DMA ugeseddel 2

Litteratur

Uge 2 vil vi primært bruge på:

- CLRS kapitel 2.

Mål for ugen

- Forståelse for vigtigheden af konstruktion af effektive algoritmer.
- Kendskab til lineær og binær søgning.
- Kendskab til insertion sort og merge sort.
- Del og hersk-teknikken.

Plan for ugen

- Mandag: Lineær og binær søgning. Insertion sort.
- Tirsdag: Merge sort.
- Fredag: Opsummering af og afrunding på de første to uger i DMA med algoritmik.

Opgaver til mandag

Pas på: Opgaver markeret med (*) er svære, (**) er meget svære, og (***) har du ikke en chance for at løse. I bunden af ugesedlen finder du flere ekstraopgaver, hvoraf nogle er svære. Dem kan du lave hvis du mangler udfordring eller er færdig med dagens opgaver.

Bemærk: I nedenstående bruges flere gange udtryk som $\Theta(n)$, $\Theta(n \log n)$, og $\Theta(n^2)$. Disse udtryk er forklaret løst i pensum fra denne ugeseddel og til forelæsningerne. De vil blive defineret mere præcist senere i kurset. Indtil videre kan du bruge følgende løse definition: Lad os fjerne konstanter og langsomt voksende led fra det totale antal operationer som algoritmen laver, når man kører den på input af størrelse n. Hvis vi står tilbage med n^2, så er køretiden $\Theta(n^2)$. Dette er forklaret i afsnittet "Order of growth", CLRS side 28–29.

1. **Håndkøring og egenskaber.** Løs følgende opgaver.
   (a) CLRS 2.1-1.
   (b) Giv et eksempel på et array A af længde 5 så while-løkken på linje 5 i pseudokoden af insertion sort (CLRS side 18) laver det højest mulige antal iterationer. Hvilken egenskab har arrays der får antallet af iterationer til at blive størst muligt?

(c) Giv et eksempel på et array A af længde 5 så while-løkken på linje 5 i pseudokoden af insertion sort (CLRS side 18) laver det lavest mulige antal iterationer. Hvilken egenskab har arrays der får antallet af iterationer til at blive mindst muligt?

(d) CLRS 2.1-2.

(e) CLRS 2.3-6.

2. **Manglende tal.** Lad A være et array af længde $n-1$ således at indgangene i A er tal i mængden $\{0, 1, 2, \ldots, n-1\}$, hvor det oplyses at alle indgangene er forskellige. Derfor er der altså et enkelt tal i mængden $\{0, 1, \ldots, n-1\}$ som ikke optræder i A, og dette kalder vi det *manglende* tal. F.eks. med $n = 5$ og $A = [2, 0, 4, 3]$ er det manglende tal $m = 1$. Vi er interesserede i effektive algoritmer til at finde det manglende tal i A. Løs følgende opgaver.

   (a) Giv en algoritme der løser problemet i $\Theta(n)$ tid. *Hint:* Brug et ekstra array af længde n.

   (b) Vi vil nu gerne løse problemet hurtigt, men også begrænse pladsforbruget så meget som muligt. Giv en algoritme der løser problemet i $\Theta(n^2)$ tid og kun bruger et konstant antal ekstra variable.

   (c) Giv en algoritme der løser problemet i $\Theta(n)$ tid og kun bruger et konstant antal ekstra variable. *Hint:* Husk sumformlen $1 + 2 + \ldots + n = \frac{n\cdot(n+1)}{2}$.

   (d) Antag nu at A er sorteret. Giv en algoritme der løser problemet i $\Theta(\log n)$ tid. *Hint:* Benyt en form for binær søgning.

3. **2-sum og 3-sum.** Lad $A[0 \ldots n-1]$ være et array af heltal (positive og negative). Tabellen A har en *2-sum*, hvis der findes indgange i og j, så $A[i] + A[j] = 0$. Tilsvarende har A en *3-sum*, hvis der findes indgange i, j og k, så $A[i] + A[j] + A[k] = 0$. (I disse definitioner kræves det ikke at i, j, k er forskellige tal.) Løs følgende opgaver.

   (a) Giv eksempler på arrays af længde 5 som har og ikke har en 2-sum.

   (b) Giv eksempler på arrays af længde 5 som har og ikke har en 3-sum.

   (c) Giv en algoritme, der afgør om A har en 2-sum i $\Theta(n^2)$ tid.

   (d) Giv en algoritme, der afgør om A har en 2-sum i $\Theta(n \log n)$ tid. *Hint:* Benyt merge sort som kan sortere i $\Theta(n \log n)$ tid, samt binær søgning.

   (e) Giv en algoritme, der afgør om A har en 3-sum i $\Theta(n^3)$ tid.

   (f) Giv en algoritme, der afgør om A har en 3-sum i $\Theta(n^2 \log n)$ tid. *Hint:* Benyt binær søgning.

   (g) (***) Giv en algoritme, der afgør om A har en 3-sum i $\Theta(n^2)$ tid. *Hint:* Lav først opgave 2 (c) fra tirsdagens opgaver på ugeseddel 1.

Opgaver til tirsdag

1. **Merge.** Håndkør merge-proceduren på følgende input:

   (a) $[1, 3, 4, 7, 8]$ og $[2, 4, 5, 7, 8]$.

   (b) $[5, 8, 11, 14]$ og $[7, 8, 13, 19]$.

   (c) $[2, 5, 7, 7, 9]$ og $[11, 23, 41, 59, 89]$

2. **Toppunkter.** For hver af de tre top punktsalgoritmer nedenfor, beskriv hvilke input der får køretiden til at blive så høj som muligt.

```
Toppunkt1(A, n)
  if A[0] >= A[1]
    return 0
  for i = 1 to n - 2
    if A[i - 1] <= A[i] and A[i] >= A[i + 1]
      return i
  if A[n - 2] <= A[n - 1]
    return n - 1
```

```
Toppunkt2(A, n)
  max = 0
  for i = 0 to n - 1
    if A[i] > A[max]
      max = i
  return max
```

```
Toppunkt3(A, i, j)
  m = ceiling((i + j) / 2)
  if A[m] is toppunkt
    return m
  else
    if A[m - 1] > A[m]
      return Toppunkt3(A, i, m - 1)
    else
      return Toppunkt3(A, m + 1, j)
```

3. **Duplikater og tætte naboer.** Lad $A[0 \dots n-1]$ være et array af heltal. Løs følgende opgaver.

   (a) Et *duplikat* i A er et par af forskellige indgange i og j så $A[i] = A[j]$. Giv en algoritme der afgør om der er et duplikat i A i $\Theta(n^2)$ tid.

   (b) Giv en algoritme der afgør om der er et duplikat i A i $\Theta(n \log n)$ tid. *Hint:* Benyt merge sort som kan sortere i $\Theta(n \log n)$ tid.

   (c) Et *tætteste par* i A er et par af indgange i og j så forskellen $|A[i] - A[j]|$ er minimal blandt alle par af indgange. Giv en algoritme der finder et tætteste par i A i $\Theta(n \log n)$ tid.

Opgaver til fredag

1. **Mergesort.** Håndkør mergesort med følgende input (lav illustrationer som CLRS figur 2.4):

   (a) $[5, 8, 3, 1, 4, 7, 2, 6]$.

   (b) $[12, 53, 13, 64, 34, 9, 21, 51]$.

2. **Køretid.** Antag at du har tre algoritmer hvis køretider er hhv. $100n$, $10n^2$ og $5n^3$. Hvor mange gange stiger køretiden hvis du fordobler inputstørrelsen n?

3. **Rekursion og fakultet.** Denne opgave omhandler en algoritme for fakultetsfunktionen

$$n! = n \cdot (n-1) \cdots 2 \cdot 1.$$

   Nedenfor er tre forsøg på at lave en rekursiv algoritme der beregner $n!$.

```
Fak1(n)
  if n == 1
    return n
  x = n * (Fak1(n) - 1)
  return x
```

```
Fak2(n)
  if n == 0
    return n
  x = n * Fak2(n - 1)
  return x
```

```
Fak3(n)
  if n == 1
    return n
  return n * Fak3(n - 1)
```

(a) Hvilke(n) af algoritmerne beregner $n!$ korrekt når n er et positivt heltal?

(b) Nedenstående algoritme tager et positivt heltal som input. Opskriv pseudokode for en iterativ variant af algoritmen.

```
Rekur(n)
  if n <= 0
    return 0
  return n + Rekur(n - 1) + 2
```

4. **Fibonacci-tal.** Definér $F_0 = 0$, $F_1 = 1$, og $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. Derved fås talfølgen

$$0, 1, 1, 2, 3, 5, 8, 13, \ldots,$$

og disse tal er kendt som *Fibonacci-tallene*.

(a) Opskriv pseudokode for en rekursiv funktion `Fibonacci(n)` der returnerer F_n.

(b) Vis at for nogle tal $k < n$, bliver det rekursive kald `Fibonacci(k)` udført mange gange, når man kalder `Fibonacci(n)`. Dette er spild af tid. Lav en ny pseudokode `FibonacciImproved(n)`, der undgår dette problem (ikke nødvendigvis rekursiv).

Ekstraopgaver

1. **2D-toppunkter.** Lad M være en $n \times n$-matrix (et 2D-array). En indgang $M[i, j]$ er et *toppunkt* hvis den ikke er mindre end dens naboer i retning N, Ø, S og V, dvs.

$$M[i, j] \geq M[i-1, j], \quad M[i, j] \geq M[i+1, j], \quad M[i, j] \geq M[i, j-1] \quad \text{og} \quad M[i, j] \geq M[i, j+1].$$

(Ligesom for 1D-toppunkter i arrays, er et tal på randen af vores matrix også et toppunkt, hvis det bare er mindst så stort som alle eksisterende naboer.) Vi er interesseret i effektive algoritmer til at finde et toppunkt i M. Løs følgende opgaver.

(a) Vis at der findes et toppunkt i enhver $n \times n$-matrix M.

(b) Giv et eksempel på en 4×4-matrix som kun har ét toppunkt.

(c) Generalisér dit eksempel ovenfor til et eksempel på en $n \times n$-matrix med kun ét toppunkt, for et vilkårligt $n \geq 1$.

(d) Giv en algoritme der tager $\Theta(n^2)$ tid til at finde et toppunkt.

(e) (***) Giv en algoritme der tager $\Theta(n \log n)$ tid. *Hint:* Start med at finde det maksimale tal i den midterste søjle og benyt det til at lave en rekursion.

(f) (***) Giv en algoritme der tager $\Theta(n)$ tid. *Hint:* Konstruér en rekursion der inddeler M i fire kvadranter.

(g) (***) Vis at enhver algoritmer til at finde et toppunkt bruger mindst $c \cdot n$ operationer i værste fald, for en konstant $c > 0$.

2. **Udvælgelse, partitionering og kviksortering.** Lad $A[0 \ldots n-1]$ være et array af heltal. Tallet med *rang* k i A er det tal der fremkommer på position k såfremt man sorterer A. *Medianen* af A er tallet i A med rang $\lfloor \frac{n-1}{2} \rfloor$. En *partitionering* af A er en opdeling af A i to arrays A_0 og A_1 således at A_0 indeholder alle tal fra A der er mindre end eller lig med medianen af A og A_1 indeholder alle tal fra A der er større end medianen af A. Antag i det følgende at du har en lineærtidsalgoritme til at finde medianen af et array. Løs følgende opgaver.

   (a) Giv en algoritme, der givet et k, finder tallet med rang k i A i $\Theta(n \log n)$ tid.

   (b) Giv en algoritme til at beregne en partitionering af A i $\Theta(n)$ tid.

   (c) (*) Giv en algoritme til at sortere A i $\Theta(n \log n)$ tid vha. rekursiv partitionering.

   (d) (**) Giv en algoritme, der givet et k, finder tallet med rang k i A i $\Theta(n)$ tid.

3. Løs CLRS problem 2-1.

Bemærkninger: Nogle opgaver er stærkt inspireret af opgaver stillet af Philip Bille og Inge Li Gørtz i kurset Algoritmer og Datastrukturer på DTU,
`http://www2.compute.dtu.dk/courses/02105+02326/2015/#generelinfo`.