Institut for Matematik og Datalogi
Syddansk Universitet, Odense

14. februar 2026
RF

Opgaver Uge 8

DM507/DM578/DS814/SE4-DMAD

Husk at i DM507/DM578/DS814, samt den del af SE4-DMAD som handler om algoritmer og datastrukturer (Rolfs del), er opgaverne til øvelsestimerne (også kaldet eksaminatorier eller e-timer) delt i to grupper:

A. En første samling opgaver, som man løser *i øvelsestimerne* med instruktoren til rådighed for spørgsmål undervejs og med fælles opsamling til sidst. Disse opgaver skal altså *ikke* løses på forhånd. Man skal blot have læst på stoffet fra forelæsningen. Man må meget gerne arbejde i grupper i timerne (oplagt, hvis man er i en studiegruppe allerede), da det er en stor hjælp af tale højt med andre om løsning af opgaver.

B. En anden samling opgaver med samme type stof som den første samling. Svaret på disse opgaver løber man kort igennem til sidst i øvelsestimen mht. mulige løsningsmetoder, men opgaverne skal løses *hjemme* (gerne sammen med sin studiegruppe) inden de *næste* øvelsestimer. Disse opgaver er der så kort opsamling på i starten af den næste øvelsestime.

I uger med to øvelsesgange for hvert hold (som i denne uge) vil der på en ugeseddel være to udgaver af ovenstående. Hver udgave angår én af øvelsesgangene, angivet i opgavegruppernes oveskrifter.

På ugesedlerne vil lidt svære opgaver vil være mærket med $(*)$, og mere svære opgaver vil være mærket med $(**)$.

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

Erfaringsmæssigt varierer deltagernes fortrolighed med 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 give træning i at overføre algoritmer fra idé- og pseudokode-niveau til færdig kode. De vil også være en god opvarmning til projektet i DM507/DS814. Har man god erfaring med programmering, kan man i stedet lægge en større indsats i de andre opgaver.

I.A: Løses i løbet af de første øvelsestimer i uge 8

1. Cormen et al., 4. udgave, øvelse 2.2-3 (side 33) [Cormen et al., 3. udgave: øvelse 2.2-3 (side 29)]. Svar også for best-case køretid. [NB: Der refereres til en tidligere opgave for beskrivelsen af linear search. Denne opgave skal ikke løses, man skal blot læse den korte beskrivelse af linear search der.]

2. Cormen et al., 4. udgave, øvelse 2.1-1 (side 24) [Cormen et al., 3. udgave: øvelse 2.1-1 (side 22)]. Du skal blot udføre Insertionsort i hånden, og behøver ikke lave tegningerne 100% som i bogen.

3. Implementer InsertionSort i Python, C# eller Java ud fra pseudokoden i Cormen et al., 4. udgave, side 19 [Cormen et al., 3. udgave: side 18]. Test at din kode fungerer ved at generere arrays/lister med forskelligt indhold og sortere dem. NB: Bogens pseudokode indekserer arrays startende med index 1, mens Python, C# og Java starter med index 0. Man må derfor ændre passende (dvs. nogle gange bruge et index som er én mindre) i de linier i pseudokoden, som involverer indekser.

4. ($*$) Cormen et al., 4. udgave, opgave 2-4 (side 47) [Cormen et al., 3. udgave: opgave 2-4 (side 41)], spørgsmål **a**, **b** og **c**.

   Hints til spørgsmål **c**: Argumenter for, at hvert move af et element i InsertionSort fjerner præcis én inversion. Argumenter derefter for, at det meste arbejde i InsertionSort følger antallet af moves, og at resten af arbejdet tager $\Theta(n)$ tid. Argumenter til sidst for, at det heraf følger, at køretiden for InsertionSort er $\Theta(n + \mathrm{INV})$

I.B: Løses hjemme inden de næste øvelsestimer i uge 8

1. Fortsæt opgaven ovenfor med implementation af InsertionSort på denne måde:

   Tilføj tidtagning til din kode ved at lave et timestamp i starten af InsertionSort og et timestamp i slutningen. Et timestamp kan i Python fås med et kald til metoden `time.time()` fra modulet `time`, det kan i C# fås ved at læse property `ElapsedMilliseconds` fra et `Stopwatch` (som er blevet startet) fra namespace `System.Diagnostics`, og det kan i Java fås med et kald til metoden `System.currentTimeMillis()`.

   Ved alle fremgangsmåder ovenfor fås et timestamp, som er et heltal (i C# og Java af typen `long`), og som angiver antal millisekunder fra et fast tidspunkt i fortiden. Ved at trække det første timestamp fra det sidste fås antal millisekunder, som koden har brugt. Der skal kun tages tid på selve sorteringen, ikke den del af koden som genererer array'ets/listens 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 på input, som er random `int`'s. I Python, brug f.eks. metoden `random.randint()` fra modulet `random` til at generere dem. I C#, brug f.eks. et `Random` objekt og dets metode `Next()`. I Java, brug f.eks. et `java.util.Random` objekt og dets metode `nextInt()`. Er køretiderne tættest på best case eller worst case?

2. ($*$) Cormen et al., 4. udgave, øvelse 2.3-8 (side 45) [Cormen et al., 3. udgave: 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).

II.A: Løses i løbet af de næste øvelsestimer i uge 8

1. Cormen et al., 4. udgave, øvelse 2.3-1 (side 44) [Cormen et al., 3. udgave: øvelse 2.3-1 (side 37)].

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 (1) og (2) fra slides om analyse af algoritmers køretider.

3. Vis at følgende funktioner er skrevet op efter stigende asymptotisk voksehastighed:
$$1,\ \log n,\ \sqrt{n},\ n,\ n \log n,\ n\sqrt{n},\ n^2,\ n^3,\ n^{10},\ 2^n$$
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 sætning (1)–(4) fra slides om analyse af algoritmers køretider.

4. ($*$) Cormen et al., 4. udgave, øvelse 3.2-1 (side 62) [Cormen et al., 3. udgave: øvelse 3.1-1 (side 52)]. (Her skal man bruge selve definitionen af $\Theta()$ fra bog/slides, ikke sætning (1)–(4) fra slides.)

II.B: Løses hjemme inden øvelsestimerne i uge 9

1. Cormen et al., 4. udgave, øvelse 2.3-6 (side 44) [Cormen et al., 3. udgave: øvelse 2.3-5 (side 39)]. Udvid opgaven således: Start med at illustrere algoritmen med en tegning. Lav derefter pseudokode for algoritmen (som i opgaveteksten). Lav derefter et program i Python, C# eller Java på basis af din pseudokode. Test til sidst korrekthed af din implementation ved at generere lister (Python) eller arrays (C# og Java) med indholdet 1,3,5,7,.... og derefter udføre søgninger. Test med både tomme lister/arrays, lange lister/arrays, og søgning både efter tal, som er der, og efter tal, som ikke er der.

2. Cormen et al., 4. udgave, øvelse 2.3-7 (side 45) [Cormen et al., 3. udgave: øvelse 2.3-6 (side 39)].

3. Cormen et al., 4. udgave, øvelse 3.2-3 (side 62) [Cormen et al., 3. udgave: øvelse 3.1-4 (side 53)]. Hint: husk regneregler for potenser (kan genopfriskes Cormen et al., 4. udgave, side 65 midt på [Cormen et al., 3. udgave: side 55 midt på]).

4. $(*)$ Vis at $n! = \omega(2^n)$ og $n! = o(n^n)$.