Eksaminatorie-timer

DM507: Algoritmer og datastrukturer

Løsninger

Uge 6 #1 2021

Del A

Opgave 1 – Cormen et al. øvelse 2.2-3

Betragt Lineær søgning igen. Hvor mange elementer skal betragtes gennemsnitligt, når elementet der søges efter kan optræde på hvert index med lige høj sandsynlighed? Hvad med worst-case og best-case? Hvad er average-, worst- og best-case i Θ-notation?

Antal betragtede elementer:

- Best case: søgte element er det første element i input-arrayet; vi skal tjekke 1 element.
- Worst case: søgte element er det sidste element; vi skal tjekke n elementer.
- Average case: søgte element kan optræde på hvert index med lige høj sandsynlighed, så man skal gennemsnitligt betragte

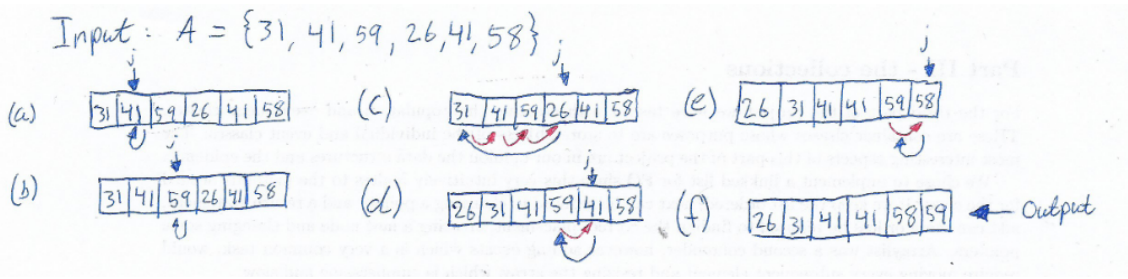
$$\frac{1+2+\cdots+n}{n} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2} \text{ elementer.}$$

I Θ-notation:

- Best case: $\Theta(1)$
- Worst case: $\Theta(n)$
- Average case: $\Theta(n)$, da $\lim_{n\to\infty} \frac{\frac{n+1}{2}}{n} = \lim_{n\to\infty} \frac{\frac{n}{n}+\frac{1}{n}}{2} = \frac{1}{2}$. (jf [1, p. 17])

Opgave 2 – Cormen et al. øvelse 2.1-1

Med Figur 2.2 som model, illustrer `Insertion-Sort` på arrayet $A = \langle 31, 41, 59, 26, 41, 58\rangle$.



Vi bytter ikke ens tal. Insertion sort er *stabil* hvilket betyder de beholder deres relative orden.

Opgave 3 – Implementer InsertionSort

Implementer InsertionSort i Java eller Python ud fra pseudo-kodenside 18 i lærebogen. Test at din kode fungerer ved at generere arrays/lister med forskelligt indhold og sortere dem.

Se `insertion_sort/insertion.py`

Opgave 4 – Cormen et al. opgave 2-4 (*) a-c

Lad $A[1 \dots n]$ være et array med n unikke tal. Hvis $i < j$ og $A[i] > A[j]$, så kaldes parret (i, j) en *inversion* for A.

a) Oprems de fem inversioner for arrayet $\langle 2, 3, 8, 6, 1 \rangle$?

(1,5), (2,5), (3,5), (4,5) og (3,4).

b) Hvilket array med elementer fra mængden $\{1, 2, \dots, n\}$ har flest inversioner. Hvor mange inversioner har det?

Det har et array, som er sorteret i faldende orden, hvor antallet af inversioner er:

$$\overbrace{(n-1)}^{\text{element at index n}} + \underbrace{(n-2)}_{\text{element at index n-1}} + \cdots + \overbrace{(n-(n+1))}^{\text{element at index 2}} = (n-1) + (n-2) + \cdots + 1 = \frac{(n-1)n}{2}$$

Her vil elementet på index n være relateret til (gennem "inversions-relationen") hvert element i intervallet $1 \dots n-1$, elementet på index $n-1$ vil være relateret til hvert element i intervallet $1 \dots n-2$ og så videre indtil elementet på index 1, som ikke er relateret til nogen elementer.

c) Hvad er forholdet mellem køretiden af insertion sort og antallet af inversioner i input arrayet?

Højt antal inversioner $\Rightarrow$ køretiden er tættere på worst-case køretiden

Hvorfor? Det indre loop skal laver flere gennemløb. Det indre loop reducerer antallet af inversioner med én, når den laver en ombytning i det indre loop. Altså, worst-case køretiden opnås, når input-arrayet er sorteret i faldende orden, da der er flest inversioner som skal fjernes.

Del B

Opgave 1 – Implementer InsertionSort

Fortsæt opgaven ovenfor med implementation af InsertionSort (tilføj noget tidstagning).

Se `insertion_sort/insertion.py`

Opgave 2 – Cormen et al. øvelse 2.3-7 (*)

Beskriv en $\Theta(n \lg n)$ algoritme, som givet mængden S med n heltal og et heltal x, afgør om $\exists y, z \in S : y + z = x$.

Der er flere korrekte algoritmer. Her er et par stykker:

- Algoritme 1: Prøv alle kombinationer
  - Korrekthed: klart korrekt.
  - Worst-case køretid er $\Theta(n^2)$ (dvs. den er ikke god nok)
- Algoritme 2: Sortér input mængde/array[1]. For hvert element $y \in S$ søg vha. *binær søgning* efter $x - y$. Hvis $x - y$ findes for et y, da returnér `True`.
  - Korrekthed: Observér $y + z = x \Leftrightarrow z = x - y$. Vi tjekker hvert y efter om værdien $x - y$ findes, så den er klart korrekt. Da vi sorterer input, så kan vi bruge binær søgning.
  - Worst-case køretid er $\Theta(n \lg n)$, da sortering tager $\Theta(n \lg n)$ og n binær søgninger tager $\Theta(n \lg n)$ tid.
- Algoritme 3: Sortér input mængde/array og lav nyt sorteret array bestående af elementer $x - y$ hvor $y \in S$. Brug modificeret merge på to arrays, og returnér `True` hvis to arrays under merge-proceduren har to ens elementer som første element.
  - Korrekthed: Observér to ens elementer vil under merge-proceduren stå forrest i de to arrays, der merges. Findes to ens elementer, så returneres `True` da $y + z = x \Leftrightarrow z = x - y$.
  - Worst-case køretid er $\Theta(n \lg n)$, da sortering tager $\Theta(n \lg n)$ og (modificeret) merge tager $\Theta(n)$ tid.

References

[1] Rolf Fagerberg. Asymptotisk analyse af algoritmers køretider. URL `https://imada.sdu.dk/~rolf/Edu/DM507/F21/asymptotiskAnalyseAfAlg.pdf`, 2021.

[1] En mængde er selvfølgelig uordnet, men når vi sorterer den betragter vi den som en liste/et array.