Programmering

# [Ex65 –](https://fronter.com/eal/links/files.phtml/59b0d9cf6a213.1733095687$277013695$/Mapper/Datamatiker+1.+studie_prcent_C3_prcent_A5r/DMU++F_prcent_C3_prcent_A6lles/DM17eX/Systemudvikling/Ex39Tir+-+Objekt-+og+Dom_prcent_C3_prcent_A6ne-modeller.docx) Køretid

|  |  |
| --- | --- |
| **Læringsmål** | * At have viden om køretid ved valg af algoritme * At kunne bestemme asymptotisk worst-case køretid for en simpel algoritme |

# Forberedelse

Brug **Ordet Rundt** og diskutér i gruppen, hvad du fik ud af forberedelsen til dagens opgave (Big O):

Hvad var let og svært at forstå?

Har I forstået materialet på samme måde?

Hvad er I enige og uenige om?

*Tidsramme: 5-10 minutter*

# Øvelse 1

Rangorden følgende O:

O(n!), O(1), O(2^n), O(log n), O (n^2), O(n log n), O (n^3)

1. O(1)
2. O(log n)
3. O(n)
4. O(nlogn)
5. O(n2)
6. O(2n)
7. O(n!)

*Tidsramme: 5 minutter*

# Øvelse 2

Naïvesort, bubblesort og insertionsort er tre forskellige sorteringsalgoritmer. Vi skal starte med at forstå hvordan disse virker. Læs forsigtigt koden på bilag 1 og diskuter for hver algoritme hvordan den virker (vælg selv passende CL-struktur). Lav noter/skitser på papir.

*Tidsramme: 30 minutter*

# Øvelse 3

I denne øvelse skal vi analysere den asymptotiske (worst-case) køretid (hvilket ofte kaldes store-O).

Hermed er nogle regler:

* En simpel operation antages at tage konstant tid: O(1)
* Antallet af inputs kaldes altid N
* Konstanter er betydningsløse og skal ikke med i en asymptotisk analyse
* Store-O refererer til worst-case køretid. Altså den maksimale køretid, lige meget hvilket input
* En løkke der looper N elementer har en asymptotisk køretid på O(N)
* En løkke der looper halvdelen af N elementer har en asymptotisk køretid på O(0.5N) = O(N)
* En ydre løkke der looper N elementer og en indre løkke der looper N elementer har O(N\*N) = O(N^2) køretid.
* Hvis en algoritme består af først en løkke, og så en løkke mere der begge gennemløber N elementer, så svarer dette til O(N+N) = O(2N) = O(N) (husk at konstanter ikke medtages)

Med ovenstående regler skal I nu bestemme den asymptotiske køretid for Naïvesort og Bubblesort.

*Tidsramme: 30 minutter*

## Øvelse 4

Følgende rekusive Fibonacci metode som har køretiden O(2^n).

public int Fibonacci(int n)

{

if (n == 0) return 0;

if (n == 1) return 1;

return Fibonacci(n - 1) + Fibonacci(n - 2);

}

Lav en ny metode Fibonacci2 som har køretiden O(n).

Udarbejd selv algoritmen.

public int Fibonacci2(int n)

{

}

Prøv og sammenligne de 2 metoder. Tag tid

|  |  |  |
| --- | --- | --- |
| N | Fibonacci | Fibonacci2 |
| 10 | 0 ms (671 ticks) |  |
| 20 | 0 ms (2236 ticks) |  |
| 30 | 64 ms (162.994 ticks) |  |
| 40 | 5990 ms (15174978 ticks) |  |
| 50 | ?? (StackOverFlow) |  |
| 60 | ?? |  |
| 70 | ?? |  |
|  |  |  |
|  |  |  |

*Tidsramme: 40 minutter*

# Bilag 1

class Sort

{

public static int[] NaïveSort(int[] x)

{

int n = x.Length;

for (int pass = 1; pass < n; pass++)

{

for (int i = 0; i < n - 1; i++)

{

if (x[i] > x[i + 1])

{

// swap

int temp = x[i];

x[i] = x[i + 1];

x[i + 1] = temp;

}

}

}

return x;

}

public static int[] BubbleSort(int[] x)

{

int n = x.Length;

for (int pass = 1; pass < n; pass++)

{

for (int i = 0; i < n - pass; i++)

{

if (x[i] > x[i + 1])

{

// swap

int temp = x[i];

x[i] = x[i + 1];

x[i + 1] = temp;

}

}

}

return x;

}

public static int[] InsertionSort(int[] x)

{

int n = x.Length;

for (int i = 1; i < n; i++)

{

int j = i;

int B = x[i];

while ((j > 0) && (x[j - 1] > B))

{

x[j] = x[j - 1];

j--;

}

x[j] = B;

}

return x;

}

}