Diskret Matematik og Algoritmer

Aflevering 2i

Adam Frederik Ingwersen Linnemann,
GQR701
Hold 4

Datalogisk Institut
Københavns Universitet

September 19, 2016

Del 1

Antallet af inversioner kan bestemmes ved at tælle det samlede antal 'ikke-sorterede' placeringer af elementerne i A. For arrray bestående af følgende $n = 6$ elementer; [2,1,8,4,3,6] er der sammenlagt 5 inversioner.

Del 2

En generel formel for at finde det maksimale antal inversioner, k, i et array bestående af n vilkårlige elementer kan bestemmes ved:

$$k = \frac{n(n-1)}{2}$$

Dette kan bl.a. skrives på følgende måde ved iterativ pseudokode:

Algorithm 1 Maksimale antal inversioner givet n

```
function MaxInv(n)
    v = 0
    for i = 1 to n
        v = (i − 1) + v
    return v
```

Del 3

Denne delopgave kan løses på adskillige måder - både rekursivt og iterativt. Her er valgt en iterativ tilgang, der bygger på lineær søgning. Algoritmen opererer på følgende måde:

1. Positionér starten af den første løkke (i=0) ved A's 0'te element
2. Positionér herefter starten af den anden løkke foran i (j=i+1)
3. Lad j iterere fra j=1 til n over nedenstående betingelse:
   (a) Hvis værdien af det i'te element er strengt større end det j'te, da:
   (b) Denotér variablen 'inv' med sin hidtige værdi plus 1
4. Afslut ved at returnere den akkumulerede værdi for 'inv'

Algoritmen er beskrevet i et mere koncist format i pseudokoden nedenfor.

Algorithm 2 Tæl antal inversioner i A

```
function CountInv(A, n)
    inv = 0
    for i = 0 to n - 2
        for j = i+1 to n - 1
            if A[i] > A[j]
                inv = inv + 1
    return inv
```

En alternativ løsning til den 'fladpandede' lineære søgning ville være, at anvende merge-sort med et if-statement og en tælle-variabel i den afsluttende 'merge'-del.

Del 4

For at analysere algoritmens køretid, anvendes tilgangen beskrevet i CLRS 2.2. Her er vi umiddelbart kun interesserede i de led, der påvirker køretiden, når $n \to \infty$, så at sige. Af denne årsag, betragtes udelukkende de iterative led i $CountInv(A, n)$. Leddene har en multiplikativ relation. Det sidste for-loop aftager i antallet af gentagelser som funktion af i's placering, mens det første for-loop gentages $n - 1$ gange.

$$rt \approx \Theta((n-1) \cdot \frac{(n-1)((n-1)+1)}{2}) =$$
$$\Theta((n-1) \cdot \frac{n^2 - n}{2}) =$$
$$\Theta(\frac{n \cdot (n-1)^2}{2})$$

Ved at plotte de køretiden mod f.eks. n^2, ses det, at algoritmen ved store n vil opnå respektable køretider sammenlignet med to 'komplette' for-loops, som ville foretage en masse unødvendige evalueringer. For at konstruere plottet er der anvendt R-kode, som er vedlagt i bilag.

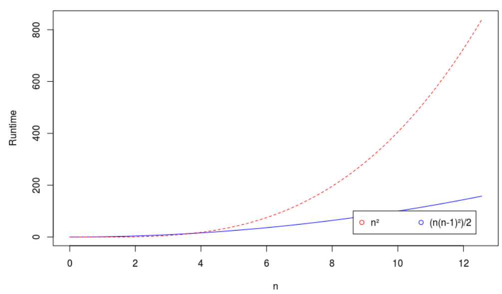

Figure 1: Algoritmens køretid sammenlignet med n^2

Bilag

```
fun1 <- function(x) x^2
fun2 <- function(x) (x*(x-1)^2)/2
x<-seq(0,4*pi,0.01)
matplot(x, cbind(fun1(x), fun2(x)),
        type = "l", col = c("blue", "red"),
        ylab = "Runtime", xlab = "n")
legend("bottomright", inset = .05, legend = c("n  ", "(n(n-1)  )/2"), pch = 1,
```