Institut for Matematik og Datalogi
Syddansk Universitet, Odense

4. februar 2016
RF

DM507 – Opgaver uge 6

NB: I denne uge er der lidt mere matematik i opgaverne til Eksaminatorier II end normalt i kurset, hvilket skyldes karakteren af emnet (asymptotisk analyse af voksehastighed).

Erfaringsmæssigt varierer deltagernes fortrolighed med Java-programmering en del. Hvis man ikke føler sig alt for stærk på det område, er det netop vigtigt at bruge tid på de programmeringsopgaver, som stilles. De er ikke mange eller store, men vil være god opvarmning til projektet. Har man god erfaring med programmering, kan man i stedet lægge en større indsats i de andre opgaver.

Eksaminatorier I

1. Cormen et al. øvelse 2.1-1 (side 22).
2. Cormen et al. øvelse 2.1-3 (side 22). Lav blot pseudo-koden (ikke delen med invarianter, vi vender tilbage til begrebet invarianter senere i kurset).
3. Cormen et al. øvelse 2.2-3 (side 29). Svar også for best-case køretid.
4. Cormen et al. øvelse 2.3-5 (side 39).
5. Cormen et al. øvelse 2.3-6 (side 39).
6. Cormen et al. øvelse 2.3-1 (side 37).
7. Cormen et al. øvelse 2.3-2 (side 37).
8. (∗) Cormen et al. opgave 2-1 (side 39). Til spørgsmål d: tanken her er at bruge køretid i praksis til at fin-tune det endelige valg af k.
9. (∗) Cormen et al. opgave 2-4 (side 41), spørgsmål **a**, **b** og **c**.
10. (∗) Cormen et al. øvelse 2.3-7 (side 39). Hint: start med at sortere tallene. Du må gerne bruge at dette kan gøres i $O(n \log n)$ tid (f.eks. med Mergesort).

Eksaminatorier II

1. Implementer InsertionSort i Java ud fra pseudo-koden side 18 i lærebogen. Test at din kode fungerer ved at generere arrays med forskelligt indhold og sortere dem. [NB: Bogens pseudokode indekserer arrays startende med index 1, mens Java starter med index 0. Man kan enten i Java bruge et array af længde $n+1$ og blot aldrig bruge index 0 (så kan bogens pseudo-kode bruges direkte), eller man kan ændre passende i pseudokodens initialisering af indekser.]

   Tilføj tidtagning af din kode ved at indsættte to kald til metoden `System.currentTimeMillis()`, eet i starten af InsertionSort og eet i slutningen (slå funktionaliteten af metoden op i Javas online dokumentation). Der skal kun tages tid på selve sorteringen, ikke den del af programmet som genererer array'ets indhold.

   Kør din kode dels med sorteret input (best case for InsertionSort), dels med omvendt sorteret input (worst case for InsertionSort). Gør dette for mindst 5 forskellige værdier af n (antal elementer at sortere). Vælg disse værdier af n så de får programmet til at bruge fra ca. 100 til ca. 5000 millisekunder (værdierne er ikke de samme for best case og worst case). Gentag hver enkelt kørsel tre gange og find gennemsnittet af antal millisekunder brugt ved de tre kørsler (fluktuationer fra baggrundsprocesser får derved mindre indflydelse). Dividér de fremkomne tal med henholdsvis n (for best case input) og n^2 (for worst case input), og check derved hvor godt analysen passer med praksis – de resulterende tal burde ifølge analysen være konstante (for best case tallene og for worst case tallene, hver for sig), jvf. graferne på slides fra forelæsningen.

   Kør derefter din kode med input, som er random `int`'s (brug f.eks. `java.util.Random`, se evt. vejledningen på `http://www.javapractices.com/topic/TopicAction.do?Id=62`, example 1). Er køretiderne tættest på best case eller worst case?

2. Vis for
$$f(n) = 0.1 \cdot n^2 + 5 \cdot n + 25$$
   at $f(n) = \Theta(n^2)$ og $f(n) = o(n^3)$. Hint: brug sætning fra slides om analyse af algoritmers køretider.

3. Vis at følgende funktioner er skrevet op efter stigende asymptotisk voksehastighed:

$$1,\ \log n,\ \sqrt{n},\ n/\log n,\ n,\ n\log n,\ n\sqrt{n},\ n^2,\ n^3,\ n^{10},\ 2^n$$

Mere præcist, vis at det for alle par $f(n), g(n)$ af naboer i listen gælder at $f(n) = o(g(n))$. Hint: brug sætninger fra slides om analyse af algoritmers køretider.

4. Cormen et al. øvelse 3.1-4 (side 53).

5. (∗) Cormen et al. øvelse 3.2-3 (side 60). Hint: for at vise formel (3.19) side 58 kan man godt bruge Stirlings formel (3.18) side 57, sådan som bogen foreslår i tekst ved (3.19), men man kan også argumentere langt mere jordnært ud fra definitionen af $n!$. Husk regnereglerne for logaritmer (se (3.15) side 56, specielt anden regel).

6. (∗) Cormen et al. øvelse 3.1-1 (side 52). (Her skal man bruge selve definitionen af $\Theta()$ fra bog/slides, ikke sætninger fra slides.)

7. (∗∗) Bevis (ud fra definitioner af $\Theta()$ og $o()$ og af begrebet grænseværdi) sætningerne

$$\text{Hvis } \frac{f(n)}{g(n)} \to k > 0 \text{ for } n \to \infty \text{ så gælder } f(n) = \Theta(g(n))$$

og

$$\text{Hvis } \frac{f(n)}{g(n)} \to 0 \text{ for } n \to \infty \text{ så gælder } f(n) = o(g(n))$$

fra slides om analyse af algoritmers køretider. Husk at definitionen af grænseværdi er: $h(n) \to k$ for $n \to \infty$ hvis der for ethvert lille tal $\epsilon > 0$ findes et N_0 så $|h(n) - k| \leq \epsilon$ når $n \geq N_0$. Du må antage at $f(n)$ og $g(n)$ er positive funktioner (jvf. side 45 nederst i Cormen et al.).

Studiegrupper

Forslag til fokus for arbejde i studiegrupper (hvis man er i en sådan):

Gennemgå afsnit 3.2 og diskutér hvilke af afsnittets ligninger og definitioner, I (alle eller blot nogle af jer) har mødt før i enten gymnasiet eller et kursus på SDU.

Forbered dele af opgaverne til eksaminatorietimer, f.eks. på nedenstående måde.

- I opgave II.1, lav programmeringen og programkørsel (evt. i par) før gruppemøde. På mødet, sammenlign køretider (er jeres maskiner f.eks. lige hurtige?).

- Løs først opgave II.2 i fælleskab. Del jer derefter i to eller flere grupper, som tager hver sine dele (hver sine nabopar af funktioner) af opgave II.3. Forklar bagefter løsningerne/beregningerne for hinanden (alle bør forklare mindst én).

- Forsøg at løse de mere kreative og udfordrende af opgaverne i fælleskab (f.eks. I.8, I.9, I.10, II.4, II.5, II.6, II.7). Arbejd både med at få ideer på skitseplanet til de ønskede algoritmer og argumenter, og med at få dem formuleret præcist til sidst. I kan evt. dele disse opgaver op imellem delgrupper, som senere forsøger at formidle de fundne løsninger til hinanden så klart og præcist som muligt.