2 Methoden zum Entwurf von Algorithmen

Divide-and-Conquer

Gegeben sei **Instanz $\mathcal{I}$ eines Problems**.

Löse $\mathcal{I}$ wie folgt:

Divide-and-Conquer

Gegeben sei **Instanz $\mathcal{I}$ eines Problems**.

Löse $\mathcal{I}$ wie folgt:

1. **Zerlege $\mathcal{I}$ in Teilinstanzen** $\mathcal{I}_1, \ldots, \mathcal{I}_k$ für ein $k \geq 1$.

Divide-and-Conquer

Gegeben sei **Instanz $\mathcal{I}$ eines Problems**.

Löse $\mathcal{I}$ wie folgt:

1. **Zerlege $\mathcal{I}$ in Teilinstanzen** $\mathcal{I}_1, \ldots, \mathcal{I}_k$ für ein $k \geq 1$.
2. **Löse die Teilinstanzen** $\mathcal{I}_1, \ldots, \mathcal{I}_k$ (rekursiv).

Divide-and-Conquer

Gegeben sei **Instanz $\mathcal{I}$ eines Problems**.

Löse $\mathcal{I}$ wie folgt:

1. **Zerlege $\mathcal{I}$ in Teilinstanzen** $\mathcal{I}_1, \ldots, \mathcal{I}_k$ für ein $k \geq 1$.
2. **Löse die Teilinstanzen** $\mathcal{I}_1, \ldots, \mathcal{I}_k$ (rekursiv).
3. **Kombiniere die Lösungen** von $\mathcal{I}_1, \ldots, \mathcal{I}_k$ **zu Gesamtlösung** von $\mathcal{I}$.

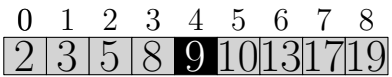
Erstes Beispiel: **Binäre Suche**

// Suche in $a[\ell \ldots r]$ nach x

```
BinarySearch(int[] a, int x, int ℓ, int r)
1    if (ℓ > r) return false;
2    int m = ℓ + (r − ℓ)/2;
3    if (a[m] < x)
4        return BinarySearch(a,x,m + 1,r);
5    else if (a[m] > x)
6        return BinarySearch(a,x,ℓ,m − 1);
7    return true;
```

initialer Aufruf: $\text{BinarySearch}(a, x, 0, n-1)$

Suche nach $x = 8$



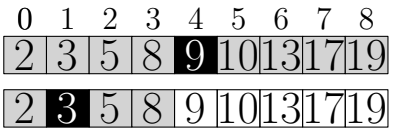
Erstes Beispiel: **Binäre Suche**

// Suche in $a[\ell \ldots r]$ nach x

```
BinarySearch(int[] a, int x, int ℓ, int r)
1    if (ℓ > r) return false;
2    int m = ℓ + (r − ℓ)/2;
3    if (a[m] < x)
4        return BinarySearch(a,x,m + 1,r);
5    else if (a[m] > x)
6        return BinarySearch(a,x,ℓ,m − 1);
7    return true;
```

initialer Aufruf: $\text{BinarySearch}(a, x, 0, n-1)$

Suche nach $x = 8$



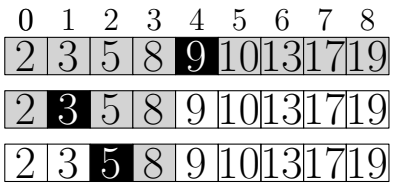
Erstes Beispiel: **Binäre Suche**

// Suche in $a[\ell \ldots r]$ nach x

```
BinarySearch(int[] a, int x, int ℓ, int r)
1   if (ℓ > r) return false;
2   int m = ℓ + (r − ℓ)/2;
3   if (a[m] < x)
4       return BinarySearch(a,x,m + 1,r);
5   else if (a[m] > x)
6       return BinarySearch(a,x,ℓ,m − 1);
7   return true;
```

initialer Aufruf: $\textsc{BinarySearch}(a, x, 0, n-1)$

Suche nach $x = 8$



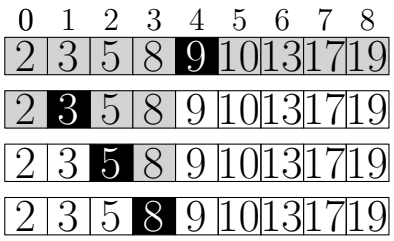
Erstes Beispiel: **Binäre Suche**

// Suche in $a[\ell \ldots r]$ nach x

```
BinarySearch(int[] a, int x, int ℓ, int r)
1    if (ℓ > r) return false;
2    int m = ℓ + (r − ℓ)/2;
3    if (a[m] < x)
4        return BinarySearch(a,x,m + 1,r);
5    else if (a[m] > x)
6        return BinarySearch(a,x,ℓ,m − 1);
7    return true;
```

initialer Aufruf: $\textsc{BinarySearch}(a, x, 0, n-1)$

Suche nach $x = 8$



Theorem 2.1

Die Laufzeit von binärer Suche beträgt $O(\log n)$.

Theorem 2.1

Die Laufzeit von binärer Suche beträgt $O(\log n)$.

Beweis:

- Betrachte Aufruf von BINARYSEARCH mit **Intervalllänge** $z = r - \ell + 1 > 0$.
  - z ungerade: rekursiver Aufruf mit Intervalllänge $(z-1)/2$
  - z gerade: rekursiver Aufruf mit Intervalllänge $z/2 - 1$ oder $z/2$

  $\Rightarrow$ **rekursiver Aufruf mit Intervalllänge** $\leq z/2$

Theorem 2.1

Die Laufzeit von binärer Suche beträgt $O(\log n)$.

Beweis:

- Betrachte Aufruf von BINARYSEARCH mit **Intervalllänge** $z = r - \ell + 1 > 0$.
  - z ungerade: rekursiver Aufruf mit Intervalllänge $(z-1)/2$
  - z gerade: rekursiver Aufruf mit Intervalllänge $z/2 - 1$ oder $z/2$

  $\Rightarrow$ **rekursiver Aufruf mit Intervalllänge** $\leq z/2$
- nach k rekursiven Aufrufen **Intervalllänge** $\leq z/2^k$

Theorem 2.1

Die Laufzeit von binärer Suche beträgt $O(\log n)$.

Beweis:

- Betrachte Aufruf von BINARYSEARCH mit **Intervalllänge** $z = r - \ell + 1 > 0$.
  - z ungerade: rekursiver Aufruf mit Intervalllänge $(z-1)/2$
  - z gerade: rekursiver Aufruf mit Intervalllänge $z/2 - 1$ oder $z/2$

  $\Rightarrow$ **rekursiver Aufruf mit Intervalllänge** $\leq z/2$
- nach k rekursiven Aufrufen **Intervalllänge** $\leq z/2^k$
- Für $K = \lfloor \log_2(n) \rfloor + 1$ gilt $n/2^K < 1$.

  $\Rightarrow$ **Höchstens** K **rekursive Aufrufe bis Intervalllänge** 0 **erreicht ist.**

Theorem 2.1

Die Laufzeit von binärer Suche beträgt $O(\log n)$.

Beweis:

- Betrachte Aufruf von BINARYSEARCH mit **Intervalllänge** $z = r - \ell + 1 > 0$.
  - z ungerade: rekursiver Aufruf mit Intervalllänge $(z-1)/2$
  - z gerade: rekursiver Aufruf mit Intervalllänge $z/2 - 1$ oder $z/2$

  $\Rightarrow$ **rekursiver Aufruf mit Intervalllänge** $\leq z/2$
- nach k rekursiven Aufrufen **Intervalllänge** $\leq z/2^k$
- Für $K = \lfloor \log_2(n) \rfloor + 1$ gilt $n/2^K < 1$.

  $\Rightarrow$ **Höchstens** K **rekursive Aufrufe bis Intervalllänge** 0 **erreicht ist.**
- Abgesehen vom rekursiven Aufruf benötigt BINARYSEARCH konstante Laufzeit.

Theorem 2.1

Die Laufzeit von binärer Suche beträgt $O(\log n)$.

Beweis:

- Betrachte Aufruf von BINARYSEARCH mit **Intervalllänge** $z = r - \ell + 1 > 0$.
  - z ungerade: rekursiver Aufruf mit Intervalllänge $(z-1)/2$
  - z gerade: rekursiver Aufruf mit Intervalllänge $z/2 - 1$ oder $z/2$

  $\Rightarrow$ **rekursiver Aufruf mit Intervalllänge** $\leq z/2$
- nach k rekursiven Aufrufen **Intervalllänge** $\leq z/2^k$
- Für $K = \lfloor \log_2(n) \rfloor + 1$ gilt $n/2^K < 1$.

  $\Rightarrow$ **Höchstens** K **rekursive Aufrufe bis Intervalllänge** 0 **erreicht ist.**
- Abgesehen vom rekursiven Aufruf benötigt BINARYSEARCH konstante Laufzeit.

$\Rightarrow$ insgesamt Laufzeit $O(\log n)$ $\square$

2.1.1 Mergesort

MERGESORT:

Teile *a* in zwei Hälften, **sortiere die Hälften** unabhängig, **vereinige** die sortierten Hälften

2.1.1 Mergesort

MERGESORT:

Teile a in zwei Hälften, **sortiere die Hälften** unabhängig, **vereinige** die sortierten Hälften

// Sortiere $a[\ell \ldots r]$

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```

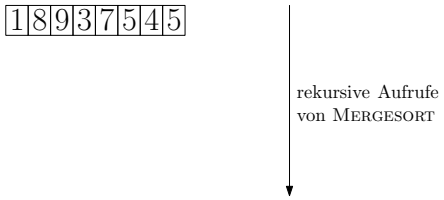
initialer Aufruf:

MERGESORT$(a, 0, n-1)$

2.1.1 Mergesort

MERGESORT:

Teile a in zwei Hälften, **sortiere die Hälften** unabhängig, **vereinige** die sortierten Hälften



// Sortiere $a[\ell \ldots r]$

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```

initialer Aufruf:

MERGESORT$(a, 0, n-1)$

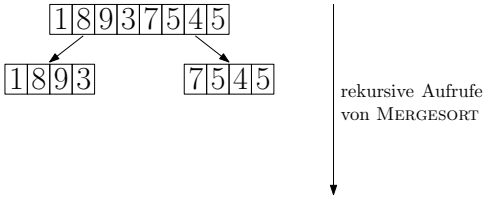
2.1.1 Mergesort

MERGESORT:

Teile a in zwei Hälften, **sortiere die Hälften** unabhängig, **vereinige** die sortierten Hälften

// Sortiere $a[\ell \ldots r]$

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```



initialer Aufruf:

MERGESORT$(a, 0, n-1)$

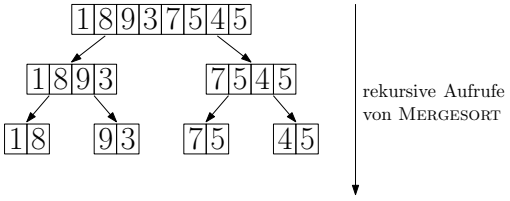
2.1.1 Mergesort

MERGESORT:

Teile a in zwei Hälften, **sortiere die Hälften** unabhängig, **vereinige** die sortierten Hälften

// Sortiere $a[\ell \ldots r]$

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```



initialer Aufruf:

MERGESORT$(a, 0, n-1)$

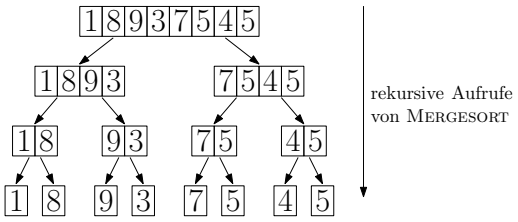
2.1.1 Mergesort

MERGESORT:

Teile a in zwei Hälften, **sortiere die Hälften** unabhängig, **vereinige** die sortierten Hälften

// Sortiere $a[\ell \ldots r]$

```
MERGESORT(int[] a, int ℓ, int r)
1   if (ℓ < r) {
2       int m = ℓ + (r − ℓ)/2;
3       MERGESORT(a,ℓ,m);
4       MERGESORT(a,m + 1,r);
5       MERGE(a,ℓ,m,r);
6   }
```



initialer Aufruf:

MERGESORT$(a, 0, n-1)$

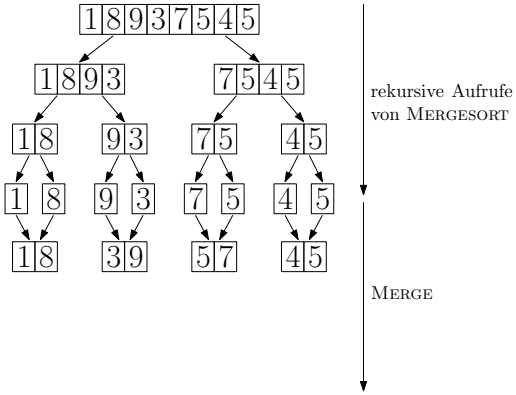
2.1.1 Mergesort

MERGESORT:

Teile a in zwei Hälften, **sortiere die Hälften** unabhängig, **vereinige** die sortierten Hälften

// Sortiere $a[\ell \ldots r]$

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```



initialer Aufruf:

MERGESORT$(a, 0, n-1)$

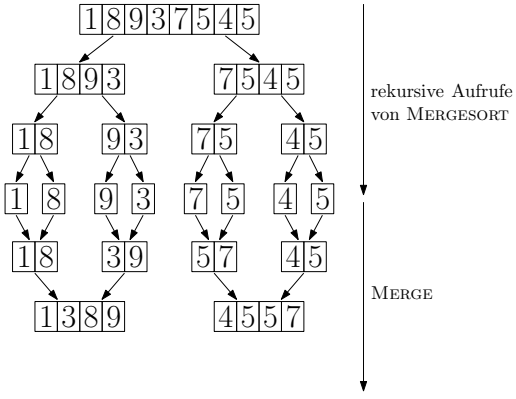
2.1.1 Mergesort

MERGESORT:

Teile a in zwei Hälften, **sortiere die Hälften** unabhängig, **vereinige** die sortierten Hälften

// Sortiere $a[\ell \ldots r]$

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```



initialer Aufruf:

MERGESORT$(a, 0, n-1)$

2.1.1 Mergesort

MERGESORT:

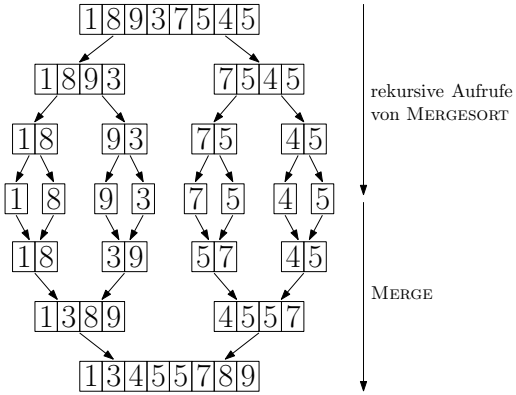
Teile a in zwei Hälften, **sortiere die Hälften** unabhängig, **vereinige** die sortierten Hälften

// Sortiere $a[\ell \ldots r]$

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```

initialer Aufruf:

MERGESORT$(a, 0, n-1)$



2.1.1 Mergesort

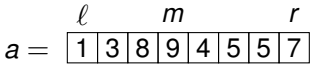
```
MERGE(int[ ] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2   int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3   for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4   for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6   int iL = 0; int iR = 0; int iA = ℓ;
7   while ((iL < m − ℓ + 1) && (iR < r − m)) {
8       if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9       else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12  while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13  while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```

$a =$

	ℓ			m				r
	1	3	8	9	4	5	5	7

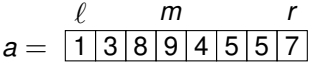
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



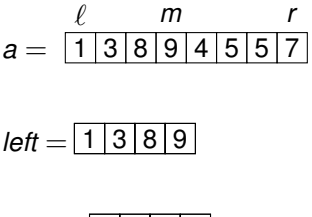
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



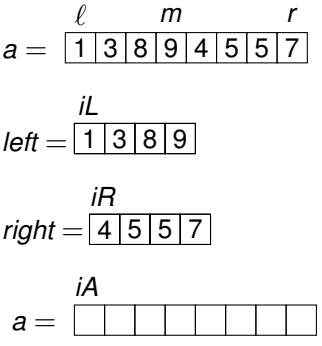
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



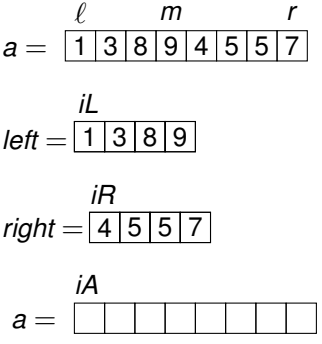
2.1.1 Mergesort

```
1  MERGE(int[] a, int ℓ, int m, int r)
2    // Kopiere die beiden sortierten Teilfelder in left und right.
3    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
4    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
5    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
6    // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
7    int iL = 0; int iR = 0; int iA = ℓ;
8    while ((iL < m − ℓ + 1) && (iR < r − m)) {
9      if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
10     else {a[iA] = right[iR]; iA++; iR++; }
11   }
12   // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
13   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
14   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



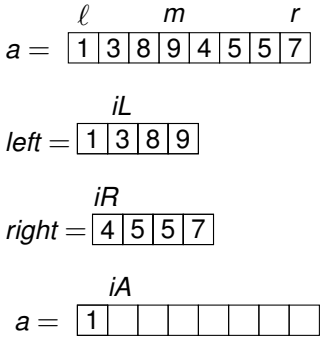
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



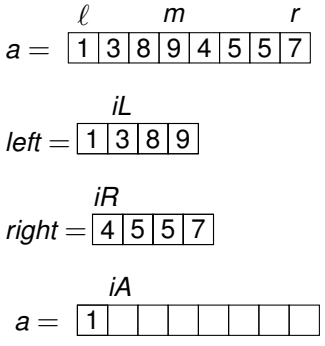
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



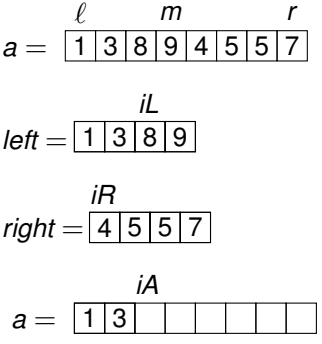
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



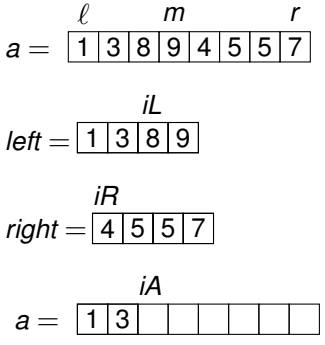
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



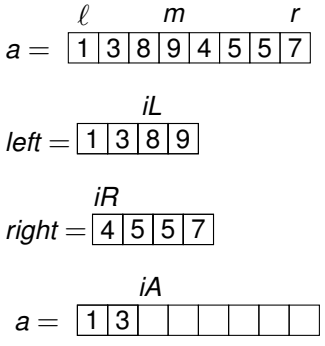
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



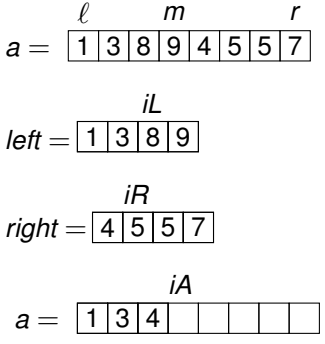
2.1.1 Mergesort

```
MERGE(int[ ] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



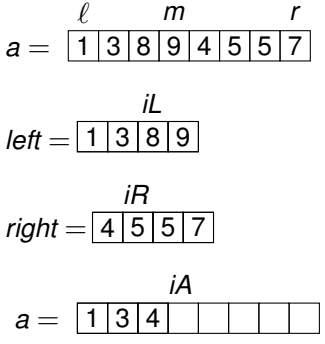
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



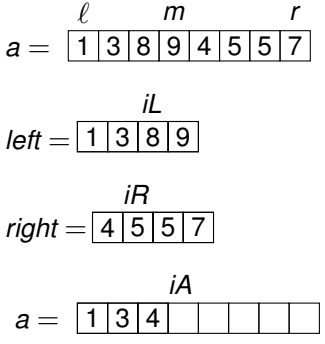
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



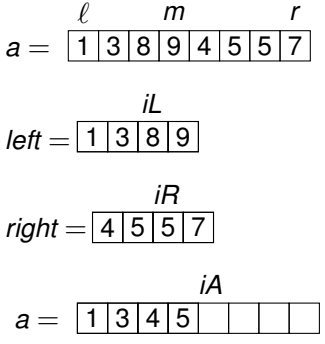
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



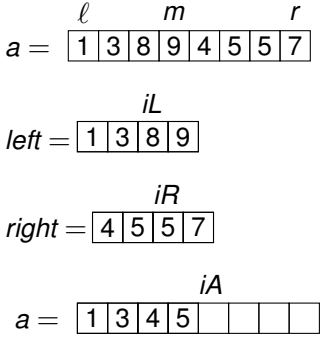
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



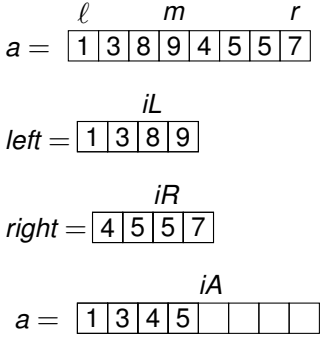
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2   int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3   for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4   for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6   int iL = 0; int iR = 0; int iA = ℓ;
7   while ((iL < m − ℓ + 1) && (iR < r − m)) {
8       if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9       else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12  while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13  while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



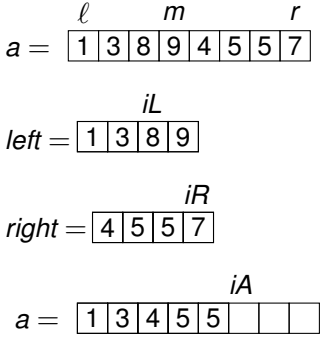
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



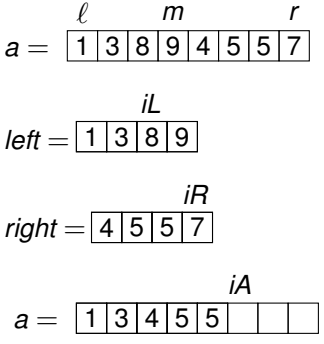
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



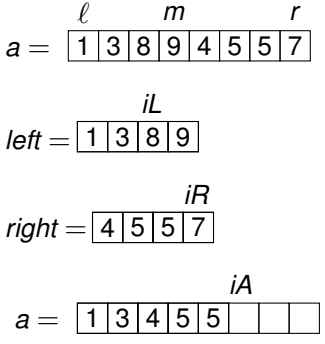
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2   int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3   for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4   for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6   int iL = 0; int iR = 0; int iA = ℓ;
7   while ((iL < m − ℓ + 1) && (iR < r − m)) {
8       if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9       else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12  while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13  while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



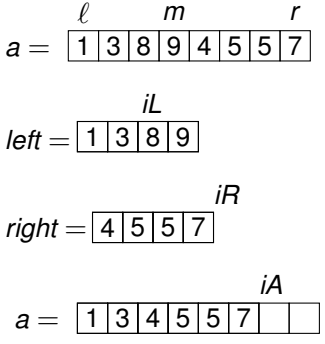
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



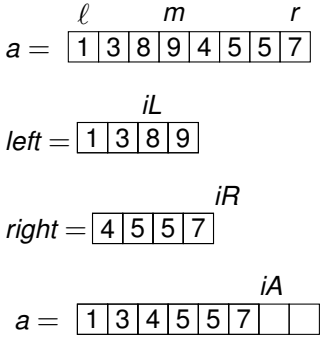
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



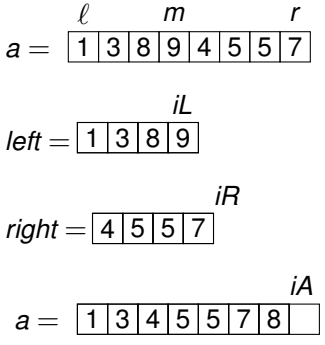
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



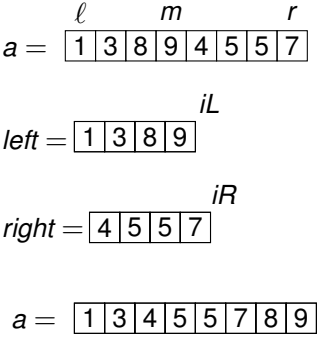
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



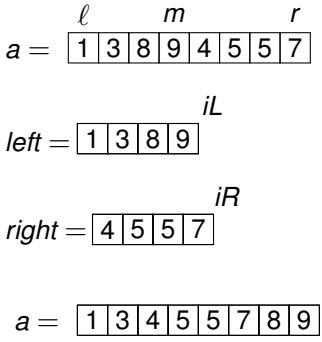
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



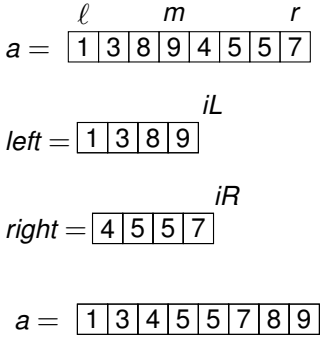
2.1.1 Mergesort

```
MERGE(int[] a, int ℓ, int m, int r)
1   // Kopiere die beiden sortierten Teilfelder in left und right.
2    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
3    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
4    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
5   // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
6    int iL = 0; int iR = 0; int iA = ℓ;
7    while ((iL < m − ℓ + 1) && (iR < r − m)) {
8        if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
9        else {a[iA] = right[iR]; iA++; iR++; }
10  }
11  // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
12   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
13   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



2.1.1 Mergesort

```
1  MERGE(int[] a, int ℓ, int m, int r)
2    // Kopiere die beiden sortierten Teilfelder in left und right.
3    int[] left = new int[m − ℓ + 1]; int[] right = new int[r − m];
4    for (int i = 0; i < m − ℓ + 1; i++) { left[i] = a[ℓ + i]; }
5    for (int i = 0; i < r − m; i++) { right[i] = a[m + 1 + i]; }
6    // Solange beide Teilfelder noch Elemente enthalten, füge sie in a ein.
7    int iL = 0; int iR = 0; int iA = ℓ;
8    while ((iL < m − ℓ + 1) && (iR < r − m)) {
9      if (left[iL] <= right[iR]) { a[iA] = left[iL]; iA++; iL++; }
10     else {a[iA] = right[iR]; iA++; iR++; }
11   }
12   // Füge die übrig gebliebenen Elemente eines Teilfeldes in a ein.
13   while (iL < m − ℓ + 1) { a[iA] = left[iL]; iA++; iL++; }
14   while (iR < r − m) { a[iA] = right[iR]; iA++; iR++; }
```



Laufzeit von MERGESORT

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```

$T(n)$ = **Worst-Case-Laufzeit** von MERGESORT für Eingaben mit n Zahlen

Laufzeit von MERGESORT

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```

$T(n)$ = **Worst-Case-Laufzeit** von MERGESORT für Eingaben mit n Zahlen

Es gilt:

$T(n) \leq 2 \cdot T(n/2) +$ Laufzeit von MERGE

Laufzeit von MERGESORT

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```

$T(n)$ = **Worst-Case-Laufzeit** von MERGESORT für Eingaben mit n Zahlen

Es gilt:

$$T(n) \leq 2 \cdot T(n/2) + \text{Laufzeit von MERGE}$$

Laufzeit von MERGE: $O(n)$ für n Zahlen

Laufzeit von MERGESORT

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```

$T(n)$ = **Worst-Case-Laufzeit** von MERGESORT für Eingaben mit n Zahlen

Es gilt:

$$T(n) \leq 2 \cdot T(n/2) + \text{Laufzeit von MERGE}$$

Laufzeit von MERGE: $O(n)$ für n Zahlen

$$\Rightarrow T(n) \leq 2 \cdot T(n/2) + cn$$

für geeignete Konstante c

Laufzeit von MERGESORT

```
MERGESORT(int[] a, int ℓ, int r)
1    if (ℓ < r) {
2        int m = ℓ + (r − ℓ)/2;
3        MERGESORT(a,ℓ,m);
4        MERGESORT(a,m + 1,r);
5        MERGE(a,ℓ,m,r);
6    }
```

$T(n)$ = **Worst-Case-Laufzeit** von MERGESORT für Eingaben mit n Zahlen

Es gilt:

$$T(n) \leq 2 \cdot T(n/2) + \text{Laufzeit von MERGE}$$

Laufzeit von MERGE: $O(n)$ für n Zahlen

$$\Rightarrow T(n) \leq 2 \cdot T(n/2) + cn$$

für geeignete Konstante c

Außerdem $T(1) \leq c$

2.1.1 Mergesort

Theorem 2.2

Die Laufzeit von MERGESORT liegt in $O(n \log n)$.

Theorem 2.2

Die Laufzeit von MERGESORT liegt in $O(n \log n)$.

Beweis:

Wir betrachten nur $n = 2^k$ für ein k.

Theorem 2.2

Die Laufzeit von MERGESORT liegt in $O(n \log n)$.

Beweis:

Wir betrachten nur $n = 2^k$ für ein k.

zu zeigen: $\exists n_0 \in \mathbb{N} : \exists c^\star \in \mathbb{R} : \forall n \geq n_0 : T(n) \leq c^\star \cdot n \log_2(n)$

Theorem 2.2

Die Laufzeit von MERGESORT liegt in $O(n \log n)$.

Beweis:

Wir betrachten nur $n = 2^k$ für ein k.

zu zeigen: $\exists n_0 \in \mathbb{N} : \exists c^\star \in \mathbb{R} : \forall n \geq n_0 : T(n) \leq c^\star \cdot n \log_2(n)$

Wegen $\log_2(1) = 0$ setzen wir $n_0 = 2$.

Theorem 2.2

Die Laufzeit von MERGESORT liegt in $O(n \log n)$.

Beweis:

Wir betrachten nur $n = 2^k$ für ein k.

zu zeigen: $\exists n_0 \in \mathbb{N} : \exists c^\star \in \mathbb{R} : \forall n \geq n_0 : T(n) \leq c^\star \cdot n \log_2(n)$

Wegen $\log_2(1) = 0$ setzen wir $n_0 = 2$.

Induktionsanfang: $T(2) \leq c^\star \cdot 2 \log_2(2)$ gilt für hinreichend großes $c^\star$.

Induktionsschritt:

Sei $n = 2^k$ für $k > 1$. Induktionsannahme $T(n/2) \leq c^\star \cdot (n/2) \log_2(n/2)$.

Induktionsschritt:

Sei $n = 2^k$ für $k > 1$. Induktionsannahme $T(n/2) \leq c^\star \cdot (n/2) \log_2(n/2)$.

$$T(n) \leq 2 \cdot T(n/2) + cn$$

Induktionsschritt:

Sei $n = 2^k$ für $k > 1$. Induktionsannahme $T(n/2) \leq c^\star \cdot (n/2) \log_2(n/2)$.

$$\begin{aligned} T(n) &\leq 2 \cdot T(n/2) + cn \\ &\leq 2 \cdot (c^\star \cdot (n/2) \cdot \log_2(n/2)) + cn \end{aligned}$$

Induktionsschritt:

Sei $n = 2^k$ für $k > 1$. Induktionsannahme $T(n/2) \leq c^\star \cdot (n/2) \log_2(n/2)$.

$$\begin{aligned} T(n) &\leq 2 \cdot T(n/2) + cn \\ &\leq 2 \cdot (c^\star \cdot (n/2) \cdot \log_2(n/2)) + cn \\ &= c^\star \cdot n \log_2(n/2) + cn \end{aligned}$$

Induktionsschritt:

Sei $n = 2^k$ für $k > 1$. Induktionsannahme $T(n/2) \leq c^\star \cdot (n/2) \log_2(n/2)$.

$$\begin{aligned} T(n) &\leq 2 \cdot T(n/2) + cn \\ &\leq 2 \cdot (c^\star \cdot (n/2) \cdot \log_2(n/2)) + cn \\ &= c^\star \cdot n \log_2(n/2) + cn \\ &= c^\star \cdot n(\log_2(n) - 1) + cn \end{aligned}$$

Induktionsschritt:

Sei $n = 2^k$ für $k > 1$. Induktionsannahme $T(n/2) \leq c^\star \cdot (n/2) \log_2(n/2)$.

$$
\begin{aligned}
T(n) &\leq 2 \cdot T(n/2) + cn \\
&\leq 2 \cdot (c^\star \cdot (n/2) \cdot \log_2(n/2)) + cn \\
&= c^\star \cdot n \log_2(n/2) + cn \\
&= c^\star \cdot n(\log_2(n) - 1) + cn \\
&= c^\star \cdot n \log_2(n) - c^\star n + cn
\end{aligned}
$$

Induktionsschritt:

Sei $n = 2^k$ für $k > 1$. Induktionsannahme $T(n/2) \leq c^\star \cdot (n/2) \log_2(n/2)$.

$$
\begin{aligned}
T(n) &\leq 2 \cdot T(n/2) + cn \\
&\leq 2 \cdot (c^\star \cdot (n/2) \cdot \log_2(n/2)) + cn \\
&= c^\star \cdot n \log_2(n/2) + cn \\
&= c^\star \cdot n(\log_2(n) - 1) + cn \\
&= c^\star \cdot n \log_2(n) - c^\star n + cn \\
&\leq c^\star \cdot n \log_2(n)
\end{aligned}
$$

für $c^\star \geq c$ □

Induktionsschritt:

Sei $n = 2^k$ für $k > 1$. Induktionsannahme $T(n/2) \leq c^\star \cdot (n/2) \log_2(n/2)$.

$$\begin{aligned} T(n) &\leq 2 \cdot T(n/2) + cn \\ &\leq 2 \cdot (c^\star \cdot (n/2) \cdot \log_2(n/2)) + cn \\ &= c^\star \cdot n \log_2(n/2) + cn \\ &= c^\star \cdot n(\log_2(n) - 1) + cn \\ &= c^\star \cdot n \log_2(n) - c^\star n + cn \\ &\leq c^\star \cdot n \log_2(n) \end{aligned}$$

für $c^\star \geq c$ □

Analog kann man auch $T(n) = \Omega(n \log n)$ argumentieren.

2.1.1 Mergesort

Falscher „Beweis“ für $T(n) = O(n)$:

2.1.1 Mergesort

Falscher „Beweis“ für $T(n) = O(n)$:

Induktionsanfang: $T(1) = O(1)$

2.1.1 Mergesort

Falscher „Beweis“ für $T(n) = O(n)$:

Induktionsanfang: $T(1) = O(1)$

Induktionsschritt: Sei $n = 2^k$ für ein $k \in \mathbb{N}$.

Induktionsannahme: $T(n/2) \leq O(n/2)$

2.1.1 Mergesort

Falscher „Beweis“ für $T(n) = O(n)$:

Induktionsanfang: $T(1) = O(1)$

Induktionsschritt: Sei $n = 2^k$ für ein $k \in \mathbb{N}$.

Induktionsannahme: $T(n/2) \leq O(n/2)$

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn \\ &= O(n/2) + O(n) \\ &= O(n) + O(n) \\ &= O(n) \end{aligned}$$

2.1.1 Mergesort

Falscher „Beweis“ für $T(n) = O(n)$:

Induktionsanfang: $T(1) = O(1)$

Induktionsschritt: Sei $n = 2^k$ für ein $k \in \mathbb{N}$.

Induktionsannahme: $T(n/2) \leq O(n/2)$

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn \\ &= O(n/2) + O(n) \\ &= O(n) + O(n) \\ &= O(n) \end{aligned}$$

Achtung: Dies zeigt lediglich die triviale Aussage

$$\exists n_0 \in \mathbb{N} : \forall n \geq n_0 : \exists c \in \mathbb{R} : T(n) \leq cn.$$

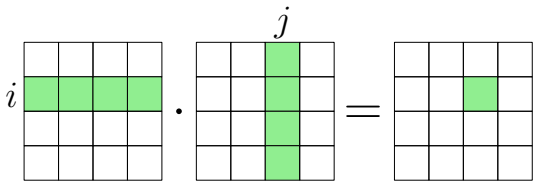
2.1.2 Strassen-Algorithmus

Matrixmultiplikation

Seien $A = (a_{ij})$ und $B = (b_{ij})$ reelle $n \times n$-Matrizen und sei $C = A \cdot B$.

C ist ebenfalls $n \times n$-Matrix $C = (c_{ij})$ mit

$$\forall i, j \in \{1, \ldots, n\} : \mathbf{c_{ij}} = \sum_{\mathbf{k=1}}^{\mathbf{n}} \mathbf{a_{ik} b_{kj}}.$$



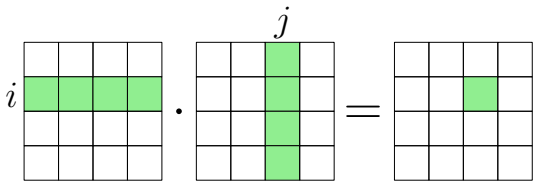
2.1.2 Strassen-Algorithmus

Matrixmultiplikation

Seien $A = (a_{ij})$ und $B = (b_{ij})$ reelle $n \times n$-Matrizen und sei $C = A \cdot B$.

C ist ebenfalls $n \times n$-Matrix $C = (c_{ij})$ mit

$$\forall i, j \in \{1, \ldots, n\} : \mathbf{c_{ij}} = \sum_{\mathbf{k=1}}^{\mathbf{n}} \mathbf{a_{ik} b_{kj}}.$$



Einfacher Algorithmus: Berechne C **direkt gemäß Definition**.

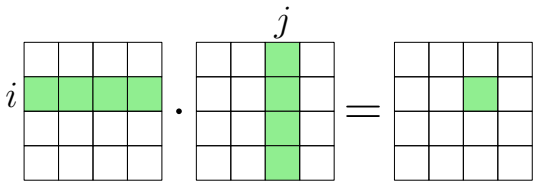
2.1.2 Strassen-Algorithmus

Matrixmultiplikation

Seien $A = (a_{ij})$ und $B = (b_{ij})$ reelle $n \times n$-Matrizen und sei $C = A \cdot B$.

C ist ebenfalls $n \times n$-Matrix $C = (c_{ij})$ mit

$$\forall i, j \in \{1, \ldots, n\} : \mathbf{c_{ij}} = \sum_{\mathbf{k=1}}^{\mathbf{n}} \mathbf{a_{ik} b_{kj}}.$$



Einfacher Algorithmus: Berechne C **direkt gemäß Definition**.

Laufzeit $\Theta(n^3)$

2.1.2 Strassen-Algorithmus

Divide-and-Conquer-Algorithmus SIMPLEPRODUCT**:**

2.1.2 Strassen-Algorithmus

Divide-and-Conquer-Algorithmus SIMPLEPRODUCT:

Zerlege A, B, C jeweils in vier $(n/2) \times (n/2)$-Teilmatrizen:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \qquad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

2.1.2 Strassen-Algorithmus

Divide-and-Conquer-Algorithmus SIMPLEPRODUCT:

Zerlege A, B, C jeweils in vier $(n/2) \times (n/2)$-Teilmatrizen:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \qquad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

Für diese Matrizen gilt:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \qquad C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22},$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \qquad C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}.$$

2.1.2 Strassen-Algorithmus

Divide-and-Conquer-Algorithmus SIMPLEPRODUCT:

Zerlege A, B, C jeweils in vier $(n/2) \times (n/2)$-Teilmatrizen:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \qquad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

Für diese Matrizen gilt:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \qquad C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22},$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \qquad C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}.$$

SIMPLEPRODUCT: Berechne $C_{11}, C_{12}, C_{21}, C_{22}$ gemäß dieser Formeln.

2.1.2 Strassen-Algorithmus

Divide-and-Conquer-Algorithmus SIMPLEPRODUCT:

Zerlege A, B, C jeweils in vier $(n/2) \times (n/2)$-Teilmatrizen:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \qquad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

Für diese Matrizen gilt:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \qquad C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22},$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \qquad C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}.$$

SIMPLEPRODUCT: Berechne $C_{11}, C_{12}, C_{21}, C_{22}$ gemäß dieser Formeln.

Laufzeit:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1, \\ 8T(n/2) + \Theta(n^2) & \text{falls } n = 2^k \text{ für } k \in \mathbb{N}. \end{cases}$$

2.1.2 Strassen-Algorithmus

STRASSEN(A,B)

1 Falls $n = 1$: berechne $A \cdot B$ mit einer Multiplikation und gib das Ergebnis zurück.

2 Ansonsten zerlege A und B wie oben beschrieben in jeweils vier $(n/2) \times (n/2)$-Matrizen A_{ij} und B_{ij}.

3 Berechne die folgenden zehn $(n/2) \times (n/2)$-Matrizen:

$$S_1 = B_{12} - B_{22}, \quad S_2 = A_{11} + A_{12}, \quad S_3 = A_{21} + A_{22}, \quad S_4 = B_{21} - B_{11}, \quad S_5 = A_{11} + A_{22},$$

$$S_6 = B_{11} + B_{22} \quad S_7 = A_{12} - A_{22}, \quad S_8 = B_{21} + B_{22}, \quad S_9 = A_{11} - A_{21}, \quad S_{10} = B_{11} + B_{12}.$$

4 Berechne rekursiv die folgenden Produkte von $(n/2) \times (n/2)$-Matrizen:

$$P_1 = A_{11} \cdot S_1, \quad P_2 = S_2 \cdot B_{22}, \quad P_3 = S_3 \cdot B_{11}, \quad P_4 = A_{22} \cdot S_4,$$

$$P_5 = S_5 \cdot S_6, \quad P_6 = S_7 \cdot S_8, \quad P_7 = S_9 \cdot S_{10}.$$

5 Berechne die Teilmatrizen von C wie folgt:

$$C_{11} = P_5 + P_4 - P_2 + P_6, \quad C_{12} = P_1 + P_2,$$

$$C_{21} = P_3 + P_4, \quad C_{22} = P_5 + P_1 - P_3 - P_7.$$

2.1.2 Strassen-Algorithmus

Laufzeit von STRASSEN:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1, \\ 7T(n/2) + \Theta(n^2) & \text{falls } n = 2^k \text{ für } k \in \mathbb{N}. \end{cases}$$

2.1.2 Strassen-Algorithmus

Laufzeit von STRASSEN:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1, \\ 7T(n/2) + \Theta(n^2) & \text{falls } n = 2^k \text{ für } k \in \mathbb{N}. \end{cases}$$

Laufzeit von SIMPLEPRODUCT:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1, \\ 8T(n/2) + \Theta(n^2) & \text{falls } n = 2^k \text{ für } k \in \mathbb{N}. \end{cases}$$

2.1.2 Strassen-Algorithmus

Laufzeit von STRASSEN:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1, \\ 7T(n/2) + \Theta(n^2) & \text{falls } n = 2^k \text{ für } k \in \mathbb{N}. \end{cases}$$

Laufzeit von SIMPLEPRODUCT:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1, \\ 8T(n/2) + \Theta(n^2) & \text{falls } n = 2^k \text{ für } k \in \mathbb{N}. \end{cases}$$

Lösung der Rekursionsgleichung: $T(n) = \Theta(n^3)$

2.1.2 Strassen-Algorithmus

Laufzeit von STRASSEN:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1, \\ 7T(n/2) + \Theta(n^2) & \text{falls } n = 2^k \text{ für } k \in \mathbb{N}. \end{cases}$$

Lösung der Rekursionsgleichung: $T(n) = \Theta(n^{\log_2 7}) = O(n^{2,81})$

Laufzeit von SIMPLEPRODUCT:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1, \\ 8T(n/2) + \Theta(n^2) & \text{falls } n = 2^k \text{ für } k \in \mathbb{N}. \end{cases}$$

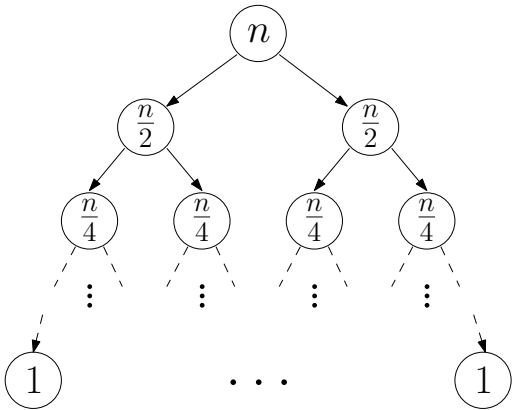
Lösung der Rekursionsgleichung: $T(n) = \Theta(n^3)$

2.1.3 Lösen von Rekursionsgleichungen

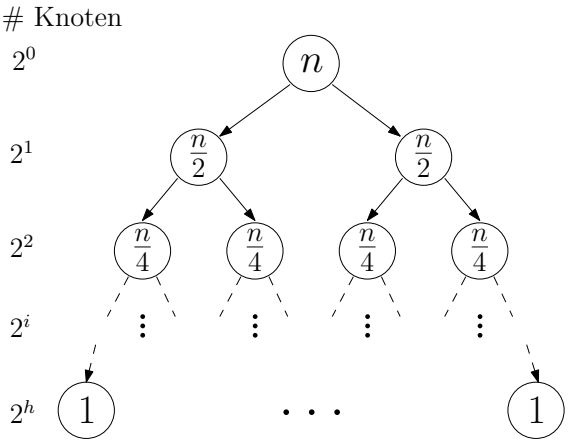
Rekursionsgleichungen für die Laufzeiten der betrachteten Algorithmen:

$$\begin{aligned}
\textsc{BinarySearch}: &\quad T(n) = T(n/2) + \Theta(1),\\
\textsc{Mergesort}: &\quad T(n) = 2T(n/2) + \Theta(n),\\
\textsc{SimpleProduct}: &\quad T(n) = 8T(n/2) + \Theta(n^2),\\
\textsc{Strassen}: &\quad T(n) = 7T(n/2) + \Theta(n^2).
\end{aligned}$$

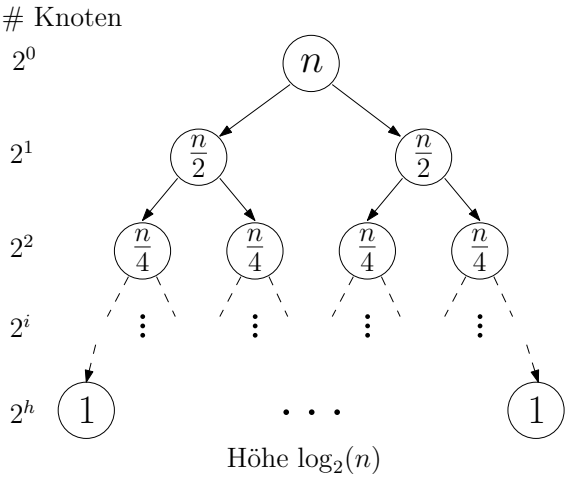
Rekursionsbaum von MERGESORT: $T(n) = 2T(n/2) + \Theta(n)$



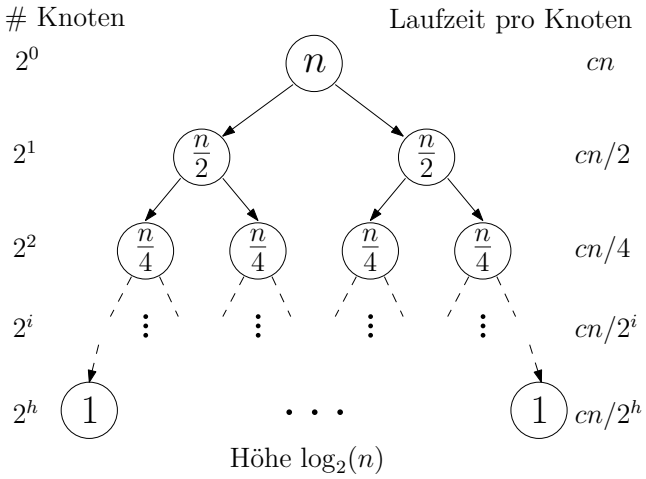
Rekursionsbaum von MERGESORT: $T(n) = 2T(n/2) + \Theta(n)$



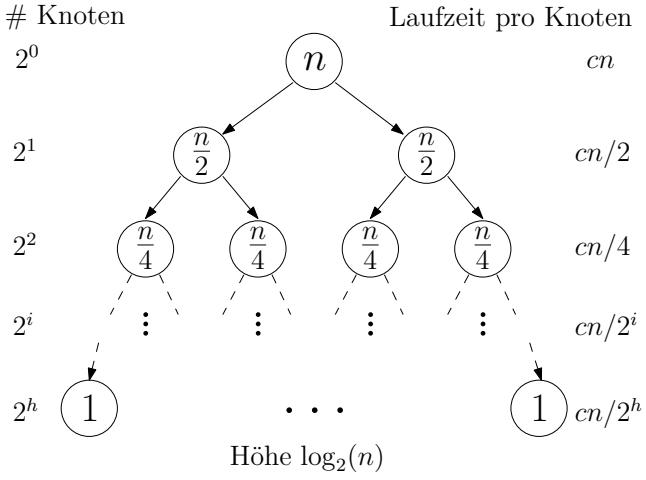
Rekursionsbaum von MERGESORT: $T(n) = 2T(n/2) + \Theta(n)$



Rekursionsbaum von MERGESORT: $T(n) = 2T(n/2) + \Theta(n)$



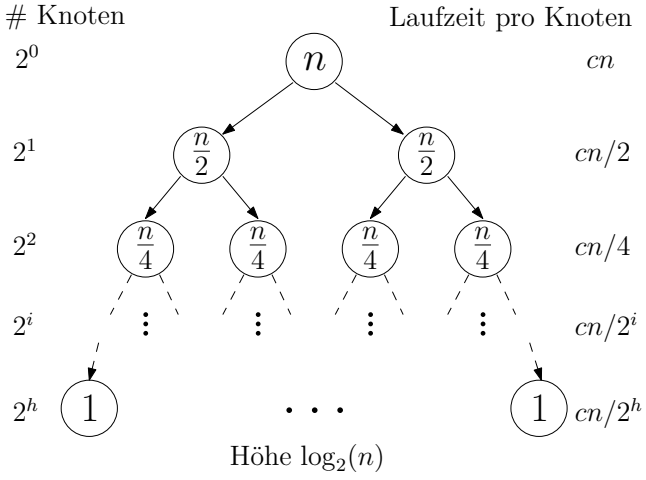
Rekursionsbaum von MERGESORT: $T(n) = 2T(n/2) + \Theta(n)$



Addition der Laufzeiten aller Knoten:

$$T(n) \leq \sum_{i=0}^{\log_2 n} 2^i \cdot \frac{cn}{2^i}$$

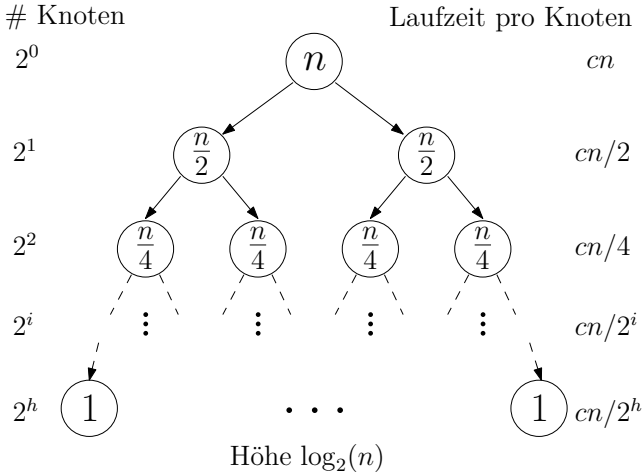
Rekursionsbaum von MERGESORT: $T(n) = 2T(n/2) + \Theta(n)$



Addition der Laufzeiten aller Knoten:

$$
\begin{aligned}
T(n) &\leq \sum_{i=0}^{\log_2 n} 2^i \cdot \frac{cn}{2^i} \\
&= cn \sum_{i=0}^{\log_2 n} 1 \\
&= cn(\log_2 n + 1).
\end{aligned}
$$

Rekursionsbaum von MERGESORT: $T(n) = 2T(n/2) + \Theta(n)$



Addition der Laufzeiten aller Knoten:

$$T(n) \leq \sum_{i=0}^{\log_2 n} 2^i \cdot \frac{cn}{2^i}$$

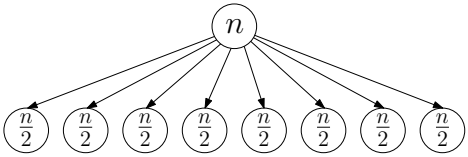
$$= cn \sum_{i=0}^{\log_2 n} 1$$

$$= cn(\log_2 n + 1).$$

$$\Rightarrow T(n) = O(n \log n)$$

Rekursionsbaum von SIMPLEPRODUCT:

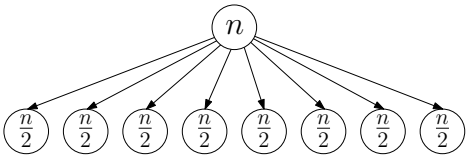
$T(n) = 8T(n/2) + \Theta(n^2)$



2.1.3 Lösen von Rekursionsgleichungen

Rekursionsbaum von SIMPLEPRODUCT:

$T(n) = 8T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

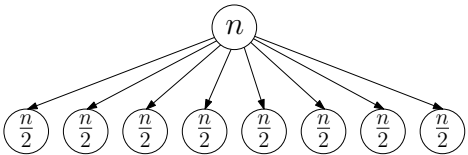
 # Teilprobleme auf Level i: 8^i

Größe Teilprobleme auf Level i: $n/2^i$

2.1.3 Lösen von Rekursionsgleichungen

Rekursionsbaum von SIMPLEPRODUCT:

$T(n) = 8T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

 # Teilprobleme auf Level i: 8^i

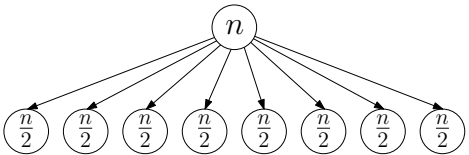
Größe Teilprobleme auf Level i: $n/2^i$

$$T(n) \leq \sum_{i=0}^{\log_2 n} 8^i \cdot c\left(\frac{n}{2^i}\right)^2$$

2.1.3 Lösen von Rekursionsgleichungen

Rekursionsbaum von SIMPLEPRODUCT:

$T(n) = 8T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

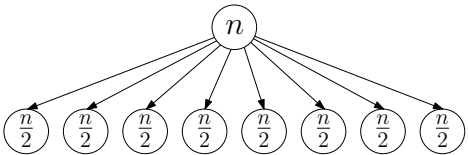
 # Teilprobleme auf Level i: 8^i

Größe Teilprobleme auf Level i: $n/2^i$

$$
\begin{aligned}
T(n) &\leq \sum_{i=0}^{\log_2 n} 8^i \cdot c\left(\frac{n}{2^i}\right)^2 \\
&= cn^2 \sum_{i=0}^{\log_2 n} \frac{8^i}{2^{2i}} \\
&= cn^2 \sum_{i=0}^{\log_2 n} 2^i
\end{aligned}
$$

Rekursionsbaum von SIMPLEPRODUCT:

$T(n) = 8T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

 # Teilprobleme auf Level i: 8^i

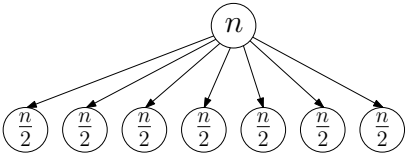
Größe Teilprobleme auf Level i: $n/2^i$

$$
\begin{aligned}
T(n) &\leq \sum_{i=0}^{\log_2 n} 8^i \cdot c\left(\frac{n}{2^i}\right)^2 \\
&= cn^2 \sum_{i=0}^{\log_2 n} \frac{8^i}{2^{2i}} \\
&= cn^2 \sum_{i=0}^{\log_2 n} 2^i \\
&= cn^2(2^{\log_2(n)+1} - 1) \\
&= O(n^3)
\end{aligned}
$$

2.1.3 Lösen von Rekursionsgleichungen

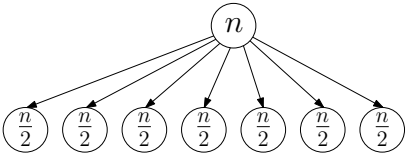
Rekursionsbaum von STRASSEN:

$T(n) = 7T(n/2) + \Theta(n^2)$



Rekursionsbaum von STRASSEN:

$T(n) = 7T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

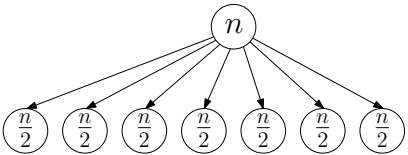
 # Teilprobleme auf Level i: 7^i

Größe Teilprobleme auf Level i: $n/2^i$

2.1.3 Lösen von Rekursionsgleichungen

Rekursionsbaum von STRASSEN:

$T(n) = 7T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

 # Teilprobleme auf Level i: 7^i

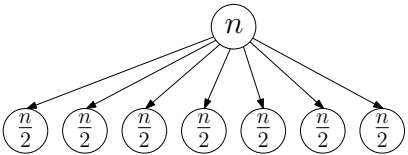
Größe Teilprobleme auf Level i: $n/2^i$

$$T(n) \leq \sum_{i=0}^{\log_2 n} 7^i \cdot c\left(\frac{n}{2^i}\right)^2$$

2.1.3 Lösen von Rekursionsgleichungen

Rekursionsbaum von STRASSEN:

$T(n) = 7T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

 # Teilprobleme auf Level i: 7^i

Größe Teilprobleme auf Level i: $n/2^i$

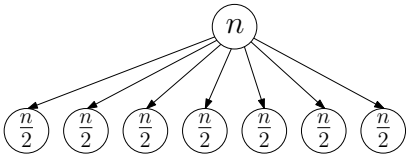
$$T(n) \leq \sum_{i=0}^{\log_2 n} 7^i \cdot c\left(\frac{n}{2^i}\right)^2 = cn^2 \sum_{i=0}^{\log_2 n} \frac{7^i}{2^{2i}}$$

$$= cn^2 \sum_{i=0}^{\log_2 n} \left(\frac{7}{4}\right)^i$$

2.1.3 Lösen von Rekursionsgleichungen

Rekursionsbaum von STRASSEN:

$T(n) = 7T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

 # Teilprobleme auf Level i: 7^i

Größe Teilprobleme auf Level i: $n/2^i$

$$T(n) \leq \sum_{i=0}^{\log_2 n} 7^i \cdot c\left(\frac{n}{2^i}\right)^2 = cn^2 \sum_{i=0}^{\log_2 n} \frac{7^i}{2^{2i}}$$

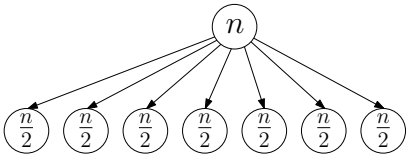
$$= cn^2 \sum_{i=0}^{\log_2 n} \left(\frac{7}{4}\right)^i = cn^2 \frac{\left(\frac{7}{4}\right)^{\log_2(n)+1} - 1}{\left(\frac{7}{4}\right) - 1}$$

$\sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1}$ für $q > 1$ und $n \in \mathbb{N}$

2.1.3 Lösen von Rekursionsgleichungen

Rekursionsbaum von STRASSEN:

$T(n) = 7T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

 # Teilprobleme auf Level i: 7^i

Größe Teilprobleme auf Level i: $n/2^i$

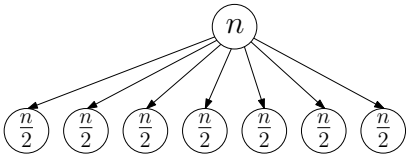
$$
\begin{aligned}
T(n) &\leq \sum_{i=0}^{\log_2 n} 7^i \cdot c\left(\frac{n}{2^i}\right)^2 = cn^2 \sum_{i=0}^{\log_2 n} \frac{7^i}{2^{2i}} \\
&= cn^2 \sum_{i=0}^{\log_2 n} \left(\frac{7}{4}\right)^i = cn^2 \frac{\left(\frac{7}{4}\right)^{\log_2(n)+1} - 1}{\left(\frac{7}{4}\right) - 1} \\
&= O\left(n^2 \left(\frac{7}{4}\right)^{\log_2 n}\right)
\end{aligned}
$$

$\sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1}$ für $q > 1$ und $n \in \mathbb{N}$

2.1.3 Lösen von Rekursionsgleichungen

Rekursionsbaum von STRASSEN:

$T(n) = 7T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

 # Teilprobleme auf Level i: 7^i

Größe Teilprobleme auf Level i: $n/2^i$

$$T(n) \leq \sum_{i=0}^{\log_2 n} 7^i \cdot c\left(\frac{n}{2^i}\right)^2 = cn^2 \sum_{i=0}^{\log_2 n} \frac{7^i}{2^{2i}}$$

$$= cn^2 \sum_{i=0}^{\log_2 n} \left(\frac{7}{4}\right)^i = cn^2 \frac{\left(\frac{7}{4}\right)^{\log_2(n)+1} - 1}{\left(\frac{7}{4}\right) - 1}$$

$$= O\left(n^2 \left(\frac{7}{4}\right)^{\log_2 n}\right) = O\left(n^2 n^{\log_2(7/4)}\right)$$

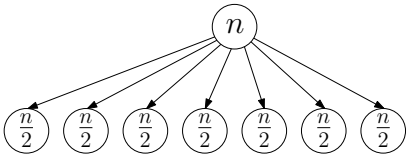
$\sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1}$ für $q > 1$ und $n \in \mathbb{N}$

$a^{\log_2 b} = b^{\log_2 a}$ für $a, b > 1$

2.1.3 Lösen von Rekursionsgleichungen

Rekursionsbaum von STRASSEN:

$T(n) = 7T(n/2) + \Theta(n^2)$



Höhe: $\log_2(n)$

 # Teilprobleme auf Level i: 7^i

Größe Teilprobleme auf Level i: $n/2^i$

$$
\begin{aligned}
T(n) &\leq \sum_{i=0}^{\log_2 n} 7^i \cdot c\left(\frac{n}{2^i}\right)^2 = cn^2 \sum_{i=0}^{\log_2 n} \frac{7^i}{2^{2i}} \\
&= cn^2 \sum_{i=0}^{\log_2 n} \left(\frac{7}{4}\right)^i = cn^2 \frac{\left(\frac{7}{4}\right)^{\log_2(n)+1} - 1}{\left(\frac{7}{4}\right) - 1} \\
&= O\left(n^2 \left(\frac{7}{4}\right)^{\log_2 n}\right) = O\left(n^2 n^{\log_2(7/4)}\right) \\
&= O\left(n^2 n^{\log_2(7)-2}\right) \\
&= O\left(n^{\log_2 7}\right).
\end{aligned}
$$

$\sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1}$ für $q > 1$ und $n \in \mathbb{N}$

$a^{\log_2 b} = b^{\log_2 a}$ für $a, b > 1$

Theorem 2.3

Seien $a \geq 1$ und $b > 1$ und $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$.

Sei $T : \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und für $n \geq 2$

$$T(n) = aT(n/b) + f(n).$$

Dabei steht n/b entweder für $\lceil n/b \rceil$ oder $\lfloor n/b \rfloor$.

Theorem 2.3

Seien $a \geq 1$ und $b > 1$ und $f\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$.

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und für $n \geq 2$

$$T(n) = aT(n/b) + f(n).$$

Dabei steht n/b entweder für $\lceil n/b \rceil$ oder $\lfloor n/b \rfloor$.

1. Falls $f(n) = O(n^{\log_b a - \varepsilon})$ für eine Konstante $\varepsilon > 0$ gilt, so gilt $T(n) = \Theta(n^{\log_b a})$.

2.1.3 Lösen von Rekursionsgleichungen

Theorem 2.3

Seien $a \geq 1$ und $b > 1$ und $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$.

Sei $T : \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und für $n \geq 2$

$$T(n) = aT(n/b) + f(n).$$

Dabei steht n/b entweder für $\lceil n/b \rceil$ oder $\lfloor n/b \rfloor$.

1. Falls $f(n) = O(n^{\log_b a - \varepsilon})$ für eine Konstante $\varepsilon > 0$ gilt, so gilt $T(n) = \Theta(n^{\log_b a})$.
2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

2.1.3 Lösen von Rekursionsgleichungen

Theorem 2.3

Seien $a \geq 1$ und $b > 1$ und $f\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$.

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und für $n \geq 2$

$$T(n) = aT(n/b) + f(n).$$

Dabei steht n/b entweder für $\lceil n/b \rceil$ oder $\lfloor n/b \rfloor$.

1. Falls $f(n) = O(n^{\log_b a - \varepsilon})$ für eine Konstante $\varepsilon > 0$ gilt, so gilt $T(n) = \Theta(n^{\log_b a})$.
2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
3. Falls $f(n) = \Omega(n^{\log_b a + \varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n gilt, so gilt $T(n) = \Theta(f(n))$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

1. Falls $f(n) = O(n^{\log_b a - \varepsilon})$ für eine Konstante $\varepsilon > 0$ gilt, so gilt $T(n) = \Theta(n^{\log_b a})$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

1. Falls $f(n) = O(n^{\log_b a - \varepsilon})$ für eine Konstante $\varepsilon > 0$ gilt, so gilt $T(n) = \Theta(n^{\log_b a})$.

Beispiele

- $T(n) = 8T(n/2) + \Theta(n^2)$
  (Rekursionsgleichung von SIMPLEPRODUCT)

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

1. Falls $f(n) = O(n^{\log_b a - \varepsilon})$ für eine Konstante $\varepsilon > 0$ gilt, so gilt $T(n) = \Theta(n^{\log_b a})$.

Beispiele

- $T(n) = 8T(n/2) + \Theta(n^2)$
  (Rekursionsgleichung von SIMPLEPRODUCT)
  Es gilt $a = 8$ und $b = 2$ und $f(n) = n^2$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

1. Falls $f(n) = O(n^{\log_b a - \varepsilon})$ für eine Konstante $\varepsilon > 0$ gilt, so gilt $T(n) = \Theta(n^{\log_b a})$.

Beispiele

- $T(n) = 8T(n/2) + \Theta(n^2)$
  (Rekursionsgleichung von SIMPLEPRODUCT)
  Es gilt $a = 8$ und $b = 2$ und $f(n) = n^2$.
  Damit gilt $n^{\log_b a} = n^3$ und $f(n) = n^2 = O(n^{\log_b a - \varepsilon})$ für $\varepsilon = 1$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

1. Falls $f(n) = O(n^{\log_b a - \varepsilon})$ für eine Konstante $\varepsilon > 0$ gilt, so gilt $T(n) = \Theta(n^{\log_b a})$.

Beispiele

- $T(n) = 8T(n/2) + \Theta(n^2)$
  (Rekursionsgleichung von SIMPLEPRODUCT)
  Es gilt $a = 8$ und $b = 2$ und $f(n) = n^2$.
  Damit gilt $n^{\log_b a} = n^3$ und $f(n) = n^2 = O(n^{\log_b a - \varepsilon})$ für $\varepsilon = 1$.
  $\Rightarrow T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

Beispiele

- $T(n) = kT(n/k) + n$ für $k \in \mathbb{N}$ mit $k \geq 2$

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

Beispiele

- $T(n) = kT(n/k) + n$ für $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

Beispiele

- $T(n) = kT(n/k) + n$ für $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n$.
  Damit gilt $n^{\log_b a} = n$ und $f(n) = n = \Theta(n^{\log_b a})$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

Beispiele

- $T(n) = kT(n/k) + n$ für $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n$.
  Damit gilt $n^{\log_b a} = n$ und $f(n) = n = \Theta(n^{\log_b a})$.
  $\Rightarrow T(n) = \Theta(n \log n)$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

Beispiele

- $T(n) = kT(n/k) + n$ für $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n$.
  Damit gilt $n^{\log_b a} = n$ und $f(n) = n = \Theta(n^{\log_b a})$.
  $\Rightarrow T(n) = \Theta(n \log n)$.
- $T(n) = T(2n/3) + 1$

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

Beispiele

- $T(n) = kT(n/k) + n$ für $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n$.
  Damit gilt $n^{\log_b a} = n$ und $f(n) = n = \Theta(n^{\log_b a})$.
  $\Rightarrow T(n) = \Theta(n \log n)$.
- $T(n) = T(2n/3) + 1$
  Es gilt $a = 1$, $b = 3/2$ und $f(n) = 1$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

Beispiele

- $T(n) = kT(n/k) + n$ für $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n$.
  Damit gilt $n^{\log_b a} = n$ und $f(n) = n = \Theta(n^{\log_b a})$.
  $\Rightarrow T(n) = \Theta(n \log n)$.
- $T(n) = T(2n/3) + 1$
  Es gilt $a = 1$, $b = 3/2$ und $f(n) = 1$.
  Damit gilt $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$ und $f(n) = 1 = \Theta(n^{\log_b a})$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

2. Falls $f(n) = \Theta(n^{\log_b a})$ gilt, so gilt $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

Beispiele

- $T(n) = kT(n/k) + n$ für $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n$.
  Damit gilt $n^{\log_b a} = n$ und $f(n) = n = \Theta(n^{\log_b a})$.
  $\Rightarrow T(n) = \Theta(n \log n)$.
- $T(n) = T(2n/3) + 1$
  Es gilt $a = 1$, $b = 3/2$ und $f(n) = 1$.
  Damit gilt $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$ und $f(n) = 1 = \Theta(n^{\log_b a})$.
  $\Rightarrow T(n) = \Theta(\log n)$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

3. Falls $f(n) = \Omega(n^{\log_b a + \varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n gilt, so gilt $T(n) = \Theta(f(n))$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

3. Falls $f(n) = \Omega(n^{\log_b a + \varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n gilt, so gilt $T(n) = \Theta(f(n))$.

Beispiele

- $T(n) = kT(n/k) + n\log_2 n$ für ein $k \in \mathbb{N}$ mit $k \geq 2$

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

3. Falls $f(n) = \Omega(n^{\log_b a + \varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n gilt, so gilt $T(n) = \Theta(f(n))$.

Beispiele

- $T(n) = kT(n/k) + n\log_2 n$ für ein $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n\log_2 n$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

3. Falls $f(n) = \Omega(n^{\log_b a+\varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n gilt, so gilt $T(n) = \Theta(f(n))$.

Beispiele

- $T(n) = kT(n/k) + n\log_2 n$ für ein $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n\log_2 n$.
  Zwar wächst $f(n) = n\log_2 n$ schneller als $n^{\log_b a} = n$, es kann jedoch trotzdem nicht der dritte Fall angewendet werden.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

3. Falls $f(n) = \Omega(n^{\log_b a+\varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n gilt, so gilt $T(n) = \Theta(f(n))$.

Beispiele

- $T(n) = kT(n/k) + n\log_2 n$ für ein $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n\log_2 n$.
  Zwar wächst $f(n) = n\log_2 n$ schneller als $n^{\log_b a} = n$, es kann jedoch trotzdem nicht der dritte Fall angewendet werden.
- $T(n) = 8T(n/2) + \Theta(n^4)$

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

3. Falls $f(n) = \Omega(n^{\log_b a + \varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n gilt, so gilt $T(n) = \Theta(f(n))$.

Beispiele

- $T(n) = kT(n/k) + n\log_2 n$ für ein $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n\log_2 n$.
  Zwar wächst $f(n) = n\log_2 n$ schneller als $n^{\log_b a} = n$, es kann jedoch trotzdem nicht der dritte Fall angewendet werden.
- $T(n) = 8T(n/2) + \Theta(n^4)$
  Es gilt $a = 8$ und $b = 2$ und $f(n) = n^4$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

3. Falls $f(n) = \Omega(n^{\log_b a+\varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n gilt, so gilt $T(n) = \Theta(f(n))$.

Beispiele

- $T(n) = kT(n/k) + n\log_2 n$ für ein $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n\log_2 n$.
  Zwar wächst $f(n) = n\log_2 n$ schneller als $n^{\log_b a} = n$, es kann jedoch trotzdem nicht der dritte Fall angewendet werden.
- $T(n) = 8T(n/2) + \Theta(n^4)$
  Es gilt $a = 8$ und $b = 2$ und $f(n) = n^4$.
  Damit gilt $n^{\log_b a} = n^3$ und $f(n) = n^4 = \Omega(n^{\log_b a+\varepsilon})$ für $\varepsilon = 1$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

3. Falls $f(n) = \Omega(n^{\log_b a+\varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n gilt, so gilt $T(n) = \Theta(f(n))$.

Beispiele

- $T(n) = kT(n/k) + n\log_2 n$ für ein $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n\log_2 n$.
  Zwar wächst $f(n) = n\log_2 n$ schneller als $n^{\log_b a} = n$, es kann jedoch trotzdem nicht der dritte Fall angewendet werden.
- $T(n) = 8T(n/2) + \Theta(n^4)$
  Es gilt $a = 8$ und $b = 2$ und $f(n) = n^4$.
  Damit gilt $n^{\log_b a} = n^3$ und $f(n) = n^4 = \Omega(n^{\log_b a+\varepsilon})$ für $\varepsilon = 1$.
  Außerdem gilt $8f(n/2) = n^4/2 = f(n)/2$.

2.1.3 Lösen von Rekursionsgleichungen

Sei $T\colon \mathbb{N} \to \mathbb{R}$ mit $T(1) = \Theta(1)$ und $T(n) = aT(n/b) + f(n)$ für $n \geq 2$.

3. Falls $f(n) = \Omega(n^{\log_b a+\varepsilon})$ für eine Konstante $\varepsilon > 0$ und $af(n/b) \leq cf(n)$ für eine Konstante $c < 1$ und alle hinreichend großen n gilt, so gilt $T(n) = \Theta(f(n))$.

Beispiele

- $T(n) = kT(n/k) + n\log_2 n$ für ein $k \in \mathbb{N}$ mit $k \geq 2$
  Es gilt $a = b = k$ und $f(n) = n\log_2 n$.
  Zwar wächst $f(n) = n\log_2 n$ schneller als $n^{\log_b a} = n$, es kann jedoch trotzdem nicht der dritte Fall angewendet werden.
- $T(n) = 8T(n/2) + \Theta(n^4)$
  Es gilt $a = 8$ und $b = 2$ und $f(n) = n^4$.
  Damit gilt $n^{\log_b a} = n^3$ und $f(n) = n^4 = \Omega(n^{\log_b a+\varepsilon})$ für $\varepsilon = 1$.
  Außerdem gilt $8f(n/2) = n^4/2 = f(n)/2$.
  $\Rightarrow T(n) = \Theta(f(n)) = \Theta(n^4)$.