Was: Sortierung

Wie: Mischsortierung
rekursives **sort**
merge

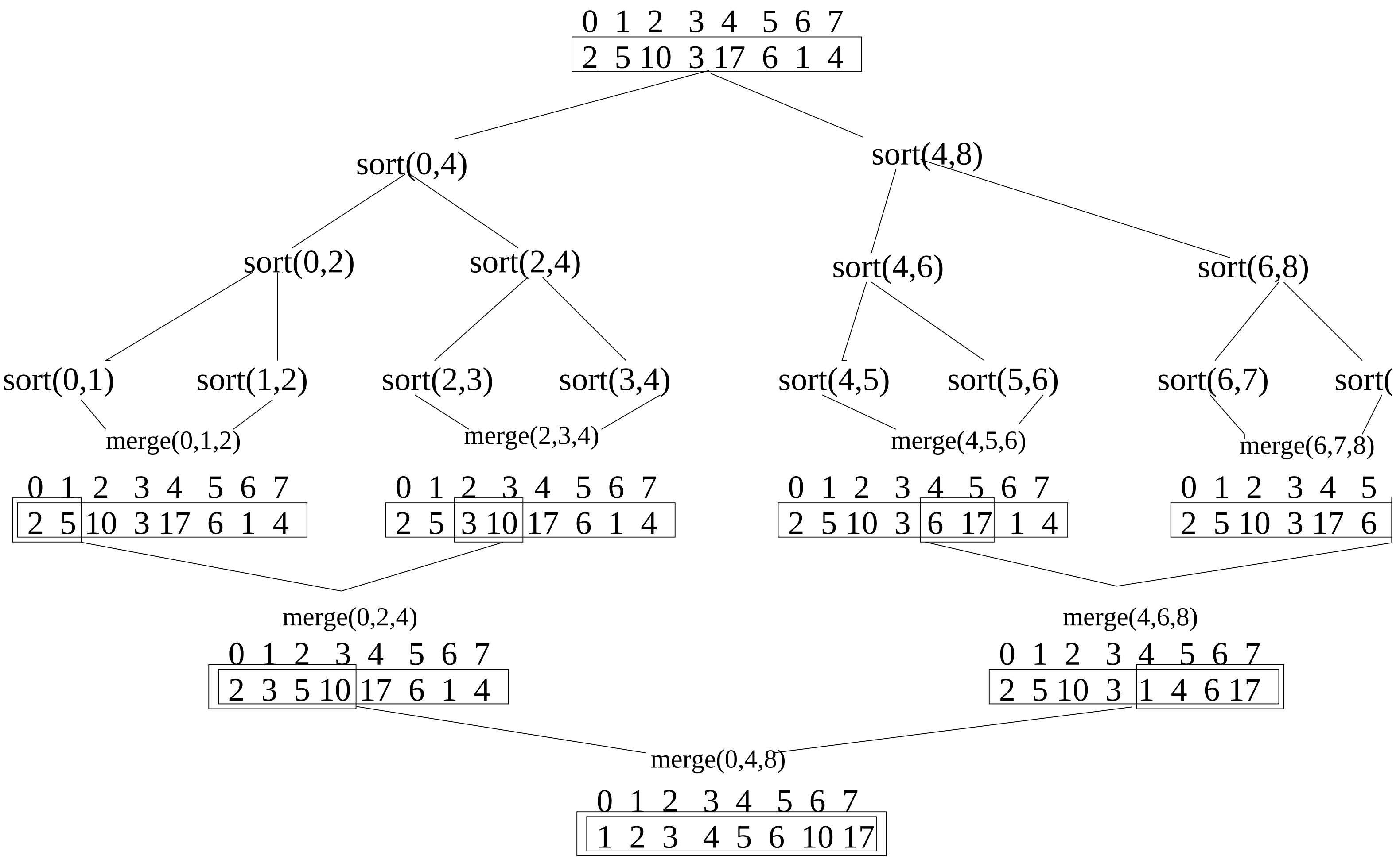
Warum: Aufwandsabschätzung
O-Notation

Problem Sortierung

Sortieren ist der Prozeß, der eine ungeordnete Menge in eine geordnete Menge überführt. Was wir dazu brauchen ist eine Ordnungsrelation, die uns für zwei Elemente der Menge entscheidet, ob sie den gleichen Rang haben oder das eine Element einen höheren Rang hat als das andere.

Mischsortieren

- Wir möchten gern eine gemäß eines Ordnungskriteriums geordnete Folge von Objekten haben. Wir nehmen Zahlen und ihre $>$-Ordnungsrelation.
- Wir teilen die ungeordnete Menge in zwei Teile und rufen für jeden Teil unsere Sortiermethode auf.
- Wir mischen die beiden jede für sich geordneten Folgen, indem wir sie elementweise von links nach rechts vergleichen: bei jedem Schritt wird das kleinere Element der beiden in die Ergebnisfolge eingetragen.


0 1 2 3 4 5 6 7
2 5 10 3 17 6 1 4
sort(0,4)
sort(4,8)
sort(0,2)
sort(2,4)
sort(4,6)
sort(6,8)
sort(0,1)
sort(1,2)
sort(2,3)
sort(3,4)
sort(4,5)
sort(5,6)
sort(6,7)
sort(
merge(0,1,2)
merge(2,3,4)
merge(4,5,6)
merge(6,7,8)
0 1 2 3 4 5 6 7
2 5 10 3 17 6 1 4
0 1 2 3 4 5 6 7
2 5 3 10 17 6 1 4
0 1 2 3 4 5 6 7
2 5 10 3 6 17 1 4
0 1 2 3 4 5
2 5 10 3 17 6
merge(0,2,4)
merge(4,6,8)
0 1 2 3 4 5 6 7
2 3 5 10 17 6 1 4
0 1 2 3 4 5 6 7
2 5 10 3 1 4 6 17
merge(0,4,8)
0 1 2 3 4 5 6 7
1 2 3 4 5 6 10 17

... in JAVA

```
public class AnimatedMergeSort {
    public static void sort (int[] feld) {
                                  // wrapper auf parameterisierten
        sort (feld, 0, feld.length);                      // aufruf
    }
    public static void sort (int[] feld, int unten, int oben) {
        if (unten < oben - 1) {             // noch was zu tun?
            int mitte = (unten + oben) / 2;         // ja, split

            sort (feld, unten, mitte);          // beide teilfolgen
            sort (feld, mitte, oben);          // rekursiv sortieren

            merge (feld, unten, mitte, oben);         // mischen
        }
    }
    public static void merge (int[] feld, int unten, int mitte,
int oben) {

        int i = unten, j = mitte, k=0;              // Laufindizes
        int[] ergebnis = new int[oben - unten];
                              // Platz fuer Ergebnisfolge besorgen

        while ((i < mitte) && (j < oben)) {
                                      // mischen, bis ein Feld leer
            if (feld[i] < feld[j])                   // das kleinere
            ergebnis[k++] = feld[i++];
```

```
                                    // in ergebnis uebernehmen
            else
            ergebnis[k++] = feld[j++];
        }
        if (i == mitte)                         // falls i am Ende:
        while (j<oben) ergebnis[k++] = feld[j++];
                                    // Rest von oben uebernehmen
        else                                    // falls j am Ende:
        while (i<mitte) ergebnis[k++] = feld[i++];
                                    // Rest von unten uebernehmen

        for (i = 0; i < ergebnis.length; i++)
                            // Ergebnis in Feld zurueckkopieren
         {
            feld [i + unten] = ergebnis [i];
            ShowArray.show (feld, unten, oben-1, unten+i);
        }
    }
}
```

Ergebnis

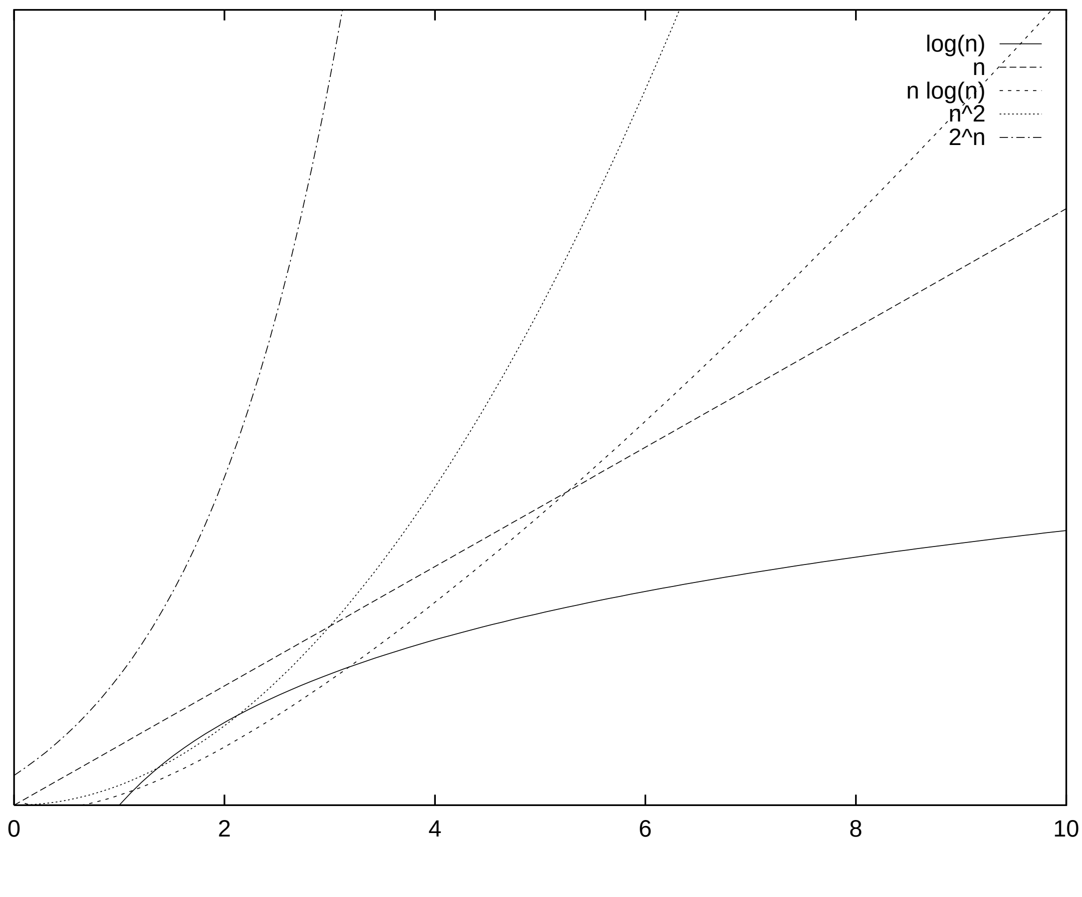
```
    sort([I@3a806a3b,0,1)
sort([I@3a806a3b,1,2)
feld[0]:2 feld[1]:4 feld[2]:0 feld[3]:0 feld[4]:7 feld[5]:2
feld[6]:6
merge([I@3a806a3b,0,1,2)
sort([I@3a806a3b,0,2)
sort([I@3a806a3b,2,3)
sort([I@3a806a3b,3,4)
feld[0]:2 feld[1]:4 feld[2]:0 feld[3]:0 feld[4]:7 feld[5]:2
feld[6]:6
merge([I@3a806a3b,2,3,4)
sort([I@3a806a3b,2,4)
feld[0]:0 feld[1]:0 feld[2]:2 feld[3]:4 feld[4]:7 feld[5]:2
feld[6]:6
merge([I@3a806a3b,0,2,4)
sort([I@3a806a3b,0,4)
sort([I@3a806a3b,4,5)
sort([I@3a806a3b,5,6)
feld[0]:0 feld[1]:0 feld[2]:2 feld[3]:4 feld[4]:2 feld[5]:7
feld[6]:6
merge([I@3a806a3b,4,5,6)
sort([I@3a806a3b,4,6)
sort([I@3a806a3b,6,7)
```

sort([I@3a806a3b,7,8)
feld[0]:0 feld[1]:0 feld[2]:2 feld[3]:4 feld[4]:2 feld[5]:7
feld[6]:2
merge([I@3a806a3b,6,7,8)
sort([I@3a806a3b,6,8)
feld[0]:0 feld[1]:0 feld[2]:2 feld[3]:4 feld[4]:2 feld[5]:2
feld[6]:6
merge([I@3a806a3b,4,6,8)
sort([I@3a806a3b,4,8)
feld[0]:0 feld[1]:0 feld[2]:2 feld[3]:2 feld[4]:2 feld[5]:4
feld[6]:6
merge([I@3a806a3b,0,4,8)

Effizienz: Aufwandsabschätzung

Reale Laufzeit vs. abstrakte Laufzeit

Wie wächst die Laufzeit (der Speicherbedarf) im Verhältnis mit der Größe der Eingabe?



Abstrakte Laufzeit

Laufzeit $T(n)$ zur Größe n der Eingabe, wobei c irgendeine Konstante ist:

konstant: $T(n) = c$

logarithmisch: $T(n) = c \cdot \log n$

linear: $T(n) = c \cdot n$

n log n: $T(n) = c \cdot n \log n$

quadratisch: $T(n) = c \cdot n^2$

polynomiell: $T(n) = c \cdot n^k, \quad k \geq 2$

exponentiell: $T(n) = c \cdot 2^n$

Nehmen wir an, Programm A hätte die Laufzeit $100 \cdot n$ und Programm B hätte die Laufzeit $2 \cdot n^2$. Dann ist B schneller als A, solange $n < 50$. Für $n = 100$ ist A schon doppelt so schnell wie B. Für $n = 1000$ ist A 20mal so schnell.

Aufwand O

Die Laufzeit eines Programms wird durch eine Funktion $T(n)$ über der Größe n (der Eingabe) angegeben.

n und $T(n)$ sind nichtnegative (meist ganze) Zahlen.

$f(n)$ sei eine Funktion, die über nichtnegativen (ganzen) Zahlen definiert ist.

" $T(n)$ ist $O(f(n))$", wenn es

- eine Zahl n_0 gibt und
- eine Konstante $c > 0$,
- so daß für alle Zahlen $n \geq n_0$ gilt $T(n) \leq c \cdot f(n)$.

abstraktes Beispiel

Nehmen wir an, $T(n) = (n+1)^2$,

dann ist $T(n)$ quadratisch, d.h. $O(n^2)$,

denn wir können $c = 2$ und $n_0 = 3$ auswählen oder $c = 4$ und $n_0 = 1$.

Es gilt sowohl

$T(n) = (n+1)^2 \leq 2 \cdot n^2 = 2 \cdot f(n)$ als auch

$T(n) = (n+1)^2 \leq 4 \cdot n^2 = 4 \cdot f(n)$.

Man sieht dies besonders leicht, wenn man die Formel anwendet:

$(n+1)^2 = n^2 + 2 \cdot n + 1$

Dann ist nämlich $T(n)$ sichtbar kleiner als unser $c \cdot f(n)$:

$T(n) = n^2 + 2 \cdot n + 1 \leq n^2 + 2 \cdot n^2 + n^2 = 4 \cdot n^2$.

Natürlich darf n_0 nie 0 sein, weil jede Multiplikation mit n_0 dann 0 ergibt.

Induktion bei Rekursion

Wie drücken wir die Rekursion aus?

- die Methode wird mit einer um 1 verkürzten Datenstruktur aufgerufen:

  $T(n) = O(1) + T(n-1)$

  Induktionsbeweis über n: 1,2,3,4,5, ...

- die Methode wird mit einer halbierten Datenstruktur aufgerufen:

  $T(n) = O(1) + 2^i \cdot T(n/2^i)$

  $i_1 = 1, i_2 = 2, i_3 = 4, i_4 = 8, ...$

  $i_j = 2^{j-1}$

  Induktionsbeweis über j: 1,2,3,4,5, ...

  $T(n/2^i) = O(1) + 2 \cdot T(n/2^{i+1})$

Nicht vergessen: $log_2 n = i$, $2^i = n$

$log_2 8 = 3$, $2^3 = 8$

Aufwandsabschätzung der Mischsortierung: merge

- Wie beim Induktionsbeweis betrachten wir zuerst den Basisfall: wie ist der Aufwand bei $n = 1$?
  - Als erstes sehen wir in der $while$-Schleife die Abbruchsbedingung. Wenn sie falsch ist, also ein Feld leer ist, dann sind wir mit dieser Schleife fertig. Bei nur einem Element ist sicherlich eines der Felder leer. Der Aufwand für die Schleife war nur die eine Handlung, die Bedingung zu testen, also $O(1)$.
  - Es ist dann auch in der *bedingten Schleife* nur eines der Felder mit seinem Rest in das Ergebnis zu übernehmen. Wieviele Schritte dies sind, hängt von der Länge des Feldes ab. Im Basisfall also 1. Wir haben wieder $O(1)$.
  - Die for- Schleife hängt ebenfalls direkt von der Feldlänge ab, ist also ebenfalls nur einmal durchzuführen.

Wir haben drei Mal den Aufwand $O(1)$ im Basisfall. Das macht insgesamt $O(1)$, denn es geht ja um die Funktionsklasse und nicht um das Zählen der tatsächlichen Schritte. $T(1)$ ist $O(1)$.

- Nehmen wir nun ein größeres n.
  - *while*-Schleife:
    Es kann auch jetzt eines der Felder leer sein. Wenn das eine Feld immer die kleineren Elemente hatte als das andere, so ist es leer und das andere noch nicht. Wir haben dann – unabhängig von der Feldlänge – den Aufwand $O(1)$ für den (erfolgreichen) Test.
    Solange kein Feld leer ist, müssen wir das kleinste Element auswählen. In der Schleife ist dies ein konstanter Aufwand, der nicht von n abhängt. Wie oft wir die Schleife durchlaufen, hängt von n ab. Wir zählen den Feldindex immer um 1 hoch, so daß wir jedes Mal mit einem um 1 verkürzten Feld erneut in die Schleife eintreten bis eines von beiden leer ist. Dies ergibt also im schlimmsten Falle $n-1$ Schritte ($n = 1$ hatten wir schon). Die erste Schleife ist also $O(1) + T(n-1)$.
  - Die *bedingte Schleife* muß nun den Rest des Feldes in das Ergebnis übernehmen. Im schlimmsten Fall muß das eine Feld ganz durchgegangen werden. Damit haben wir nochmals $n-1$ Schritte.
  - Die *for*-Schleife geht auf jeden Fall das gesam-

te Feld, mit dem wir **merge** aufgerufen haben, durch. Auch hier haben wir wieder $n-1$ Schritte.

Wir setzen die Abschätzungen der Schleifen für $n \geq 1$ zusammen und erhalten: $T(n) = O(1) + T(n-1)$. Also ist der Aufwand von **merge** $O(n)$.

Aufwandsabschätzung der Mischsortierung: sort

- Als Basisfall nehmen wir wieder $m = 1$. Dieses winzige Feld wird nur in **sort** mit dem ersten Test bearbeitet, der negativ beantwortet wird. **merge** wird nicht aufgerufen. Wir haben $T(1)$ ist $O(1)$.

- Bei $m > 1$ müssen wir nun das Aufteilen betrachten. Dies ist rekursiv. Solange nicht ein einelementiges Feld durch das Aufteilen entsteht, wird die Feldlänge immer durch 2 geteilt. Für jede der beiden Felder wird **sort** wieder aufgerufen und dann **merge**.

Der Aufwand von **merge** war für die geteilten Felder $O(n)$. Nun ist $n = \frac{m}{2}$. Wir haben folglich den Aufwand $T(\frac{m}{2})$ zweimal. Weil hier die Anzahl der Schritte von der Feldlänge abhängt, müssen wir auch $2 \cdot T(\frac{m}{2})$ angeben. Das Ergebnisfeld ist wieder so lang wie das Ausgangsfeld, also $O(m)$. Wir erhalten für $m > 1$:

$$T(m) = 2 \cdot T(\tfrac{m}{2}) + m, \qquad \text{(Rekursion)}$$

wobei m eine Potenz von 2 ist (wir teilen ja immer durch 2). Wir merken uns dies als Beschreibung für den Rekursionsschritt.

Was für ein Verhältnis ist das? Wir sehen sofort, daß es nicht linear und nicht exponentiell ist. Quadratisch sieht es auch nicht aus. Versuchen wir einfach, zu beweisen, daß der Aufwand n log n ist!

Beispiel

Als Beispiel, das die Vorstellung unterstützt, nehmen Sie:

Feldlänge $= 8$

Wir teilen das 1. Mal auf und erhalten

$2\cdot$ Feldlänge $= 2 \cdot 4$

Wir teilen das 2. Mal auf und erhalten

$4\cdot$ Feldlänge $= 4 \cdot 2$

Wir teilen das $\log_2 8 = 3$. Mal auf und erhalten

$8\cdot$ Feldlänge $= 8 \cdot 1$.

Mergesort ist n log n in der Feldlänge

Basis $T(1) = a$

Induktion $T(n) = 2 \cdot T(\frac{n}{2}) + bn$, wobei n eine Potenz von 2 ist.

Jetzt raten wir, daß $f(n) = c \cdot n \cdot log_2 n + d$, wobei c und d Konstanten sind. Und dann beweisen wir, daß $T(n) \leq f(n)$ mit vollständiger Induktion über n.

Aussage $S(n)$: Wenn n eine Potenz von 2 ist und $n > 1$ gilt, dann ist $T(n) \leq f(n)$ mit $f(n) = c \cdot n \cdot log_2 n + d$.

Induktionsanfang Wenn $n = 1$ gilt $T(1) \leq f(1)$, falls $a \leq d$ ist. $f(1) = d$ weil $c \cdot 1 \cdot log_2 1$ gleich 0 ist.

Induktionsschritt Nehmen wir an, wir hätten bis $n-1$ die Aussage schon bewiesen. Dann müssen wir jetzt den nächsten Schritt beweisen, $S(n)$.

Wenn n keine Potenz von 2 ist, gilt der "Wenn"-Teil der Aussage nicht und damit gilt $S(n)$ ohnehin.

Wenn n eine Potenz von 2 ist, nutzen wir unsere Annahme aus, daß alle früheren Schritte bereits bewiesen sind, also auch $S(n/2)$, d.h. es gilt

$T(n/2) \leq (c \cdot n/2) \cdot log_2(n/2) + d$ (Induktionsannahme)

Nun müssen wir zeigen, daß

$T(n) \leq c \cdot nlog_2 n + d$ gilt.

Die induktive Definition von $T(n)$ sagt, daß gilt

$$T(n) = 2T(n/2) + bn.$$

In dieser Gleichung setzen wir die Induktionsannahme für $T(n/2)$ ein.

Das ergibt

$$T(n) \leq 2((c \cdot n/2) \cdot log_2(n/2) + d) + bn.$$

Mit Hilfe von

$log_2(n/2) = (log_2 n) - 1$,

Ausmultiplizieren,

$2 \cdot (n/2) = n$,

$-c \cdot n + b \cdot n = (b - c) \cdot n$ vereinfachen wir die Ungleichung so:

$$T(n) \leq 2((c \cdot n/2) \cdot log_2(n/2) + d) + bn$$

$$= c \cdot n \cdot (log_2 n - 1) + b \cdot n + 2d$$

$$= c \cdot n \cdot log_2 n - c \cdot n \cdot 1 + b \cdot n + 2d$$

$$= c \cdot n \cdot log_2 n + b \cdot n - c \cdot n + 2d$$

daß wir die folgende Ungleichung erreichen:

$$T(n) \leq c \cdot n \cdot log_2 n + (b - c) \cdot n + 2d.$$

Damit unsere Aussage gilt, muß $(b - c)n + d \leq 0$ sein. $n > 1$, also muß $b - c \leq -d$ sein. Anders ausgedrückt muß gelten $c \geq b + d$.

Dies gilt bei $d = a$ und $c = a + b$, was auch zu der Beschränkung von d im Basisfall paßt, nämlich $a \leq d$.

Wir haben also durch Induktion über n gezeigt, daß für alle $n \geq 1$, die Zweierpotenzen sind, gilt:

$$T(n) \leq (a+b) \cdot n \cdot log_2 n + a.$$

Damit gehört der Aufwand für die Mischsortierung in die Klasse $O(n \log n)$.