DMA aflevering 2

Bjørn Møller

18. september 2017

1 Opgave 1

Vi skal i denne opgave undersøge, hvor mange inversioner der er i i array A[0.....n-1], hvor inversioner er defineret som: A[i] > A[j] og i < j
I denne opgave er A = [2,2,8,4,3,6].
Vi kan altså se, at der må være 3 + 1 = 4 inversioner. Dette ses ved at A[2] = 8, hvilket er større end alle andre A[i].
Desuden er A[3] = 4 større end A[4] =3. I konklusion er der 4 inversioner.

2 Opgave 2

Hvor mange inversioner kan der maximalt være ved vilkårligt n?
Det maksimale antal inversioner fås ved at listen er sorteret i omvendt rækkefølge f.eks. [3,2,1]
I dette tilfælde gælder der, at der vil være n-1 + n-2 + ... +1 inversioner. Vi kender allerede til summen af n + n-1 + ... + 1
Denne sum er n*(n+1)/2. Vi kan dog se, at for at få den ønskede sum, skal vi trække n fra denne sum således:
n*(n+1)/2 - n = (n*(n+1) - 2n)/2 = n*(n-1)/2.

Således er det maksimale antal inversioner givet ved n: n*(n-1)/2

3 Opgave 3 og 4

En simpel kode, som får jobbet klaret er denne kode:
inversioner (A,n)
tæl = 0
for i = 0 to n-2
- for j = i+1 to n-1
– if A[i] > A[j]
— tæl = tæl +1 return tæl

Denne relative simple algoritme har et dobbelt for-loop,og har dermed en worst runtime på O(n). Jeg har dog overvejet en alternativ løsning. Jeg har indset, at den obenstående kode er relativ tæt på insertion sort. Dette fik mig til at overveje om man ikke kunne lave en algoritme med bedre køretid, hvor man bruger en model baseret på merge sort.

inversionerForbedret(A, p , r)
tæl = 0
if p < r
q = (p + r) / 2
- inversionerForbedret(A, p , q)
- inversionerForbedret(A, q+1 , r)
- inversionCombine (A , q ,r , p)
return tæl

```
inversionCombine ( A , q ,r , p )
n1 = q-p+1
n2 = r - q
lav nye arrays L[1...n1] og R[1..n2]
for i = 0 to n1-1
- L[i] = A[p+i-1]
for j = 0 to n2-1
- R[j] = A[q+j]
L[n1] = uendelig
R[n2] = uendelig
i , j = 0
for k = p to r
- if L[i]< = R[j]
– A[k] = L[i]
– i=i+1
- else A[k] = R[j]
– j=j+1
– if L[i] < uendelig
— tæl = tæl + 1
```

Først og fremmest, kan vi se, at denne kode er bygget på merge sort og derfor har en køretid på n* lg(n). Dette er selvfølgelig en forbedring over den tidligere kode. En anden bemærkning er, at denne også sorterer arrayet, hvilket måske ikke er eftertragtet. Dette kunne ordnes ved f.eks. at køre koden på en kopi af listen.