Ursprungligen av Rolf Axelsson

Föreläsning 3

Sortering

MALMÖ HÖGSKOLA

Sortering

- Bubblesort
- Insertionsort
- Mergesort
- Quicksort
- Sortering av objekt – Comparable
- Sortering av objekt – Comparator

Varför sortera

- Skapa ordning bland lagrade objekt – förenklar problem (t.ex. att presentera kontoutdrag i datumordning)
- Effektivisera lösningar på vissa problem – t.ex. binary search

Bubblesort

Problem: Sortera en array

Algoritm:
Element jämförs parvis (arr[j] , arr[j+1]) från slutet till början av arrayen.
Om arr[j] > arr[j + 1] så skiftar elementen plats.
Den första iterationen flyttar det minsta elementet till position 0.
Nästa iteration flyttar det näst minsta elementet till position 1.
osv tills endast ett element är kvar (det största i position (arr.length - 1))

	Iteration 1				Iter 1	Iter 2	Iter 3	Iter 4
49	49	49	49	**29**	**29**	**29**	**29**	**29**
92	92	92	**29**	**49**	49	**36**	**36**	**36**
29	29	**29**	**92**	92	92	49	**49**	**49**
51	**36**	**36**	36	36	36	92	51	**51**
36	**51**	51	51	51	51	51	92	92

MALMÖ HÖGSKOLA

Bubblesort

Implementering

```
public static void bublesort(int[] array) {
    for( int i=0; i < array.length - 1; i++ ) {
        for( int j = array.length - 1; j > i; j-- ) {
            if( array[ j ] < array[ j - 1 ]) {
                Utility.swap( array, j, j - 1 );
            }
        }
    }
}
```

Kommentarer:

- Mycket ineffektiv sortering, undvikes alltid
- Enkel algoritm
- Behöver inget extra minnesutrymme
- Stabil – passar vid sortering av objekt

MALMÖ HÖGSKOLA

Insertionsort

Problem: Sortera en array

Algoritm:
Listan sorteras från början och successivt mot slutet.

Först ordnas de två första elementen

Sedan ordnas de tre första elementen. 3:e elementet får bubbla uppåt till rätt plats.

Sedan ordnas de fyra första elementen. 4:e elementet får bubbla uppåt till rätt plats.

Osv tills hela listan är ordnad

	Iter 1	Iter 2	Iter 3	Iter 4
49	49	29	29	29
92	92	49	49	36
29	29	92	51	49
51	51	51	92	51
36	36	36	36	92

MALMÖ HÖGSKOLA

Insertionsort

Implementering (ungefär suns version)

```
// Används på små arrayer i java (mindre än 7 element)
public static void insertionsort( int[] array ) {
    for( int i = 1; i < array.length; i++ ) {
        for ( int j = i; ( j > 0 ) && ( array[ j - 1 ] > array[ j ] ) ; j-- ) {
            Utility.swap( array, j, j - 1 );
        }
    }
}
```

Kommentarer:

- Mycket ineffektiv sortering. Effektivare sortering än Bubblesort
- Användbar vid små arrayer
- Enkel algoritm
- Behöver inget extra minnesutrymme
- Stabil – passar vid sortering av objekt

MALMÖ HÖGSKOLA

Mergesort

Problem: Sortera en array

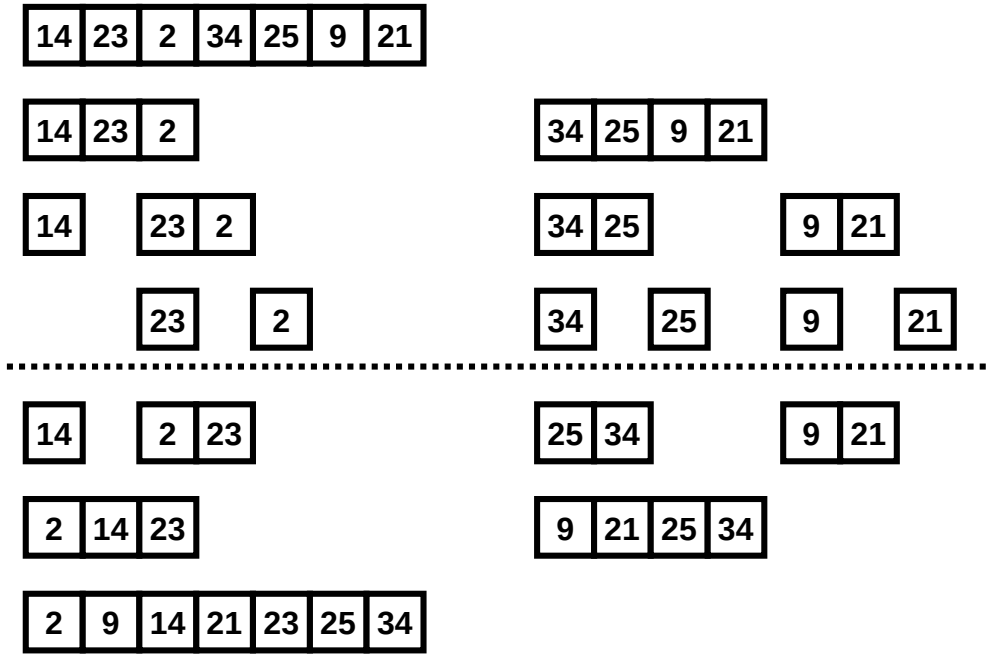
Algoritm:

Divide and conquer, ”söndra och härska”. Dela upp problem i delproblem. Pågår tills problemen blivit så små så de är lösbara.

Om arrayen innehåller mer än ett element så

- Dela upp arrayen i två delar och sortera vardera delen med mergesort
- Sortera samman delarna

Exempel



MALMÖ HÖGSKOLA

Mergesort

Implementering

```
public static void mergesort( int [] array ) {
    mergesort( array, 0, array.length );
}

private static void mergesort( int[] array, int first, int n ) {
    int n1,n2;
    if( n > 1 ) {
        n1 = n / 2;
        n2 = n - n1;
        mergesort( array, first, n1 );
        mergesort( array, first + n1, n2);
        merge( array, first, n1, n2 );
    }
}

private static void merge( int[] array, int first, int n1, int n2 ) {…}
```

Kommentarer:

- Effektiv sortering.
- ”Enkel” algoritm
- Behöver extra minnesutrymme
- Stabil – passar vid sortering av objekt

MALMÖ HÖGSKOLA

Quicksort

Problem: Sortera en array

Algoritm:

Divide and conquer, ”söndra och härska”. Dela upp problem i delproblem. Pågår tills problemen blivit så små så de är lösbara.

Om arrayen innehåller mer än ett element så

- **Välj ett element i arrayen**
- Dela upp arrayen i två delar så att de element som är mindre än det valda elementet är till vänster om elementet och de som är större är till höger om elementet. **Nu är det valda elementet i rätt position.**
- Sortera elementen till vänster om det valda elementet på samma sätt
- Sortera elementen till höger om det valda elementet på samma sätt

MALMÖ HÖGSKOLA

Quicksort

Implementering

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

private static void quicksort( int[] array, int left, int right ) {
    int pivotIndex;
    if( left < right ) { // minst två element
        pivotIndex = partition(array, left, right, (left+right)/2); // flytta elementen
        quicksort( array, left, pivotIndex-1);
        quicksort( array, pivotIndex+1, right);
    }
}

private static void partition( int[] array, int left, int right, int pivotIndex) {…}
```

Kommentarer:

- Effektiv sortering.
- ”Enkel” algoritm
- Behöver inget extra utrymme
- Ej stabil – passar ej vid sortering av objekt

MALMÖ HÖGSKOLA

Comparable

Problem: Sortera en array med objekt

Ofta är det **arrayer med objekt** som man sorterar. För att sortera arrayer måste objekten kunna jämföras.
Klasser som implementerar **Comparable** medger sortering

Exempel

```
public class Commodity implements Comparable {      // Comparable<Commodity>
    private String name;
    private double price;


    :

    public int compareTo( Object obj ) {            // compareTo(Commodity c)
        Commodity c = ( Commodity )obj;             //
        if( this.price < c.price )
            return -1;
        else if( this.price == c.price )
            return 0;
        else
            return 1;
    }
}
```

MALMÖ HÖGSKOLA

Comparable – jämföra två Commodity-objekt

```
Commodity c1 = new Commodity("D", 12.25);
Commodity c2 = new Commodity("B", 8.90);

Comparable comp = ( Comparable )c1;
int res = comp.compareTo( c2 );
if( res == -1 )
   System.out.println( c1 + "\n" + c2);
else if( res == 0 )
   System.out.println("Lika stora");
else
   System.out.println( c2 + "\n" + c1);
```

Resultat

```
B: 8.9
D: 12.25
```

MALMÖ HÖGSKOLA

Comparable – i insertionsort

I ***insertionsort*** ser jämförelsen mellan två ***int-element*** ut så här:

```
public static void insertionsort(int[] array) {
   for( int i = 1; i < array.length; i++ ) {
      for ( int j = i; ( j > 0 ) && ( array[j] < array[j-1] ) ; j--) {
         Utility.swap( array, j, j-1 );
      }
   }
}
```

I ***insertionsort*** ser jämförelsen mellan ***två objektreferenser*** ut så här:

```
public static void insertionsort(Object[] array) {
   Comparable comp;
   for( int i = 1; i < array.length; i++ ) {
      comp = (Comparable)array[i];
      for ( int j = i; ( j > 0 ) && ( comp.compareTo( array[j-1] ) < 0 ) ; j--) {
         Utility.swap( array, j, j-1 );
      }
   }
}
```

eller kortare:

```
public static void insertionsort(Object[] array) {
   for( int i = 1; i < array.length; i++ ) {
      for ( int j = i; ( j > 0 ) && ( ( (Comparable)array[j] ).compareTo( array[j-1] ) < 0 ) ; j--) {
         Utility.swap( array, j, j-1 );
      }
   }
}
```

MALMÖ HÖGSKOLA

Comparator – sortera objekt

En klass som implementerar interfacet Comparator kan hjälpa till vid sortering. Detta är aktuellt om:

Den naturliga sorteringsordningen (Comparable) inte ger den sortering som man önskar.

Objekten som ska sorteras implementerar inte Comparable.

Att sortera Commodity-objekt avtagande med avseende på priset:

```
import java.util.Comparator;

public class PriceDescending implements Comparator { // ... implements Comparator<Commodity> {
   public int compare( Object obj1, Object obj2 ) {          // ... compare(Commodity c1, commodity c2) {
      Commodity c1 = ( Commodity )obj1;
      Commodity c2 = ( Commodity )obj2;
      double res = c1.getPrice() - c2.getPrice();
      if( res < 0 )
         return 1;
      else if( res == 0 )
         return 0;
      else
         return -1;
   }
}
```

MALMÖ HÖGSKOLA

Comparator – i bubblesort

Bubblesort för int-array

```
public static void bubblesort(int[] array) {
    for( int i=0; i < array.length - 1; i++ ) {
        for( int j = array.length - 1; j > i; j-- ) {
            if( array[ j ] < array[ j - 1 ] )
                Utility.swap( array, j, j - 1 );
        }
    }
}
```

Bubblesort för Object-array med hjälp av comparator-implementering

```
public static void bubblesort(Object[] array, Comparator comp) {
    for( int i=0; i < array.length - 1; i++ ) {
        for( int j = array.length - 1; j > i; j-- ) {
            if( comp.compare( array[ j ], array[ j - 1 ] ) < 0 )
                Utility.swap( array, j, j - 1 );
        }
    }
}
```

MALMÖ HÖGSKOLA

Frågor?