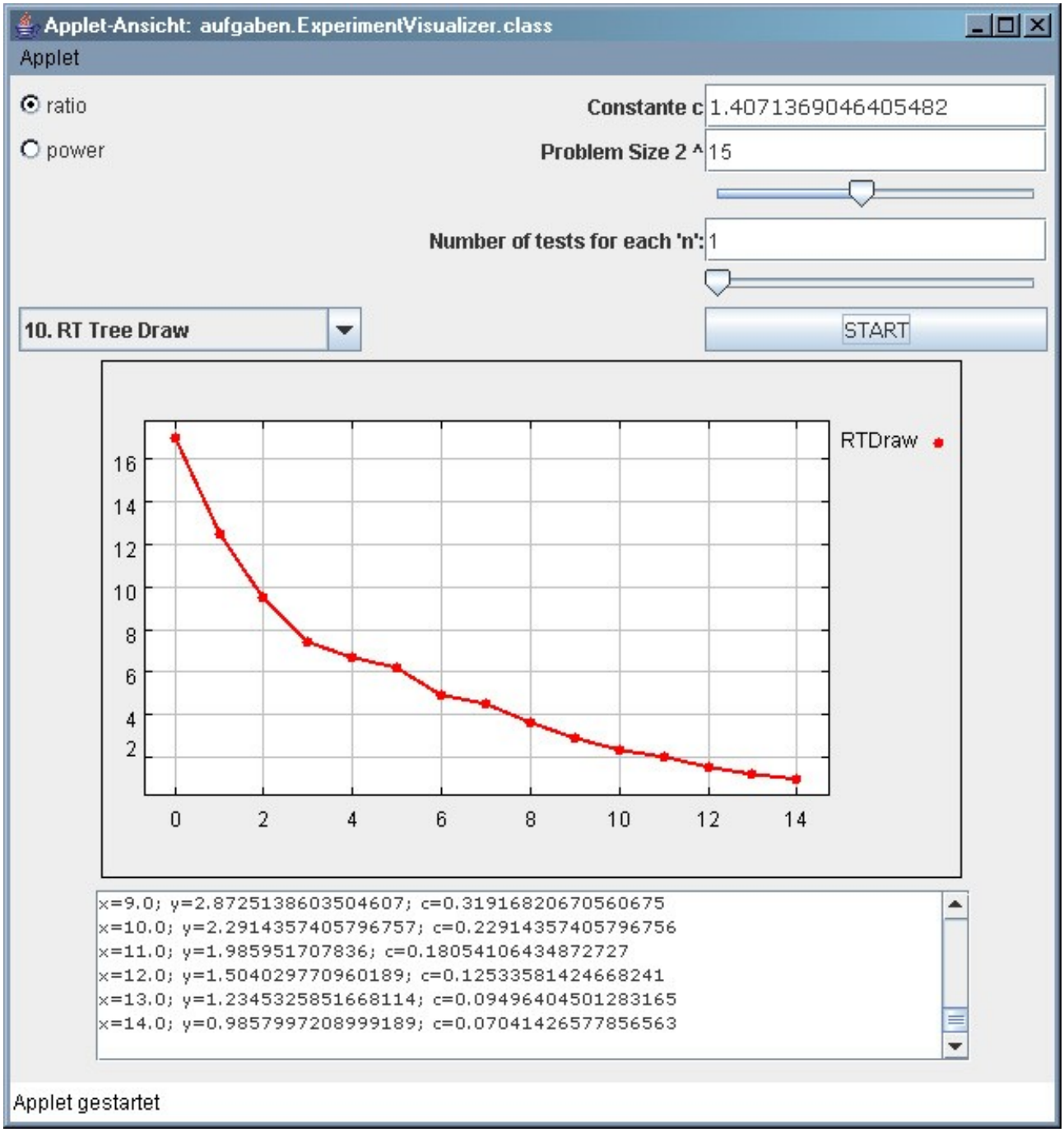
Die Laufzeit unserer Implementierung haben wir auch experimentell getestet.
Dazu benutzen wir zwei durch die Vorlesung Algorithmik bekannten Testverfahren: Power- und Ratiotest.

Wir testen einen zufällig generierten Baum mit 2^{15} Knoten.



Die mit Hilfe des PowerTests errechnete Konstante c ist c = 1,4071369

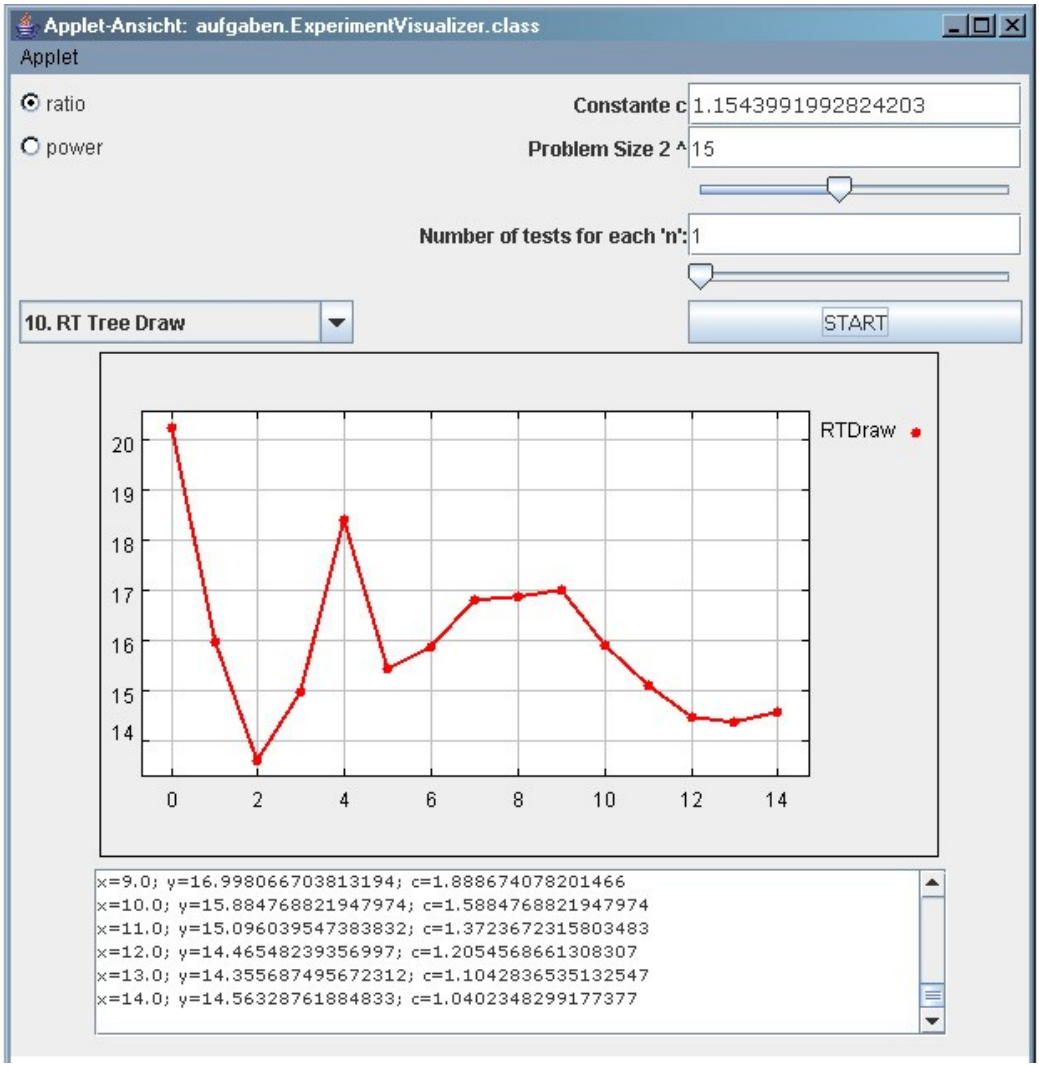
Setzen wir diese Konstante c in den Ratiotest ein erhalten wir:



Die Kurve läuft gegen 0. Die Konstante ist also zu groß gewählt.

Durch ausprobieren finden wir den richtigen Wert für c, nämlich: c =1,1543991992824203

Der Ratiotest für dieses c:



Die experimentelle Analyse zeigt eine Laufzeit von O(n^c) = O(n^1.15), diese Unterscheidet sich von der errechneten Laufzeit von O(n). Dies könnte an eine nicht optimale Implementierung zurückgeführt werden.