Institut for Matematik og Datalogi
Syddansk Universitet, Odense

18. februar 2022
RF

Opgaver Uge 8

SE4-DMAD

A: Løses i løbet af øvelsestimerne 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 udførslen 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. 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.

6. 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 $A[q]$).

7. Cormen et al. øvelse 6.1-5 (side 154).

8. Cormen et al. øvelse 6.1-6 (side 154).

9. Eksamen januar 2008, opgave 1 b (sidehenvisningen skal være til side 164 i vores udgave (tredie) af Cormen et al.).

10. Eksamen januar 2006, opgave 1 b. Bemærk at der er tale om en min-heap.

11. Cormen et al. øvelse 6.2-1 (side 156).

12. Udfør HEAP-EXTRACT-MAX(A) på nedenstående max-heap A.

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

13. Cormen et al. øvelse 6.1-4 (side 154).

B: Løses hjemme inden øvelsestimerne i uge 9

1. 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 7. Er Quicksort eller Mergesort hurtigst?

[Ekstraopgave: prøv en let optimeret variant af Quicksort, hvor pivotelementet 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 lister (denne bruger Timsort, der er en variant af Mergesort, som er designet til at udnytte eventuel eksisterende orden i input).

2. Eksamen juni 2008, opgave 4 a. Hob er en fordanskning af ordet heap.

3. Cormen et al. øvelse 6.5-9 (side 166).

4. ($*$) 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.

5. ($*$) 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.