INF102 Algoritmar, datastrukturar og programmering - Innlevering 1 hausten 2013

Dette er ei obligatorisk innlevering som må vera godkjend for å gå opp til eksamen. Resultatet tel 10% av sluttkarakteren. Innleveringa skal vera individuell. Vi godtar berre innlevering via MiSide si innleveringsmappe, og alle filene skal pakkast til ei zip-fil. Alle filer som ikkje er Java-kode skal vera på pdf-format. For å få full utteljing må algoritmar og program vera lettleselege.

I alle implementasjonsoppgåvene kan du fritt gjera bruk av Java-kode som kan lastast ned frå læreboka si heimeside.

Oppgåve 1

Kva er køyretida til desse kodefragmenta? Uttrykk svara med ein Θ-relasjon.

a)
```
for (int i=0; i<n; i++) {
  for (int k=0; k<p; k++) {
    int x=0;
    for (int j=0; j<m; j++) {
      x += A[i][j] * B[j][k];
    }
    C[i][k] = x;
  }
}
```

b)
```
int p = -1;
int q = n;
while (p+1 < q) {
  int m = (p+q)/2;
  if (A[m] < x) { p=m; }
  else { q=m; }
}
```

c)
```
i = n;
while(i > 1) {
  j = 1;
  while(j < i) {
    j = j + 1;
  }
  i = i / 2;
}
```

Oppgåve 2

Finn ein $\sim$-relasjon til kvar av desse funksjonane:

a) $20n^4 + 123n^2 + 1000 \lg n$

b) $120n^3 + 5n^2 \lg n$

c) $100n^4 + 2^n$

d) $n + n^2 \lg n + \lg n$

e) $2^{10} + n$

f) $2^{10} n$

Oppgåve 3

a) Implementer ein metode `sorterFrekvens(Comparable[] a)` som får inn ein tabell som kan innehalda fleire like element (fleire element med same nøkkelverdi), og som sorterer tabellen etter avtakande *frekvens*. Med frekvensen til eit element `a[i]` meiner vi talet på element `a[j]` slik at `a[i].compareTo(a[j])==0`. Er alle elementa ulike, har altså kvart element frekvensen 1. Element `a[i]` og `a[j]` med same frekvens skal sorterast stigande etter kriteriet gitt med `compareTo`, dvs. slik at `a[i]` står før `a[j]` dersom `a[i].compareTo(a[j])<0`.

Eksempel: Dersom metoden får inn `a={3, 4, 1, 8, 5, 3, 5, 5}`, vil resultatet bli `a={5, 5, 5, 3, 3, 1, 4, 8}`.

b) Uttrykk kjøretida til metoden med ein Θ-relasjon.

c) Elementa i ein usortert todimensjonal tabell `Comparable[][] b` skal sorterast til ein eindimensjonal tabell `Comparable[] a`. Dette skal gjerast enten ved å

   i) kopiera elementa i `b` til `a` og deretter flettesortera `a`, eller ved å

   ii) sortera kvar rekkje i `b`, og deretter fletta saman rekkjene over i `a`.

   For kvar av desse metodane, uttrykk talet på samanlikningar i verste tilfelle ved hjelp av ein $\sim$-relasjon. I denne analysen går vi ut frå at talet på både rekkjer og søyler i `b` er $n = 2^k$, der k er eit heiltal. Du kan gjera bruk av *Proposition F*, s. 272 i *Sedgewick & Wayne*.

d) Implementer ein av metodane i oppgåve c).

e) Kvar av dei m medlemmene i eit sosialt nettverk oppgir n ulike tema som dei likar. Skriv eit program som les inn m, n og oversikt over kva tema som blir likt, og som skriv ut alle tema rangert etter kor mange som likar dei (mest populære tema først). Tema med lik popularitet skal skrivast i alfabetisk rekkefølgje. Alle data blir lest frå ei tekstfil (sjå den vedlagte fila `likar.txt`) med m og n på første line, og deretter m liner med n tema separert med mellomrom.

Oppgåve 4

Vi skal gjera ei eksperimentell samanlikning av kjøretida til algoritmane *utvalssortering*, *shellsortering* og *flettesortering*. Vel sjølv om du vil bruka dine eigne eller læreboka sine implementasjonar. Ei lita omskriving av SortCompare.java vil vera nyttig for å måla tida på algoritmane.

a) Varier n frå 10000 til 200000 med intervall på 10000, generer 10 flyttalstabellar med lengd n, og mål gjennomsnittleg kjøretid for kvar algoritme. For $n \geq 100000$ er det nok å generera berre ein tabell. Plott kjøretidene, samanlikn og kommenter.

b) For algoritmane *shellsortering* og *flettesortering*, gå vidare og varier n til 1000000 med intervall på 100000. Mål og plott kjøretidene, og kommenter resultatet.

c) Basert på vidare eksperiment, gi eit estimat på kor store tabellar kvar av algoritmane klarer å sortera på eit minutt kjøretid på maskinen du brukar.