

Opgaver Uge 8

DM507/DS814/T510040101

Opsummering fra forelæsninger: der er mindst tre naturlige niveauer at beskrive en algoritme på:

1. Som idé beskrevet i ord og tegninger.
2. Som pseudo-kode.
3. Som færdig kode (i f.eks. Java eller Python).

Dette er også de naturlige stadier at passere, når man skal udvikle algoritmer til et program i den virkelige verden. Hovedidéen i en algoritme er allerede på plads på niveau 1, og korrekthedsanalyse og asymptotisk analyse med grovsortering af algoritmer efter voksehastighed kan normalt finde sted der. Niveau 2 tilføjer flere detaljer, som skal være på plads for at sikre korrektheden af koden i sidste ende. Det kan ofte tage overraskende lang tid. Det kan betale sig at være omhyggelig med at lave korrekthedsanalyse af detaljerne (initialisering og opdatering af variable, f.eks.) i ens forslag til pseudo-kode, før man går videre til næste niveau. Hvis niveau 2 er lavet grundigt, bliver niveau 3 ofte relativt nemt.

Til forelæsningerne udtrykker vi os primært på niveau 1 og til dels på niveau 2. Lærebogen udtrykker sig også på niveau 1 og niveau 2, med de fleste algoritmer angivet meget detaljeret på niveau 2. I projektet arbejder vi med niveau 3, ofte baseret på niveau 2 fra bogen.

Til øvelsestimerne er det i opgaver, hvor man skal udvikle, beskrive og/eller analysere algoritmer, nok at gøre dette på niveau 1, medmindre andet er angivet (dvs. medmindre man direkte bliver bedt om pseudo-kode eller Java/Python-implementation).

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

1. Cormen et al. øvelse 7.1-1 (side 173).
2. Cormen et al. øvelse 7.1-2 (side 174). Her er q værdien som PARTITION returnerer, dvs. $(i + 1)$ i linie 8 i pseudo-koden i Cormen et al. side 171. Hint til det andet spørgsmål: Start med at lave princippet for det mørkegrå område på figur 7.2 (side 173) om fra " $> x$ " til " $\geq x$ " (overvej hvorfor dette ikke vil ændre på korrektheden af selve Quicksort), og fyld så det lysegrå og det mørkegrå område så balanceret som muligt under udførelsen af PARTITION.
3. Cormen et al. øvelse 7.2-2 (side 178). Antag, at det er PARTITION fra side 171, der bruges (ikke f.eks. PARTITION varianten lavet i øvelse 7.1-2 ovenfor). Hint: i øvelse 7.1-2 (side 174), som er løst ovenfor, er der undersøgt i detaljer hvad PARTITION fra side 171 gør på det pågældende input. Argumenter for køretiden, når dette sker gentagne gange i Quicksort.
4. Cormen et al. øvelse 7.2-3 (side 178). Vis at det faktisk gælder både når input er stigende sorteret og når det er aftagende sorteret (start med stigende sorteret). Hint: undersøg i detaljer hvad PARTITION fra side 171 gør på de pågældende input. Argumenter så for køretiden, når dette sker gentagne gange i Quicksort.
5. Cormen et al. øvelse 6.1-5 (side 154).
6. Cormen et al. øvelse 6.1-6 (side 154).

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

1. Eksamen juni 2008, opgave 1 b. Man må gerne referere til oplysninger i bogen, når man giver begrundelser. Quicksort antages at bruge PARTITION fra side 171.
2. Cormen et al. problem 7-2, spørgsmål b (side 186). Svar derefter igen på Cormen et al. øvelse 7.2-2 (side 178) og øvelse 7.2-3 (side 178), under antagelse af at Quicksort bruger den nye PARTITION procedure (og naturligvis laver sine to rekursive kald på de to dele af array A hvor PARTITION har placeret elementer forskellige fra $A[q]$).

3. Eksamen juni 2008, opgave 4 a. Hob er en fordanskning af ordet heap.
4. Cormen et al. øvelse 6.1-4 (side 154).

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

1. Eksamen januar 2008, opgave 1 b (sidehenvisningen skal være til side 164 i vores udgave (tredie) af Cormen et al.).
2. Eksamen januar 2006, opgave 1 b. Bemærk at der er tale om en min-heap.
3. Cormen et al. øvelse 6.2-1 (side 156).
4. Udfør HEAP-EXTRACT-MAX(A) på nedenstående max-heap A .

$A:$	1	2	3	4	5	6	7	8	9	10
	21	18	10	12	8	9	4	7	5	2

5. Implementer Quicksort i Java eller Python ud fra bogens pseudokode (side 171). Test at din kode fungerer ved at generere arrays med forskelligt indhold og sortere dem. Tilføj tidstagning af din kode (kun selve sorteringen, ikke den del af programmet som genererer array'ets indhold).

Kør derefter din kode med input, som er random `int`'s. Gør dette for mindst 5 forskellige værdier af n (antal elementer at sortere), vælg værdier som får programmet til at bruge fra ca. 100 til ca. 5000 millisekunder. Gentag hver enkelt kørsel tre gange og find gennemsnittet af antal millisekunder brugt ved de tre kørsler. Dividér de fremkomne tal med $n \log_2 n$ og check derved hvor godt analysen passer med praksis – de resulterende tal burde ifølge analysen være konstante.

Sammenlign med dine køretider for det tilsvarende forsøg med Mergesort fra opgaverne i uge 8. Er Quicksort eller Mergesort hurtigst?

[Ekstraopgave: prøv en let optimeret variant af Quicksort, hvor pivot-elementet x i PARTITION vælges ved at se på de tre elementer på første, midterste og sidste plads i den del af array'et, som skal partitioneres. Disse tre elementer sammenlignes indbyrdes, og det ordningsmæssigt midterste (medianen) af disse tre bruges som pivot-element. Gør dette for kald til PARTITION, hvor der er 16 elementer eller mere, men ikke for

kald på mindre instanser (her bruges stadig bogens PARTITION). Kører denne version af Quicksort (lidt) hurtigere end standardversionen?]

I Java, gentag derefter eksperimenterne med Java's sorteringsmetode `sort` fra klassen `java.util.Arrays` (en endnu mere optimeret Quicksort implementation), og sammenlign køretiderne med dine egne implementationer af Quicksort og Mergesort. I Python, gør det samme med `sort()` metoden fra `list` (denne bruger Timsort, der er en variant af Mergesort, som er designet til at udnytte eventuel eksisterende orden i input).

II.B: Løses hjemme inden øvelsestimerne i uge 9

1. Cormen et al. øvelse 6.5-9 (side 166).
2. (*) Bevis at de beregnede indekser i `PARENT`, `LEFT` og `RIGHT` på side 152 er korrekte (dvs. for en knude med arrayindeks i altid er indeks af dens forælder, venstre barn og højre barn). Hint: Start f.eks. med at vise det for knuder på stien helt til venstre i heapen. Eller brug induktion på indeks af forælderknuden.
3. (*) Cormen et al. problem 6.2 (side 167). Lav spørgsmål **e** før spørgsmål **d**. For spørgsmål **b**, brug f.eks. formel (A.5) side 1147.