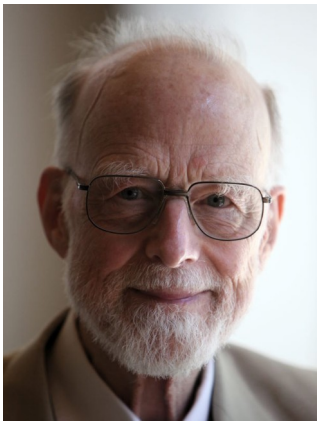
Neben den hübschen rekursiven Grafiken lässt sich Rekursion auch einsetzen, um einige typische Probleme der Informatik zu lösen. Rekursion findet dabei Anwendung in sogenannten

Divide and Conquer – Algorithmen

„Divide and conquer“[1] bedeutet in der Informatik, eine Aufgabe in kleinere Teilaufgaben zu zerlegen und diese zuerst zu lösen, bevor die Gesamtaufgabe gelöst wird. Die Teilaufgaben kann man rekursiv wieder in kleinere Teilaufgaben zerlegen, bis die Teilaufgaben so klein sind, dass sie ganz leicht gelöst werden können.

Am Beispiel wird klarer, wie das in der Praxis funktioniert. Wir werden zwei der bekannten rekursiven Sortieralgorithmen betrachten: den Merge Sort, bereits 1945 von John von Neumann entwickelt, und den Quicksort, entwickelt 1960 von Tony Hoare.

John von Neumann (1903 – 1957)



Sir Tony Hoare (geb. 1934)

1 „Divide and conquer“ wird im Deutschen mit „Teile und herrsche“ übersetzt.
To conquer bedeutet „erobern“ oder „besiegen“, was eher dazu passt, ein Problem zu lösen als „herrschen“.
Die deutsche Übersetzung bezieht sich wohl eher auf das lateinische „divide et impera“.

Merge Sort – Idee

1. Wir gehen aus von einer unsortierten Liste von Zahlen, die als Array implementiert ist:

10	5	8	3	1	7	4	9

2. Der Merge Sort teilt das Array genau in der Mitte in zwei Hälften.
   Dann ruft er sich rekursiv auf, einmal mit der linken, dann mit der rechten Hälfte.
   (Es sei denn, das Array ist schon so weit geteilt, dass es nur noch ein Element enthält.)

10	5	8	3

1	7	4	9

3. Nach diesen beiden rekursiven Aufrufen kann man davon ausgehen, dass sowohl der linke wie der rechte Teil *in sich* sortiert sind – das heißt, die Elemente der linken Hälfte sind in der richtigen Reihenfolge, und die Elemente der rechten Hälfte auch.

3	5	8	10

1	4	7	9

4. Jetzt können die beiden Teil-Arrays effizient zu einem gesamten Array zusammengeführt werden, das dann insgesamt sortiert ist. Dazu muss man allerdings eine der beiden Hälften in einem temporären Array zwischenspeichern.

1	3	4	5	7	8	9	10

Aufgabe 1

Führe den Merge Sort für die hier abgebildete Liste nach Schritt 2. weiter, indem du die Teil-Arrays weiter aufteilst und nachher wieder zusammensetzt.
Am Ende solltest du beim fertig sortierten Array (Schritt 3. und 4.) auskommen.

Merge Sort – Implementierung

```
/*
 * Rekursives Sortierverfahren durch "Verschmelzen"
 * left / right: Indizes, die den zu sortierenden Array-Teil begrenzen
 * Zum Start der Rekursion also left = 0 und right = length-1
 */
void mergeSort(double[] array, int left, int right)
{
   // Falls die aktuelle Teil-Liste die Länge 1 hat, breche ab.
   if (left == right) return;

   // Rekursiver Aufruf für linke und rechte Hälfte
   int middle = (left+right)/2;
   mergeSort(array, left,     middle);
   mergeSort(array, middle+1, right);

   // Linke und rechte Hälfte sind jetzt in sich sortiert.
   // Führe sie zu einer sortierten Liste zusammen.
   mergeArray(array, left, right);
}

/*
 * Hilfsmethode, die zwei Teile eines Arrays sortiert zusammenführt
 * Die beiden Teil-Arrays müssen in sich sortiert sein.
 * left / right: Indizes, die den zu sortierenden Array-Teil begrenzen
 */
void mergeArray(double[] array, int left, int right)
{
   // Kopiere die linke Teil-Liste in ein temporäres Array
   int middle = (left+right)/2;
   double[] temp = copyArray(array, left, middle);

   int i = 0;          // Index für die linke Teil-Liste
   int j = middle+1;   // Index für die rechte Teil-Liste
   int k;              // Index für die Gesamt-Liste

   // Führe linke und rechte Teil-Liste sortiert zusammen
   for (k = left; k <= right; k++)
   {
      // falls die linke Hälfte fertig ist, ist der Rest auch fertig
      if (i >= temp.length) break;

      // falls die rechte Hälfte fertig ist, wird noch die linke kopiert
      // falls beide noch nicht fertig sind, wird verglichen
      if (j > right || temp[i] < array[j])
      {
         array[k] = temp[i];
         i++;
      }
      else
      {
         array[k] = array[j];
         j++;
      }
   }
}
```

Integer-Division schneidet Nachkommastellen einfach ab

|| bedeutet „oder“

Aufgabe 2

Um den Merge Sort zu verstehen ist es hilfreich, zunächst die Methode mergeArray() anhand eines Beispiels nachzuvollziehen.
Ein Array wurde bereits soweit vorsortiert, dass linke und rechte Hälfte in sich sortiert sind:

array

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
2	7	11	19	5	18	21	28

Analysiere den Ablauf des Methodenaufrufs **`mergeArray(array,0,7)`** anhand des Quellcodes auf Seite 3:

1. Gib die Werte der Parameter left und right sowie den Wert der Variable middle an, außerdem welche Werte im Array „temp“ abgelegt werden.

2. Protokolliere, welche Werte i, j und k im Verlauf der for-Schleife annehmen, und welche Werte dem Array dabei schrittweise zugewiesen werden.

left	middle	right

temp

0	*1*	*2*	*3*

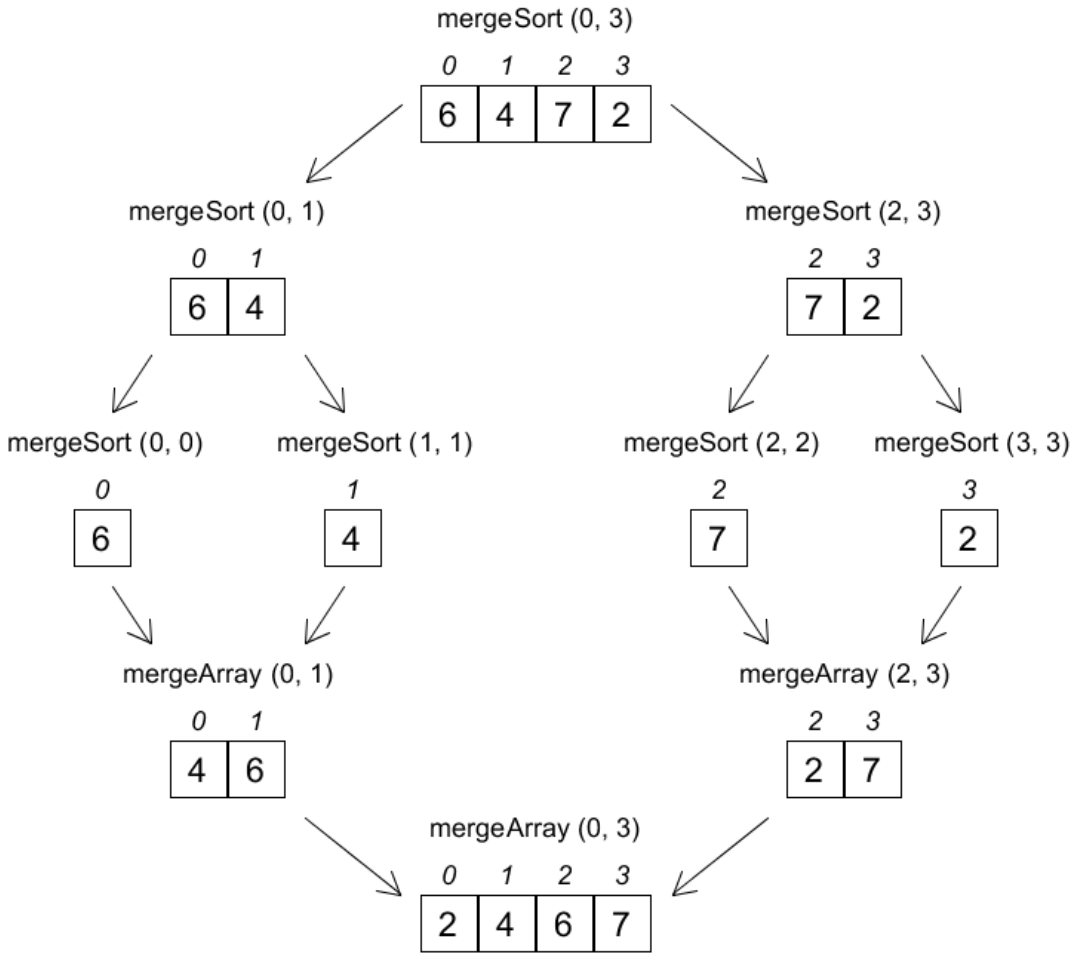
array

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*

i	j	k

Aufgabe 3

Jetzt untersuche die rekursive Methode **mergeSort()**. Die rekursiven Aufrufe bilden eine baumartige Struktur. Die folgende Abbildung zeigt diese Struktur für ein Array mit vier Elementen:



Zeichne die Struktur der Aufrufe von mergeSort (0, 8) für das folgende Array mit neun Elementen. Analysiere dazu den Quellcode auf Seite 2, insbesondere wie ein Array mit einer ungeraden Länge geteilt wird.

0	1	2	3	4	5	6	7	8
14	11	16	16	1	8	23	6	2