

Opgaver til forelæsningen 22/10-2002

Opgave 1 og 2 er afleveringsopgave for Datalogi C. Afleveringsfristen er 1. oktober men aftal evt. en senere frist med Lars, da opgaven er annonceret lidt sent.

Opgave 1

Afsnit 17.2 i bogen beskriver en class `LinkedList`. Udvide klassen med `toString`- og `equals`-metoder; `equals` skal svare til at to lister indeholder de samme (`equals`) elementer i den samme rækkefølge. Udvid yderligere klassen med en `compareTo`-metode, som er baseret på følgende principper, som kaldes en *leksikalsk* ordning.

- hvis to lister indeholder de samme elementer i samme rækkefølge (som `equals` ovenfor) betragtes de som værende ens.
- hvis `liste1`'s første element kommer før `liste2`'s første element, så siger vi, at `liste1` kommer før `liste2`.
- hvis `liste1` og `liste2` indeholder de samme elementer i samme rækkefølge op til og med position `n`, så gælder følgende:
 - hvis `liste1` ikke har noget `n+1`'te element, så kommer `liste1` før `liste2`
 - ellers, hvis `liste1`'s `(n+1)`'te element kommer før `liste2`'s `(n+1)`'te, så kommer `liste1` før `liste2`.

Grunden til, vi kalder denne ordning leksikalsk, er at den svarer til sortering af nøgleordene i en ordbog eller et leksikon — hvis altså hvert ord betragtes som en liste af bogstaver. F.eks. “a” kommer før “ab”, som kommer før “abc”, som kommer før “abe”. Implementér og aftest løsningen.

Opgave 2

I bogens afsnit 17.4 beskrives en klasse `SortedLinkedList`. Skriv en metode
`public SortedLinkedList merge(SortedLinkedList L1, SortedLinkedList L2)`
som tager to sortererede lister `L1` og `L2` og producerer en ny sorteret liste bestående af elementerne fra `L1` og `L2`. Implementér og aftest løsningen.

Opgave 3 (Læs og forstå opgaveformuleringen; det at løse den er kun for særligt interesserede)

Benyt den version af quicksort som blev brugt ved forelæsningen 8. oktober (se kurssets `www`-side). Opgaven handler om at håndoptimere koden ved at erstatte rekursion med en stak implementeret ved et array — og hvor man ikke skal benytte nogen af Javas medfølgende klasser men skrive stakmanipulationen direkte i koden (så som at lægge ting ind i arrayet og tælle staktoppen op og ned). Så i stedet for at have en rekursiv metode

```
quicksort( Comparable [ ] a, int low, int high )
```

placeres al koden inde i en ikke rekursiv

```
quicksortNonRecursive( Comparable [ ] a )
```

som (i optimeringens øjemed) også indeholder koden fra “partition”.

For at undgå at pakke ting ind i objekter, benytter vi to stakke, en for “low” og en for high”:

```
int [] lowStack = new int [100]; int [] highStack = new int [100];  
stackTop = 0; lowStack[0]=0; highStack[0]= a.length - 1;
```

Stakken initialiseres altså så den indeholder data svarende til det første kald af den oprindelige rekursive metode.

Strategien er nu at quicksortNonRecursive benytter en løkke, der indeholder følgende:

1. Der tages et par af low og high af stakkene og der foretages “partition”
(med mindre forskellen mellem de to er under tærsklen, så vi beder insertionSort om at gøre arbejdet; indsæt også kopi af indmaden fra insertionSort i quicksortNonRecursive for at være sikker på at spare metodekald)
2. Der hvor den oprindelige quicksort siger
quicksort(a, low, newPosForOldMiddleElt - 1);
quicksort(a, newPosForOldMiddleElt + 1, high);
smider vi i stedet de to gange argument-par på stakkene.

Benyt de tidsmålingsværktøjer som blev beskrevet ved en tidligere opgave til at afgøre hvor meget hurtigere (om noget?) den ikke-rekursive er sammenlignet med den oprindelige rekursive.