

Opgaver til forelæsningen 8/10-2002

Opgave 1 og 2 har højest prioritet; opgave 3 til efterfølgende hjemmearbejde.

Afleveringsopgave til DatC: Opgave 1 og 2 (incl. 1.1, 1.2, 2.1 og 2.2) med afleveringsfrist **fredag 18. oktober**.

Opgave 1

Definér en klasse, som svarer til spillekort, hvor hvert objekt har to attributter, en *farve*, som er klør, hjerter, spar, eller ruder, og en *værdi* som er et tal 1..13. Vi skal bruge klassen til at eksperimentere med sortering, så den skal altså være en Comparable klasse, dvs. som skal udstyres med en compareTo metode.

I første omgang skal compareTo udtrykke følgende ordning:

- Klør er mindre end hjerter, som er mindre end spar, som er mindre end ruder.
- Hvis to kort har samme farve, så bestemmes rækkefølge af værdien, dvs. 1 (Es) er mindre end 2, som, som er mindre end 12, som er mindre end 13 (Konge).

Spørgsmål 1.1

Skriv en sådan klasse for spillekort; skriv et testprogram med et array af spillekort, fyld nogle spillekort i, og afprøv den quicksort-version som blev fremvist ved forelæsningen (filnavn QSort.java; kan hentes via kurssets hjemmeside).

Spørgsmål 1.2

Nu har man i forskellige spil kort forskellige opfattelser af, hvilke spillekort som vejer tungere end andre. F.eks. er det ret almindeligt, at Es (svarende til værdi=1) er et vigtigere kort end Konge (værdi = 13). For at kunne modellere dette er vi nødt til at have mere end en måde at sammenligne kort på, end den "standardordning" som er indbygget i løsningen på spg. 1.1. For at råde bod på det må vi oprette en "Comparator". Definér en sådan som udtrykker at kortværdi 2 er mindre end 3, som er mindre end som er mindre end 12, som er mindre end 13, som er mindre end 1. Og nu vil vi gerne gentage spørgsmål 1.1 med den Comparator. Lav en ny version af quicksortmetoden, som tager et ekstra argument som er en Comparator, og arrayet, den får, behøver nu ikke mere bestå af Comparables men blot Object'er. Nu kan du afteste metoden.

Opgave 2

Vi skal nu eksperimentere med at sortere heltal — vel og mærket ved array af typen "int []" og ikke de kluntede "Integer []". Som det blev diskuteret ved forelæsningen kan den generelle quicksort kun arbejde på arrays af Object'er men ikke arrays af heltal.

Spørgsmål 2.1

Lav en ny version af quicksortmetoden som blev fremvist ved forelæsningen (filnavn QSort.java; kan hentes via kurssets hjemmeside), så den sorterer på "int []", som

sammenlignes med de almindelige sammenligningsoperatorer. Implementér og aftest på et lille eksempel.

Spørgsmål 2.2

Nu skal vi til at lave tidsmålinger! Metoden `java.lang.System.currentTimeMillis()` fortæller dig hvor lang tid du har brugt siden programmet startede, så hvis du skal finde ud af, hvor lang tid en bestemt beregning tager kan du gøre sådan her:

```
System.out.println("Calculating...");
long start = System.currentTimeMillis();
BEREGNING;
System.out.println("[ "+(System.currentTimeMillis()-start)+" ]");
```

I `java.util.Random` finder du værktøj til at generere tilfældige tal som testdata.

Skriv et eller flere testprogrammer som gør det muligt at sammenligne faktiske sorteringstider for følgende:

- Sortering af et “`Integer []`” med den version af Quicksort som arbejder på “`Comparable []`”
- Sortering af et “`int []`” med den version du konstruerede i spørgsmål 2.1.

Afprøv dem begge på de samme arraylængder og giv et estimat på hvor meget hurtigere den sidste metode er. Prøv med forskellige længder array, op til hvad din computer kan klare uden at gå i kløjs. Opfører algoritmerne sig som $n \log n$?

Obs: hvis tidsforbruget lige pludselig ikke ser ud til at følge den $n \log n$ kurve vi har fra teorien, når man sætter n op, men noget der ligner eksponentielt, kan det meget vel skyldes at Javas garbagecollector er vågnet til dåd.

Opgave 3.

I den quicksortversion, som er benyttet overfor, bruges en værdi for `CUTOFF = 10`. Ukritisk kopieret efter bogen. Foretag eksperimenter i stil med dem i opgave 2 for at få bestemt optimale `CUTOFF` værdier for hver af de to versioner, dvs. med “`Comparable []`” og med “`int []`”. Hvis der er stor forskel på de to optimale `CUTOFF`, forsøg at forklare hvorfor.