Institut for Matematik og Datalogi
Syddansk Universitet, Odense

13. februar 2021
RF

Opgaver Uge 7

DM507/DS814/T510040101

A: Løses i løbet af øvelsestimerne i uge 7

1. Eksamen juni 2008, opgave 2 (tidligere eksamensopgaver kan findes øverst på kursets hjemmeside).

2. Eksamen juni 2011, opgave 2. [Hint: For et sandt svar, argumenter ud fra definitionerne på O og Ω. For et falsk svar, find modeksempler.] Opgaven bruger notationen $f(n) \in O(g(n))$, hvilket betyder det samme som $f(n) = O(g(n))$. Se evt. diskussion side 45 i bogen.

3. Implementer Mergesort i Java eller Python ud fra bogens pseudokode (side 31 og 34). Lad input være et array (Java) eller liste (Python) af heltal. Som værdien ∞ (pseudokode side 31) brug et stort tal, og sørg for at kun heltal under denne værdi optræder i input. Alternativt (og bedre, da man undgår at skulle tage hensyn til værdien valgt som ∞), brug til MERGE din pseudokode fra øvelse 2.3-2 (fra seddel med opgaver til uge 7), som ikke anvender ∞.[1]

   Test at din kode fungerer ved at generere små arrays/lister med forskelligt indhold (f.eks. stigende, faldende, tilfældige), sortere dem og checke resultatet.

4. ($*$) Cormen et al. opgave 2-4 (side 41), spørgsmål **d**.

[1] Faktisk har både Java og Python en indbygget værdi for ∞ (i Java `Double.POSITIVE_INFINITY`, i Python `float("inf")`). Disse er teknisk set kommatal, men fungerer efter hensigten, hvis de sammenlignes med heltal. I Java kan et kommatal og et heltal dog ikke optræde i samme array, så bogens pseudokode kan ikke implementeres vha. denne værdi. I Python kan et kommatal og et heltal godt optræde i samme liste, så der kan bogens pseudokode godt implementeres vha. denne værdi.

B: Løses hjemme inden øvelsestimerne i uge 8

1. Eksamen januar 2007, opgave 3. Opgaven bruger notationen $f(n) \in O(g(n))$, hvilket betyder det samme som $f(n) = O(g(n))$.

2. Fortsæt opgaven ovenfor med implementation af Mergesort på denne måde:

   Tilføj tidtagning af din kode. Du skal kun tage tid på selve sorteringen, ikke den del af programmet som genererer array'ets/listens indhold. Kør derefter din kode med input, som er tilfældige heltal. 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. Divider 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.

   [Hint: Se seddel med opgaver til uge 7 for informationer om bogens indeksering i forhold til Javas/Pythons, samt om metoder i Java og Python til tidstagning og til generering af tilfældige heltal. For at beregne logaritmer, brug i Java metoden `java.lang.Math.log(x)`, som beregner $\log_e(x)$, dvs. logaritmen med grundtal e. For at beregne $\log_2(x)$ kan man bruge at $\log_2(x) = \log_e(x)/\log_e(2)$ (jvf. formel 3.15 i bogen side 56). I Python kan man bruge metoden `math.log(x,2)` til at beregne $\log_2(x)$ (husk at importere `math` modulet).]