Hashtabelle mit verketteten Listen

```
public class IntHashtable {
   private IntListNode[] tab = new IntListNode[100000];

   private int hc (int e) { return Math.abs(e % 100000); }
   public void add (int e) {
      tab[hc(e)] = new IntListNode (e, tab[hc(e)]);
   }
   public boolean contains (int e) {
      return tab[hc(e)] != null && tab[hc(e)].contains(e);
   }
   public void remove (int e) {
      if (tab[hc(e)] != null)
         tab[hc(e)] = tab[hc(e)].remove(e);
   }
}
```

```
public static void mergesort (int[] elems) {
   if (elems.length > 1) {
      int[] left = new int[elems.length / 2];
      for (int i=0; i < left.length; i++)
         left[i] = elems[i];
      mergesort(left);
      int[] right = new int[elems.length - left.length];
      for (int i=0; i < right.length; i++)
         right[i] = elems[left.length + i];
      mergesort(right);
      for (int i=0, l=0, r=0; i < elems.length; i++)
         if (r >= right.length ||
               (l < left.length && left[l] <= right[r]))
            elems[i] = left[l++];
         else
            elems[i] = right[r++];
   }
}
```

Teile und Herrsche

erfordert Kreativität:

- Lösungsansatz suchen

  (auf bewährte Techniken zurückgreifen)

- Voraussetzungen für Lösungsansatz erfüllen

- wenn nicht erfüllbar, dann zurück zum Anfang

```
public static void quicksort (int[] elems) {
    quicksort (elems, 0, elems.length - 1);
}

private static void quicksort (int[] elems, int l, int r) {
    int i=l, j=r, pivot=elems[(l+r)/2];
    while (i <= j) {
        while (elems[i] < pivot) i++;
        while (pivot < elems[j]) j--;
        if (i <= j) {
            int h = elems[i];
            elems[i++] = elems[j];
            elems[j--] = h;
        }
    }
    if (l < i-1) quicksort (elems, l, i-1);
    if (i < r) quicksort (elems, i, r);
}
```

Quicksort: Pragmatische Sichtweise

- zeitlicher Aufwand: durchschnittlich: $O(n \cdot \log(n))$
  maximal: $O(n^2)$
- kleine konstante Faktoren (meist schneller als Mergesort)
- maximaler Aufwand sehr unwahrscheinlich
- Auswahl des Pivot-Elements entscheidend
- in-place-Sortierverfahren

Automatisiertes Zahlenraten

```
public static int versuche (int max) {
    UnbekannteZahl zuErraten = new UnbekannteZahl(max);
    int i = 0;
    int j = max - 1;
    for (int anzahl = 1; ; anzahl++) {
        int k = (i + j) / 2;    // k = i + ((j - i) / 2)
        if (zuErraten.kleiner(k)) {
            j = k - 1;
        } else if (zuErraten.gleich(k)) {
            return anzahl;
        } else {
            i = k + 1;
        }
    }
}
```

Binäre Suche

```
public static int index (int x, int[] elems) {
    int i = 0;
    int j = elems.length - 1;
    while (i <= j) {
        int k = (i + j) / 2;
        if (x < elems[k]) {
            j = k - 1;
        } else if (x == elems[k]) {
            return k;
        } else {
            i = k + 1;
        }
    }
    return -1;
}
```

Generizität – Liste für beliebige Typen

```
public class GenList<A> {
    private ListNode<A> head = null;

    public void add (A elem) {
        head = new ListNode<A> (elem, head);
    }
    public boolean contains (A elem) {
        return head != null && head.contains(elem);
    }
    public void remove (A elem) {
        if (head != null)
            head = head.remove(elem);
    }
}
```

```
class ListNode<A> {
    private A elem;
    private ListNode<A> next;

    ListNode (A e, ListNode<A> n) { elem = e; next = n; }

    boolean contains (A e) {
        return e.equals(elem) ||
               (next!=null && next.contains(e));
    }
    ListNode<A> remove (A e) {
        if (e.equals(elem))
            return next;
        else if (next != null)
            next = next.remove(e);
        return this;
    }
}
```

Anwendungsbeispiel

```
public class Student {
    private int mnr;
    ...
    public boolean equals (Object that) {
       return this.getClass() == that.getClass()
              && mnr==((Student)that).mnr;
    }
}

    GenList<Student> list = new GenList<Student>();
    list.add (new Student());
```

Autoboxing

```
GenList<Integer> ilist = new GenList<Integer>();
ilist.add(1);
ilist.add(new Integer(678));

GenList<Boolean> blist = new GenList<Boolean>();
blist.add(true);
blist.add(new Boolean(false));

if (ilist.contains(678) {
    ...
}
```