Institut for Matematik og Datalogi
Syddansk Universitet, Odense

13. februar 2013
RF

DM507 – Opgaver uge 8

Eksaminatorier I

1. Eksamen juni 2008, opgave 1 b. Man må gerne referere til oplysninger i bogen, når man giver begrundelser.
2. Cormen et al. øvelse 6.1-5 (side 154).
3. Cormen et al. øvelse 6.1-6 (side 154).
4. Eksamen juni 2008, opgave 4 a. Hob er en fordanskning af ordet heap.
5. Cormen et al. øvelse 6.1-4 (side 154).
6. Eksamen januar 2008, opgave 1 b (sidehenvisningen skal være til side 164 i vores udgave (tredie) af Cormen et al.).
7. Eksamen januar 2006, opgave 1 b. Bemærk at der er tale om en min-heap.
8. Cormen et al. øvelse 6.2-1 (side 156).
9. Cormen et al. øvelse 6.5-9 (side 166).
10. Cormen et al. øvelse 6.4-4 (side 160). Hint: Er Heapsort en sammenligningsbaseret algoritme?
11. Bevis at de beregnede indexer i PARENT, LEFT og RIGHT på side 152 er korrekte (dvs. for en knude på arrayindex i altid giver index af dens forælder, venstre barn og højre barn). Hint: start med at vise det for knuder på stien mest til venstre i heapen.
12. [Let udfordrende] Cormen et al. problem 6.2 (side 167).
13. [Udfordrende] Cormen et al. problem 7-4 (side 188). Hint til del c: vælg hvilken del man vil kalde rekursivt på, i stedet for at bruge den venstre del altid.

Eksaminatorier II

1. Cormen et al. øvelse 8.2-1 (side 196).
2. Cormen et al. øvelse 8.2-4 (side 197).
3. Eksamen juni 2008, opgave 1a.
4. Cormen et al. øvelse 8.3-1 (side 199).
5. Cormen et al. øvelse 8.3-4 (side 200).
6. Cormen et al. øvelse 8.3-2 (side 200). Hint til sidste del: udvid elementers nøgler.
7. Cormen et al. øvelse 2.1-3 (side 22). Kun spørgsmålet om invariant, resten er lavet i uge 6.
8. Cormen et al. øvelse 2.2-2 (side 29).
9. Cormen et al. opgave 2-2 (side 40).
10. Implementer Quicksort i Java ud fra bogens pseudokode (side 171). Test at din kode fungerer ved at generere arrays med forskelligt indhold og sortere dem. Tilføj tidtagning 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. Kører denne version af Quicksort (lidt) hurtigere end standardversionen?]

Gentag derefter eksperimenterne med Java's (optimerede) sorteringsmetode `sort` fra klassen `java.util.Arrays`, og sammenlign køretiderne med dine implementationer af Quicksort og Mergesort.

Studiegrupper

Forslag til fokus for arbejde i studiegrupper:

Genfortæl for hinanden nogle af korrekthedsbeviserne fra forelæsningsslides. F.eks. hvorfor Countingsort er stabil, hvorfor Radixsort sorterer. Eventuelt også beviset for den nedre grænse på $\Omega(n \log n)$ for sortering ved sammenligningsbaserede algoritmer.

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

- Forsøg at lave de mere simple af opgaverne (f.eks. I.1–8, I.10, II.1–4, II.7–8) individuelt især på forhånd, og sammenlign svar i studiegruppen.
- Forsøg at løse de mere kreative og udfordrende af opgaverne i fælleskab (resten, undtagen II.10). 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 dise opgaver op imellem delgrupper, som senere forsøger at formidle de fundne løsninger til hinanden så klart og præcist som muligt.
- I opgave II.10, lav programmeringen og programkørsel (evt. i par) før gruppemøde. I studiegruppen, sammenlign køretider.