Divide&Conquer-Algorithmen lassen sich gut als rekursive Algorithmen darstellen. Das Prinzip eines rekursiven Algorithmus beruht darauf, ein schwieriges Problem in ein oder mehrere einfachere Probleme aufzuteilen. Man löst dann die einfacheren Probleme und kombiniert daraus die Lösung für das schwierige Originalproblem.

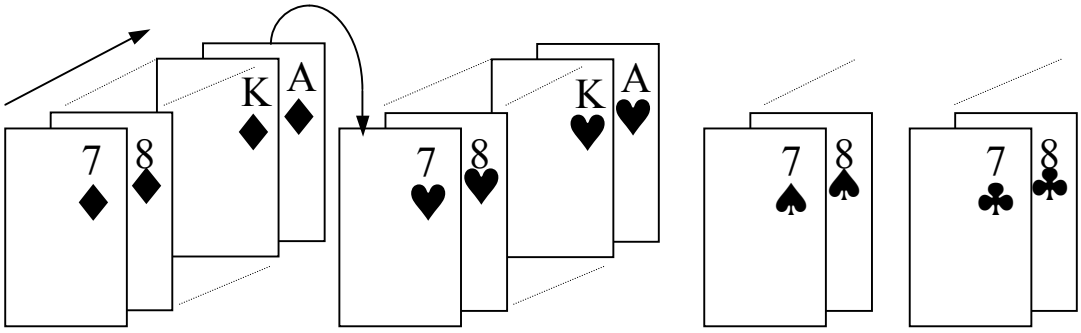
Problem Sortiere eine Folge von Objekten nach dem Divide&Conquer-Algorithmus ***MERGESORT***.

Aufgabe 1 (Experiment)

Ziel: Sortierung eines Kartenspiels

Sortierfolge nach Werten: 7 8 9 10 Bube Dame König As
nach Farben: Karo (♦) Herz (♥) Piek (♠) Kreuz (♣)

Die niedrigste Karte ist also die *Karo 7*, die höchste das *Kreuz As*.
Also: zum Beispiel liegt *Karo As* bei aufsteigender Sortierung vor *Herz 7*.



Regeln für das Experiment

1. **Teilen.** Erhalten Sie **einen Stapel**, dann ***teilen*** Sie diesen **in zwei gleichgroße Stapel** und

   Herrschen. ... geben Sie jeden Stapel an eine andere Person weiter, damit er diesen sortiert.

2. **Mischen.** Erhalten Sie **zwei Stapel** zurück, so ***mischen*** Sie diese wie folgt **zu einem neuen Stapel**: legen Sie die beiden Stapel offen vor sich hin, und entnehmen Sie entsprechend der obigen Sortierfolge jeweils von einem der beiden Stapel die **kleinste Karte** und stecken diese hinter den neuen Stapel. Sind beide Stapel leer, dann geben Sie den gemischten Stapel an denjenigen zurück, von dem Sie vorher einen Stapel bekommen haben.

3. **Ende.** Erhalten Sie einen Stapel mit **zwei Karten**, so sortieren Sie diesen und geben ihn zurück.

Aufgabe 2 (MERGESORT)

Formulieren Sie MERGESORT(A, p, r) rekursiv als Divide&Conquer-Algorithmus. Unterscheiden Sie in Ihrem Algorithmus deutlich die drei Schritte *Divide*, *Conquer* und *Combine*.

Hinweise: A = (A[1], A[2], ..., A[n]) ist die zu sortierende Folge, p ist der Index der ersten Elements und r der Index des letzten Elements der Folge A.

Gegeben ist eine Prozedur MERGE(A, p, q, r), die zwei Folgen (A[p], ..., A[q]) und (A[q+1], ..., A[r]) zu einer sortierten Folge (A[p], ..., A[r]) mischt.

Schreiben Sie MERGESORT:

Aufgabe 3 (Analyse von MERGESORT)

a) Identifizieren Sie in Ihrem Programm genau die Schritte *Divide*, *Conquer* und *Combine* und überlegen Sie sich für jeden dieser Schritte die Zeitkomplexität.

b) Stellen Sie mit den Überlegungen aus a) eine Differenzgleichung für MERGESORT auf.

c) Versuchen Sie die Differenzgleichung mit Hilfe des in der Vorlesung besprochenen Mastertheorems zu lösen.

Aufgabe 4

a) Gegeben ist ein Algorithmus, der sich selbst viermal rekursiv aufruft. In jedem der vier rekursiven Aufrufe hat das Datenfeld, das als Parameter übergeben wird, die Größe n/2; das Datenfeld wird also jeweils halbiert. Für die Zusammenführung der vier rekursiven Lösungen wird $O(n^2)$ Zeit benötigt.

1. Stellen Sie die **Differenzgleichung** des Mastertheorems T(n) für den oben beschriebenen Algorithmus auf und **erklären** Sie die Gleichung.
2. Welche asymptotische Laufzeit **T(n)** hat der Algorithmus?

Hinweis zum Mastertheorem:

$$T(n) = \begin{cases} O(n^{log_b a}), & \text{für } a > b^k \\ O(n^k \cdot \log n), & \text{für } a = b^k \\ O(n^k), & \text{für } a < b^k \end{cases}$$

b) Gegeben sind zwei Algorithmen A1 und A2 mit den Laufzeiten

$T_{A1}(n) = b \cdot n^c$ und

$T_{A2}(n) = b \cdot c^n$,

mit den Konstanten b und c.

Erklären und begründen Sie, welcher der beiden Algorithmen sich mit Hilfe des Power-Tests experimentell analysieren lässt.

Lösung Aufgabe 2 (MERGESORT)

Divide: Originalproblem → Teilprobleme

Conquer: Rekursive Lösungen der Teilprobleme. Sind die Teilprobleme klein genug, so ist ihre Lösung trivial.

Combine: Zusammenführen der Teillösungen zu der Lösung des Originalproblems.

bei MERGESORT:

Divide: Teile die zu sortierende Folge von n Elementen in zwei Teilfolgen mit jeweils n/2 Elementen. Für $n = b^m$ ist n/2 immer eine ganze Zahl.

Conquer: Sortiere die beiden Teilfolgen der Größe n/2 rekursiv mit MERGSORT.

Combine: Mische die zwei sortierten Folgen zu der gesuchten Originialfolge.

Haben die beiden Teilfolgen die Größe 1, so endet die Rekursion, denn jede Teilfolge der Größe 1 ist bereits sortiert.

Die zentrale Operation bei MERGESORT ist das ***Mischen*** zweier Teilfolgen. Wichtig ist, dass dies in einem Schritt, d.h. ohne Backtracking erfolgen kann. M.a.W: man betrachtet jedes Element der beiden Teilfolgen genau einmal.

Seien A[p...q] und A[q+1 ...r] die zu mischenden Teilfolgen mit $p \le q < r$. Dann kombiniert

Merge(A, p, q, r)

die Teilfolgen zu

A[p ... r]

Das Programm

MERGESORT (A, p, r)

sortiert die Elemente in A. Ist $p \ge r$, so ist höchstens ein Element in A; A ist damit sortiert. Ist $p < r$, so muss ein Index q berechnet werden, der A in zwei Teilfolgen A[p ... q] und A[q+1 ... r] aufteilt: A[p ... q] enthält $\lceil n/2 \rceil$ und A[q+1 ... r] enthält $\lfloor n/2 \rfloor$ Elemente

```
MERGESORT ( A, p, r )
   if p < r  then
      begin
         q := ⌊(p+r)/2⌋ ;
         MERGESORT ( A, p, q ) ;
         MERGESORT ( A, q+1, r ) ;
         Merge ( A, p, q, r ) ;
      end ;
```

Lösung von Aufgabe 3 (Analyse von MERGESORT)

Wir überlegen uns, welche Zeitkomplexität die einzelnen Schritte *Divide*, *Conquer* und *Combine* haben.

Der Vorteil des Mastertheorems bzw. über von Differenzgleichungen besteht darin, dass man kombinatorisch abstrahieren kann. Bezogen auf MergeSort bedeutet das, dass man die Hypothese aufstellen darf, dass zwei sortierte Teilfolgen der Länge O(n/2) in O(n) gemischt werden können. Dass diese Teilfolgen wiederum rekursiv zerlegt und gemischt werden spielt bei der Betrachtung der Differenzgleichung keine Rolle. Die DGL ist ja gerade dazu da, kleinere Probleme als gelöst zu betrachten.

Würde man eine direkte analytische Lösung zu finden versuchen, wäre die kombinatorische Zeitanalyse komplexer.

Für MERGESORT erhalten wir:

Divide: Der Teilungsschritt besteht darin, die Mitte des Array zu berechnen. Dies erfordert konstante Zeit und ist damit unabhängig von n. Also D(n) = O(1)

Conquer: Wir lösen jeweils zwei kleinere Subprobleme der Größe n/2 rekursiv. Damit erhalten wir:

$$1 * T(n/2)$$

Combine: Um zwei Teilfolgen der Größe n/2 zu mischen brauchen wir n Vergleiche:

$$C(n) = O(n)$$

Wir erhalten also:

$$T(n) = O(1) + 2 * T(n/2) + O(n),\ n > 1$$

Um diese Differenzgleichung mit dem Mastertheorem zu lösen, müssen die die Konstanten a, b und k bestimmen, d.h.

$$T(n) = a * T(a/b) + n^k$$

$\rightarrow$ a = 2, b = 2, k = 1, d.h. a Teilprobleme der Größe n/2

$\Rightarrow$ $T(n) = 2 * T(n/2) + n$

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

Lösung Aufgabe 4

a) Gegeben ist ein Algorithmus, der sich selbst viermal rekursiv aufruft. In jedem der vier rekursiven Aufrufe hat das Datenfeld, das als Parameter übergeben wird, die Größe n/2; das Datenfeld wird also jeweils halbiert. Für die Zusammenführung der vier rekursiven Lösungen wird $O(n^2)$ Zeit benötigt.

1. Stellen Sie die **Differenzgleichung** des Mastertheorems T(n) für den oben beschriebenen Algorithmus auf und **erklären** Sie die Gleichung. (5)
2. Welche asymptotische Laufzeit **T(n)** hat der Algorithmus? (5)

Hinweis zum Mastertheorem:

$$T(n) = \begin{cases} O\left(n^{\log_b a}\right), & \text{für } a > b^k \\ O\left(n^k \cdot \log n\right), & \text{für } a = b^k \\ O\left(n^k\right), & \text{für } a < b^k \end{cases}$$

Lösung:

Die Differenzgleichung für das Mastertheorem lautet:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) \cdot c \cdot n^k$$

Wir setzen a = 4 und k = 2 und erhalten mit b=2: $T(n) = 4 \cdot T\left(\frac{n}{2}\right) \cdot c \cdot n^2$

Es folgt: $4 = 2^2$, also ist **$T(n) = O(n^2 * \log n)$**

b) Gegeben sind zwei Algorithmen A1 und A2 mit den Laufzeiten

$T_{A1}(n) = b \cdot n^c$ und

$T_{A2}(n) = b \cdot c^n$,

mit den Konstanten b und c.

Erklären und begründen Sie, welcher der beiden Algorithmen sich mit Hilfe des Power-Tests experimentell analysieren lässt. (10)

Lösung:

Mit dem Powertest können keine Algorithmen experimentell analysiert werden, die exponentielle Laufzeit haben, wei z.B. $T_{A2}(n) = b \cdot c^n$. Durch die Logarithmierung entsteht bei exponentiellen Algorithmen ein Polynom mind. 2. Grades, dessen Parameter nicht durch lineare Regression bestimmt werden können. Nur Algorithmen mit polynomialer Laufzeit wie z.B. $T_{A1}(n) = b \cdot n^c$ lassen sich mit dem Powertest analysieren.