Leichter verständlich ist die *rekursive Darstellung:*

```
procedure SucheRek (p: in Ref_BinBaum; s: Integer;
                                    q: out Ref_BinBaum) is
begin
    if p /= null  then
       if p.Inhalt > s then SucheRek(p.L, s, q);
       elsif p.Inhalt < s then SucheRek(p.R, s, q);
       else q:=p;
       end if;
    end if;
end Suche;
```

Verwendung dieser Prozedur: Ergebniszeiger := null;
SucheRek(Anker, Schlüssel, Ergebniszeiger);
if Ergebniszeiger /= null then .... else ... end if;

3.2.2.10.b Einfügen in einen Binärbaum

Prinzipiell werden alle neuen Elemente in Blätter des Baums eingetragen.

```
procedure Einfügen (Anker: in out Ref_BinBaum; s: Integer) is
p, q: Ref_BinBaum := Anker;
begin
    if  p = null  then  Anker := new BinBaum'(s, null, null);
    else
        while p /= null loop  q:=p;
            if p.Inhalt > s then p := p.L; else p := p.R; end if;
        end loop;
        if q.Inhalt > s then q.L := new BinBaum'(s, null, null);
        else q.R := new BinBaum'(s, null, null); end if;
    end if;
end Suche;
```

Aufgabe: Schreiben Sie für diese Operation eine rekursive Prozedur.

3.2.2.10.c Löschen in einem Binärbaum

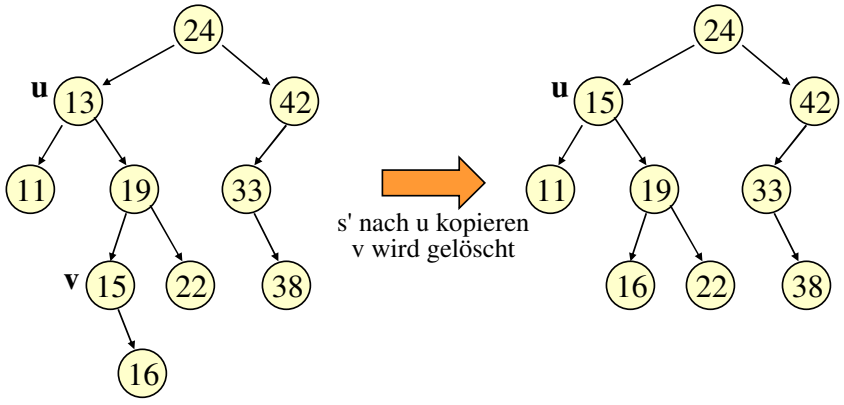
Da ein Knoten zwei Nachfolger, aber nur einen Vaterknoten hat, kann man ihn nicht einfach löschen. Prinzipiell werden daher nur solche Knoten gelöscht, die mindestens einen leeren Unterbaum besitzen.

Soll der Schlüssel s gelöscht werden, so sucht man den zugehörigen Knoten u (nach 3.2.2.10a). Besitzt dieser Knoten zwei echte Kinder, so geht man zum "Inorder-Nachfolger", also zu dem Knoten v im Baum, dessen Inhalt der auf s folgende Schlüssel s' in der sortierten Reihenfolge der Knoteninhalte ist.

v ist der linkeste Knoten im rechten Unterbaum von u.

In den Knoten u schreibt man nun s' und löscht den Knoten v, wobei dessen eventueller rechter Unbaum an den Vater von v gehängt wird.

Skizze: Lösche s=13



s'= 15

Löschen in einem Binärbaum

```
procedure Löschen (Anker: in Ref_BinBaum; s: Integer) is
u, v: Ref_BinBaum;
begin  Suche(Anker, s, u);   -- siehe 3.2.2.10a
    if  u /= null  then
        if (u.L = null) or (u.R = null) then < lösche u > ;
        else                          -- Suche den Inorder-Nachfolgerknoten v
            v := u.R;
            while  v.L /= null  loop  v := v.L; end loop;
            u.Inhalt := v.Inhalt;          -- s' wird nach u kopiert
            < lösche v > ;
        end if;
    end if;
end Löschen;
```

Hier ist noch ein Problem: Für < lösche u > und <lösche v> muss man den jeweiligen Vater von u bzw. v kennen. Also muss die Prozedur Löschen modifiziert werden.

Hierfür lässt man einen Zeiger "vater" mitlaufen, der auf den zuvor betrachteten Knoten zeigt:

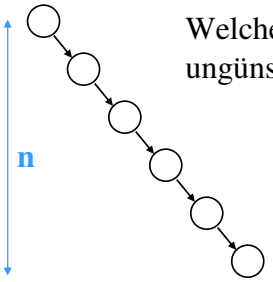
```
procedure Löschen (Anker: in Ref_BinBaum; s: Integer) is
u, v, vater: Ref_BinBaum; links: Boolean;
begin  "Suche(Anker, s, u, vater, links);"
    -- dies ist neu zu programmieren: vater zeigt auf den Vater von u (sofern
    -- vorhanden) und links ist true, falls u linker Sohn von vater ist
    if  u /= null  then
        if (u.L = null) or (u.R = null) then < lösche u > ;
        else                          -- Suche den Inorder-Nachfolgerknoten v
            v := u.R; vater := u;
            while  v.L /= null  loop  vater := v; v := v.L; end loop;
            u.Inhalt := v.Inhalt;          -- s' wird nach u kopiert
            < lösche v > ;
        end if;
    end if;
end Löschen;
```

Bedeutet < lösche v > nun einfach:
vater.L := v.R; ? (Vorsicht: Fehler!)
Wie müssen < lösche u > und Suche programmiert werden?? Siehe Übungen!

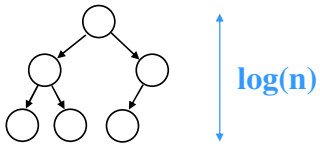
Welchen Aufwand erfordern die Operationen Suchen, Einfügen und Löschen, wenn der Suchbaum n Knoten besitzt?

Man misst diesen Aufwand meist in der Anzahl der Vergleiche, die erforderlich sind, um die Operation durchzuführen.

Suchen, Einfügen und Löschen: Im schlechtesten Fall benötigt man jeweils t Vergleiche, wobei t die Tiefe des Baumes ist; zur Definition siehe 3.2.2.1 (11).



Welche Beziehung herrrscht zwischen n und t? Im ungünstigsten Fall ist t = n, im günstigen ist t = log(n).



Ein Baum kann also zu einer Liste "entarten" und man braucht dann entsprechend viele Vergleiche.

Besonders günstig ist dagegen ein Baum, bei dem jeder Knoten höchstens das Level log(n+1) besitzt.

Was genau ist hier "log" (der Logarithmus zur Basis 2)? Wir benötigen ihn als eine Funktion zwischen natürlichen Zahlen und modifizieren daher die üblicherweise über den positiven reellen Zahlen definierte Logarithmusfunktion wie folgt:

Definition 3.2.2.11: Diskreter Zweierlogarithmus. Wenn nicht anders vermerkt sei in Zukunft log: $\mathbb{N} \rightarrow \mathbb{N}_0$ definiert durch
$\log(1) = 0$ und für $n \geq 2$:
$\log(n) = k$ für das eindeutig bestimmte k mit $2^{k-1} < n \leq 2^k$.

Es gilt dann also log(2)=1, log(3)=log(4)=2, log(5)=3 usw.

Folgerung 3.2.2.12:
log sei wie in 3.2.2.11 definiert, dann gilt für die Tiefe t jedes Suchbaums mit n Knoten $\log(n+1) \le t \le n$ für alle $n > 0$.

Beweis: $t \le n$ ist klar, da die Länge jedes doppelpunktfreien Wegs in einem Baum mit n Knoten durch n beschränkt ist.

Ein Baum der Tiefe t kann höchstens 2^t-1 Knoten besitzen, wie man durch Induktion leicht sieht: Für k=1 ist dies richtig, und wenn Bäume der Tiefe k höchstens 2^k-1 Knoten enthalten, dann kann ein Baum der Tiefe k+1 höchstens "Wurzel plus zwei Unterbäume der Tiefe k", also $1+2^k-1+2^k-1=2^{k+1}-1$ Knoten haben.
Folglich ist $n \le 2^t-1$, also $\log(n+1) \le \log(2^t) = t$.

■

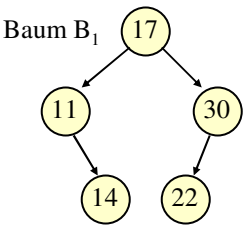
Für die Praxis ist die "mittlere Suchzeit" bei einem Baum wichtig. Hierzu berechnen wir für alle Inhalte eines Baumes, wie viele Vergleiche notwendig sind, um diesen Inhalt zu suchen (dies ist genau das Level des zugehörigen Knotens, vgl. 3.2.2.1 (12)), summieren alle diese Zahlen auf (das Ergebnis heißt auch "innere Pfadlänge") und dividieren anschließend durch die Anzahl n der Knoten.

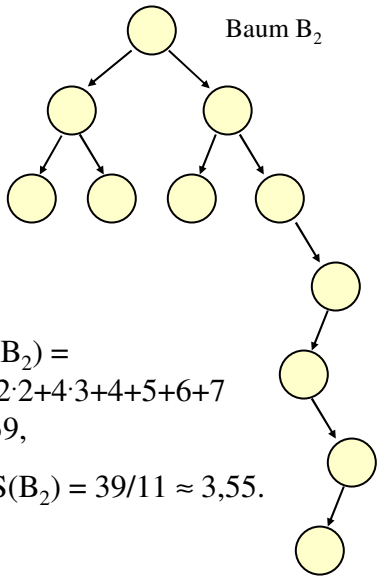
Definition 3.2.2.13: Die innere Pfadlänge eines beliebigen gerichteten Baums mit n Knoten $v_1, ..., v_n$ ist definiert als:

$$Pf(B) = \sum_{i=1}^{n} level(v_i) .$$

Die mittlere Suchzeit eines Baums B mit n Knoten ist dann:
$MS(B) = Pf(B)/n$.

Zwei Binärbäume als Beispiele:





$Pf(B_1) = 1+2+2+3+3 = 11,$

$MS(B_1) = 11/5 = 2{,}2.$

Man sieht, dass Pf und MS von den Inhalten in den Knoten unabhängig sind.

$Pf(B_2) = 1+2\cdot2+4\cdot3+4+5+6+7 = 39,$

$MS(B_2) = 39/11 \approx 3{,}55.$

Berechnen Sie selbst die mittlere Suchzeit für den schlechtesten und für den günstigsten Binärbaum, also für einen zu einer Liste entarteten Binärbaum und für einen Binärbaum mit 2^k-1 Knoten, dessen Blätter alle die Tiefe k haben.

Problem: Mit welcher mittleren Suchzeit muss man rechnen, wenn man zufällig einen Baum mit n Knoten betrachtet?

Lösungsansatz 1: Wir summieren für alle Bäume mit n Knoten alle inneren Pfadlängen auf und dividieren anschließend durch n und durch die Anzahl aller Bäume mit n Knoten.

Dieser Ansatz erweist sich als kompliziert. Wenn man sich aber auf alle Binärbäume beschränkt, so kann man eine Rekursionsformel aufstellen, die man mit ähnlicher Technik löst wie den im Folgenden angegebenen Ansatz. (Selbst versuchen!)

Lösungsansatz 2: Wir betrachte nur Binärbäume. Es sei F(n) der gesuchte mittlere Wert aller internen Pfadlängen für Binärbäume mit n Knoten. Dann gilt $F(0) = 0$ und $F(1) = 1$. Betrachte nun einen Binärbaum mit $n > 1$ Knoten. Dann gibt es eine Wurzel, die einen linken Unterbaum mit i-1 Knoten und einen rechten Unterbaum mit n-i Knoten besitzt für ein i mit $1 \le i \le n$. Ist nun jedes i gleichwahrscheinlich (und dies darf man bei der gesamten Mittelung annehmen), dann erhält man

$$F(n) = (1/n) \cdot ((F(0) + F(n-1) + \text{Erhöhung der Pfadlängen}) + (F(1) + F(n-2) + \text{Erhöhung der Pfadlängen}) + \ldots (F(n-1) + F(0) + \text{Erhöhung der Pfadlängen}))$$

Die "Erhöhung der Pfadlängen" berücksicht die Verlängerung aller Pfade durch die Wurzel. Diese Erhöhung ist aber für *jeden* Knoten 1, d.h. "Erhöhung der Pfadlängen" = n.

Somit erhalten wir die Formeln: $F(0) = 0$ und $F(1) = 1$ und

$$F(n) = (1/n) \cdot ((F(0) + F(n-1) + n) + (F(1) + F(n-2) + n) + \ldots (F(n-1) + F(0) + n)$$
$$= (1/n) \cdot 2 \cdot (F(0) + F(1) + F(2) + \ldots + F(n-1)) + n$$

Folglich gilt

$$F(n-1) = (1/(n-1)) \cdot 2 \cdot (F(0) + F(1) + F(2) + \ldots + F(n-2)) + n-1$$
$$= (n/(n-1)) \cdot (1/n) \cdot 2 \cdot (F(0) + F(1) + F(2) + \ldots + F(n-2)) + n-1$$

Den Wert für

$$(1/n) \cdot 2 \cdot (F(0) + F(1) + F(2) + \ldots + F(n-2)) = (n-1)/n \cdot (F(n-1) - n+1)$$

setzen wir nun oben ein und erhalten die Rekursionsformel:

$$F(n) = (1/n) \cdot 2 \cdot F(n-1) + (n-1)/n \cdot (F(n-1) - n+1) + n$$
$$= (n+1)/n \cdot (F(n-1) - (n-1)^2/n + n = (n+1)/n \cdot F(n-1) + (2n-1)/n$$

Hiermit kann man die interne Pfadlänge, die man für zufällig ausgewählte Binärbäume erwarten muss, bereits berechnen:

$F(2) = 3/2 \cdot F(1) + 3/2 = 3$
$F(3) = 4/3 \cdot F(2) + 5/3 = 17/3 = 5{,}6666...$
$F(4) = 5/4 \cdot F(3) + 7/4 = 106/12 = 8{,}8333...$

Um eine geschlossene Formel für die exakte Lösung zu erhalten, probiert man einige Umformungen aus, zum Beispiel vereinfacht man die Formel durch Division durch (n+1); anschließend setzt man E(n-1) und danach E(n-2), E(n-3) usw. ein, bis man eine geschlossene Darstellung erhält.
$F(n) = (n+1)/n \cdot F(n-1) + (2n-1)/n$ wird also umgewandelt in

$$\frac{F(n)}{n+1} = \frac{2n-1}{n\cdot(n+1)} + \frac{F(n-1)}{n}$$

Wir ersetzen F(n-1)/n nun mit dieser Formel:

$$\frac{F(n)}{n+1} = \frac{2n-1}{n\cdot(n+1)} + \frac{F(n-1)}{n}$$

$$= \frac{2n-1}{n\cdot(n+1)} + \frac{2n-3}{(n-1)\cdot n} + \frac{F(n-2)}{n-1}$$

$$= \frac{2n-1}{n\cdot(n+1)} + \frac{2n-3}{(n-1)\cdot n} + \frac{2n-5}{(n-2)\cdot(n-1)} + \frac{F(n-3)}{n-2}$$

usw. So erhält man folgende Summenformel (F(0)=0):

$$\frac{F(n)}{n+1} = \sum_{i=0}^{n-1} \frac{2(n-i)-1}{(n-i)\cdot(n-i+1)} + \frac{F(0)}{1}$$

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

$$\frac{F(n)}{n+1} = \sum_{i=0}^{n-1} \frac{2}{(n-i+1)} - \sum_{i=0}^{n-1} \frac{1}{(n-i)\cdot(n-i+1)}$$

$$= 2\cdot\sum_{i=2}^{n+1} \frac{1}{i} - \sum_{i=1}^{n} \frac{1}{i\cdot(i+1)}$$

$$= 2\cdot\big(H(n+1) - 1\big) - \sum_{i=1}^{n} \Big(\frac{1}{i} - \frac{1}{i+1}\Big)$$

$$= 2\cdot H(n+1) - 2 - 1 + \frac{1}{n+1} \qquad \text{mit } H(n) = \sum_{i=1}^{n} \frac{1}{i}$$

$F(n) = 2\cdot(n+1)\cdot H(n+1) - 3\cdot(n+1) + 1 = 2\cdot(n+1)\cdot H(n) - 3\cdot n$

Definition 3.2.2.14: H(n) heißt "harmonische Funktion".
Hierbei wird $H(0) = 0$ gesetzt.

Lösung: $F(n) = 2\cdot(n+1)\cdot H(n) - 3\cdot n$

Aus der Analysis ist bekannt: $H(n) - \ln(n) \longrightarrow \gamma$ für $n \to \infty$.
ln ist der natürliche Logarithmus (zur Basis e = 2,7182818284...),
$\gamma = 0{,}5772156649...$ ist die Eulersche Konstante. Einsetzen ergibt:

$$F(n) \approx 2\cdot(n+1)\cdot(\ln(n) + \gamma) - 3\cdot n$$
$$\approx 2\cdot(n+1)\cdot\log(n)/\log(e) + 2\cdot(n+1)\cdot\gamma - 3\cdot n$$
$$\approx 1.3863\cdot n\cdot\log(n) - 1{,}8456\cdot n + O(\log(n))$$

Satz 3.2.2.15:

Die mittlere Suchzeit eines Binärbaums mit n Knoten beträgt im Mittel $1.3863\cdot\log(n) - 1{,}8456$. Sie ist somit nur um knapp 39% schlechter als die mittlere Suchzeit im günstigsten Fall.

(Letztere sollten Sie selbst berechnen; sie liegt bei log(n) - 1.)

Wir können nun die Zeitkomplexität eines Sortieralgorithmus angeben, der in 1.6.4.4 beschrieben wurde: Sortiere n Elemente, indem sie nacheionander in einen Suchbaum eingefügt und anschließend inorder ausgelesen werden.

Nach Satz 3.2.2.15 benötigt dieses Verfahren im Mittel $1.3863 \cdot \log(n) + O(n)$ Schritte. Im schlechtesten Fall kann allerdings ein zu einer Liste entarteter Suchbaum entstehen, so dass das Verfahren im worst case $O(n^2)$ Schritte braucht.

Kann man vielleicht den Suchbaum so einschränken, dass diese Entartung nicht eintreten kann und auch schlechtesten Fall $O(n \cdot \log(n))$ Schritte benötigt werden? Ja, siehe später AVL-Bäume.

Doch zunächst wollen wir uns jedoch kurz den "besten" Suchbäumen zuwenden.

3.2.2.16: Hinweis für analytisch Interessierte:

Wie kann man einer Rekursionsgleichung ansehen, welche Lösungen sie hat?

Das lässt sich nicht pauschal beantworten. Denn schließlich ist dieses Problem nicht entscheidbar. Manchmal reicht es aber bereits, wenn man die Gleichung umformt (erweitern, einsetzen, in irgendeinem Sinne vereinfachen usw.).

Oft ist es hilfreich, die "Ableitung" zu betrachten, also
$F(n) - F(n-1)$, oder die zweite Ableitung
$(F(n) - F(n-1)) - (F(n-1) - F(n-2)) = F(n) - 2\,F(n-1) + F(n-2)$.

Dies liefert einen Hinweis, wie die Lösung aussehen könnte, und man kann dann mit einem Lösungansatz, den man in die Gleichung einsetzt, versuchen, eine Lösung aufzuspüren.

Wir betrachten als Beispiel obige Rekursionsformel für F(n):

$F(n) = (n+1)/n \cdot F(n-1) + (2n-1)/n$. Einfaches Umformen liefert:
$F(n) = F(n-1) + 1/n \cdot F(n-1) + (2n-1)/n$, d.h.:
$F(n) - F(n-1) = 1/n \cdot F(n-1) + (2n-1)/n$.
Hier sieht man wenig, weil der Wert F(n-1) noch auf der rechten Seite stehen geblieben ist.
Jetzt kann man versuchen, die zweite Ableitung zu berechnen.
Aus obiger Formel für F(n)-F(n-1) folgt:
$F(n-1) - F(n-2) = 1/(n-1) \cdot F(n-2) + (2n-3)/(n-1)$.
Man subtrahiere die letzte Formel von der davor:
$(F(n) - F(n-1)) - (F(n-1) - F(n-2))$
$= 1/n \cdot F(n-1) + (2n-1)/n - 1/(n-1) \cdot F(n-2) - (2n-3)/(n-1)$
$= 1/n \cdot F(n-1) - 1/(n-1) \cdot F(n-2) + 1/(n \cdot (n-1))$.
$1/n \cdot F(n-1) - 1/(n-1) \cdot F(n-2)$ erinnert nun an die Rekursionsformel, denn dort steht (ersetze n durch n+1):
$1/(n+1) \cdot F(n) = 1/n \cdot F(n-1) + ...$

Also gilt:
$1/n \cdot F(n-1) = 1/(n-1) \cdot F(n-2) + (2n-3)/(n \cdot (n-1))$ bzw.
$1/n \cdot F(n-1) - 1/(n-1) \cdot F(n-2) = (2n-3)/(n \cdot (n-1))$.
Dies setzt man oben ein:
$(F(n) - F(n-1)) - (F(n-1) - F(n-2))$
$= 1/n \cdot F(n-1) - 1/(n-1) \cdot F(n-2) + 1/(n \cdot (n-1))$
$= (2n-3)/(n \cdot (n-1)) + 1/(n \cdot (n-1))$
$= (2n-2)/(n \cdot (n-1)) = 2/n$

Es entsteht ein überraschend einfacher Ausdruck. Nun erinnern wir uns an die Stammfunktionen aus der Analysis: 1/n ist die Ableitung des Logarithmus ln(n). Also müsste die "Ableitung" ln(n) und damit die Funktion F von der Form $n \cdot ln(n)$ sein. Da keine kontinuierliche Funktion herauskommen wird, sollte man H(n) anstelle von ln(n) nehmen, d.h., das Ergebnis könnte die Form $F(n) = a \cdot n \cdot H(n) + b \cdot n + c$ (mit Konstanten a, b, c) haben.

Dies setzt man nun in die ursprüngliche Gleichung
$F(n) = (n+1)/n \cdot F(n-1) + (2n-1)/n$ ein:

$a \cdot n \cdot H(n) + b \cdot n + c$
$= (n+1)/n \cdot (a \cdot (n-1) \cdot H(n-1) + b \cdot (n-1) + c) + (2n-1)/n$

Multiplikation mit n liefert:

$a \cdot n^2 \cdot H(n) + b \cdot n^2 + c \cdot n$
$= (n+1) \cdot a \cdot (n-1) \cdot H(n-1) + b \cdot (n^2-1) + c \cdot (n+1) + (2n-1)$, also

$a \cdot n^2 \cdot H(n-1) + a \cdot n = a \cdot (n^2-1) \cdot H(n-1) - b + c + (2n-1)$
$a \cdot H(n-1) + a \cdot n = c - b + (2n-1)$.

Man sieht, dass diese Gleichung für Konstanten a, b und c nicht zu erfüllen ist. Man braucht offenbar im Ansatz ein weiteres Glied $d \cdot H(n)$. Neuer Ansatz:
$F(n) = a \cdot n \cdot H(n) + b \cdot n + c + d \cdot H(n)$.
Wegen der Nebenbedingung F(0)=0 muss c=0 sein. Erneut einsetzen:

$a \cdot n \cdot H(n) + b \cdot n + d \cdot H(n)$
$= (n+1)/n \cdot (a \cdot (n-1) \cdot H(n-1) + b \cdot (n-1) + d \cdot H(n-1)) + (2n-1)/n$

Multiplikation mit n liefert:

$a \cdot n^2 \cdot H(n) + b \cdot n^2 + d \cdot n \cdot H(n)$
$= a \cdot n^2 \cdot H(n-1) + a \cdot n + b \cdot n^2 + d \cdot n \cdot H(n)$
$= (n+1) \cdot a \cdot (n-1) \cdot H(n-1) + b \cdot (n^2-1) + d \cdot (n+1) \cdot H(n-1) + (2n-1)$
$= a \cdot (n^2-1) \cdot H(n-1) + b \cdot (n^2-1) + d \cdot n \cdot H(n-1) + d \cdot H(n-1) + (2n-1)$.

Umformen:

$a \cdot H(n-1) + a \cdot n + b \cdot n^2 + d \cdot n \cdot H(n)$
$= b \cdot (n^2-1) + d \cdot n \cdot H(n-1)) + d \cdot H(n-1)) + (2n-1)$ wird zu

$a \cdot H(n-1) + a \cdot n + d = -b + d \cdot H(n-1) + (2n-1)$.

Wenn dies lösbar ist, so muss a=d und a=2 sein; es verbleibt:
$2 = -b - 1$, d.h., b=-3. Somit haben wir eine Lösung gefunden:
$F(n) = 2 \cdot n \cdot H(n) - 3 \cdot n + 2 \cdot H(n)$, also die gleiche Lösung wie oben.