Entwurf von Algorithmen: Sortierproblem

1. Strategie: Tue das Nächstliegende (greedy-Strategie)

Lösungsskizze: *Selectionsort*

- Durchlaufe die Positionen des neuen Datenfelds der Reihe nach.
- Suche das kleinste Element ab der laufenden Position im neuen Datenfeld.
- Vertausche dieses Element mit dem Element der laufenden Position.
- Gib am Ende des Durchlaufs das neue Datenfeld aus.

```
procedure selectionsort (data): array
begin
  pos := 1;
  while pos < length(data) do
  begin
    newPos := minPos (data, pos, length(data));
    aux := data[pos];
    data[pos] := data[newPos];
    data[newPos] := aux;
    pos := pos + 1;
  end; {while}
  return data;
end {selectionsort}
```

```
procedure sort (data): array
begin
  newData := copy (data);
  return selectionsort (newData);
end {sort}
```

Entwurf von Algorithmen: Sortierproblem

1. Strategie: Tue das Nächstliegende (greedy-Strategie)

Ausformulierung der Hilfsprozedur *minPos*:

```
procedure minPos (data, first, last): integer
begin
  resultPos := first;
  resultValue := data[resultPos];
  pos := first;
  while pos < last do
  begin
    pos := pos + 1;
    if data[pos] < resultValue
      then
        begin
          resultPos := pos;
          resultValue := data[resultPos];
        end;
  end; {while}
  return resultPos;
end {minPos}
```

Entwurf von Algorithmen: Sortierproblem

2. Strategie: Teile und herrsche (divide and conquer)

Lösungsskizze: *Mergesort*

- Teile das Datenfeld in 2 Hälften auf.
- Sortiere beide Hälften getrennt.
- Mische die beiden sortierten Hälften in ein zweites Datenfeld.

```
procedure mergesort
        (fromData, toData, left, right)
begin
  if left < right
  then
    begin
      mid := (left + right) div 2;
      mergesort (toData, fromData,
                              left, mid);
      mergesort (toData, fromData,
                          mid+1, right);
      merge (fromData, toData,
             left, mid, mid+1, right);
    end {if}
  end {mergesort}
```

Rekursive Darstellungsvariante

```
procedure sort (data): array
begin
  data1 := copy (data);
  data2 := copy (data);
  mergesort (data1,
     data2, 1, length(data));
  return data2
end {sort}
```

Entwurf von Algorithmen: Sortierproblem

2. Strategie: Teile und herrsche (divide and conquer)

Lösungsskizze: *Mergesort*

- Teile das Datenfeld in 2 Hälften auf.
- Sortiere beide Hälften getrennt.
- Mische die beiden sortierten Hälften in ein zweites Datenfeld.

```
procedure mergesortIter (data): array
begin
  data2 := copy (data); n := length(data);
  sortedLength := 1;
  while sortedLength < n do
  begin
    left1 := 1;
    while (left1+sortedLength) < n do
    begin
      right1 := left1+sortedLength; left2 := right1+1; right2 := left2+sortedLength;
      merge (data, data2, left1, right1, left2, right2);
      left1 := right2 + 1
    end;
    sortedLength := sortedLength + sortedLength;
    aux := data; data := data2; data2 := aux
  end;
  return data
end {sort2}
```

Iterative Darstellungsvariante

```
procedure sort (data): array
begin
  newData := copy (data);
  return mergesortIter(newData)
end {sort}
```

Entwurf von Algorithmen: Sortierproblem

2. Strategie: Teile und herrsche (divide and conquer)

Ausformulierung der Hilfsprozedur *merge*:

```
procedure merge (fromData, toData, left1,
                     right1, left2, right2)
begin
  pos1 := left1; pos2 := left2; pos := left1;
  while (pos ≤ right2) do
  begin
    if pos1 > right1
      then
        assign (fromData, toData, pos2, pos)
      else if pos2 > right2
        then
          assign (fromData, toData, pos1, pos)
      else if fromData[pos1] ≤ fromData[pos2]
        then
          assign (fromData, toData, pos1, pos)
        else
          assign (fromData, toData, pos2, pos);
    pos := pos + 1
  end {while}
end {merge}
```

```
procedure assign (fromData, toData,
                          fromPos, toPos)
begin
  toData[toPos] := fromData[fromPos];
  fromPos := fromPos + 1;
end {assign}
```

Bei assign muss der Parameter fromPos als call by reference deklariert werden !