Sortieren

Authors of slides:
Partly extracted from script of Prof. Kai-Uwe Sattler

Lernziele

- SelectionSort verstehen und implementieren können
- InsertionSort verstehen und implementieren können
- MergeSort verstehen und implementieren können
- QuickSort verstehen und implementieren können

SelectionSort: Konzept

- Wie beim Sortieren von Karten
- Das größte Element wird gesucht und am Ende eingefügt

Implementieren Sie Selection Sort in Java

```
public static int[] ssort(int[] toSort) {
  int marker = toSort.length - 1;
  while (marker > 0) {
    int indexOfMax = 0;
    for (int i = 0; i <= marker; i++) {
      if (toSort[i] > toSort[indexOfMax]) {
        indexOfMax = i;
      }
    }
    Utility.swap(toSort, marker, indexOfMax);
    marker--;
  }
  return toSort;
}
```

Insertion Sort

- Sortieren durch einfügen
- Es wird von einem Stapel Karten gezogen, und die Karte wird entsprechend einsortiert

Implementieren Sie Insertion Sort in Java

```
public static int[] iSort(int[] toSort) {
  for (int i = 1; i < toSort.length; i++) {
    int key = toSort[i];
    int j = i - 1;
    while (j >= 0 && toSort[j] > key) {
      toSort[j + 1] = toSort[j];
      j--;
    }
    toSort[j + 1] = key;
  }
  return toSort;
}
```

MergeSort

- Erfolgt in zwei Schritten:
  - Das Array wird so lange halbiert, bis man nur noch ein Element betrachtet
  - Die einzelnen Elemente werden wieder sortiert zusammen gefügt

Implementieren Sie Merge Sort in Java

```
static void msort(int[] array, int left, int right) {
  int i, j, k;
  int[] b = new int[array.length];

  if (right > left) {
    int mid = (right + left)/2;

    msort(array, left, mid);
    msort(array, mid + 1, right);

    for (k = left; k <= mid; k++)
      b[k] = array[k];
    for (k = mid; k < right; k++)
      b[right + mid - k] = array[k + 1];
    i = left;
    j  = right;
    for (k = left; k <= right; k++) {
      if (b[i] < b[j])
        array[k] = b[i++];
      else
        array[k] = b[j--];
    }
  }
}

static void mergeSort(int[] array) {
  msort(array, 0, array.length - 1);
}
```

- Hinweis: Das ist eine der erwähnten anderen Implementierungen des Mergesorts

QuickSort

- Ähnlich wie MergeSort basiert es auf dem Prinzip *Divide and Conquer*
- Aber ohne mischen, so dass es weniger Ressourcen verbraucht
- Bestimmung eines Pivot-Elements in der Mitte des Arrays
- Von links und rechts wird das Array durchlaufen, bis jeweils ein Element gefunden wurde, das größer bzw. kleiner als Pivot-Element ist
- Diese beiden werden dann getauscht
- Dann wird jeweils mit der linken und rechten Hälfte des Arrays das Verfahren wiederholt

QuickSort II

Implementieren Sie Quick Sort in Java

```
public static int[] quickSort(int[] array) {
  return qsort(array, 0, array.length - 1);
}

private static int[] qsort(int[] array, int left, int right) {
  int high = right;
  int low = left;

  if (low < high) {
    int pivot = array[(left + right)/2];

    while (low <= high) {
      while (low < right && array[low] < pivot)
        low++;

    while (high > left && array[high] > pivot)
        high--;
    }
    if (low <= high) {
      Utility.swap(array, low, high);
      low++;
      high--;
    }
  }

  if (left < high)
    qsort(array, left, high);
  if (low < right)
    qsort(array, low, right);

}
return array;
```

Aufwand

Verfahren	Komplexitätsklasse	Stabilität
SelectionSort	$O(n^2)$	instabil
InsertionSort	$O(n^2)$	stabil
MergeSort	O(n*log n)	stabil
QuickSort	O(n*log n)	instabil

Lernziele

- SelectionSort verstehen und implementieren können
- InsertionSort verstehen und implementieren können
- MergeSort verstehen und implementieren können
- QuickSort verstehen und implementieren können