Computersystemer 5

Lavet af: Vivek Misra

Hvad er Algoritme?

- *Finite sets of steps to solve a problem is called Algorithm.*
- *Analysis is a process of comparing two Algos, write, time, space and etc.*

En Algoritme er en beskrivelse af, hvordan man løser et problem. Hvad kommer fra en input og til en ønsket output (svar) på ens spørgsmål!

Hvis man ønsker, at fortælle andre ens problem og det kan ske gennem tryllekunst og linje for linje.

Men i vores sammenhæng skriver vi det i en program, som løser vores problem og får det løst.

Hvad er Program?

- Programmering: Er det som man gør for, at kunne få udviklet dette program. Man kan diskutere hvordan algoritmen skal være og hvordan proceduren skal være.
- Software: Er der, hvor man har kode og løsningen der er hængt sammen og det er indkodet i en computer som skal udføre det som skal til det.
- Tryllekunst: Er en måde, at beskrive det på og man kan lave en program som kan gøre lignende. Men algoritmer er et centralt begreb for hele vores verden og det kan fortælle os hvad der sker!

Hvad er Algoritme?

- *Finite sets of steps to solve a problem is called Algorithm.*
- *Analysis is a process of comparing two Algos, write, time, space and etc.*

1. Repræsentationen er den måde algoritmen skal være.
2. Analyses hvor go er den og hvor effektiv.
3. Applikationen er hvor god er den?
4. Kommunikationen er hvordan man får formidlet algoritmen

Ambigious = Sætter irrelavante operatører

Unambigous = Sætter de relevante foretegn/operatører

Betyder også tvetydig som betyder klar mening. Når der eksempelvis står at man skal lægge en variabel og vide hvor meget en er, go hvad skal man gøre når man modtager en instruktion som man ikke misforstår.

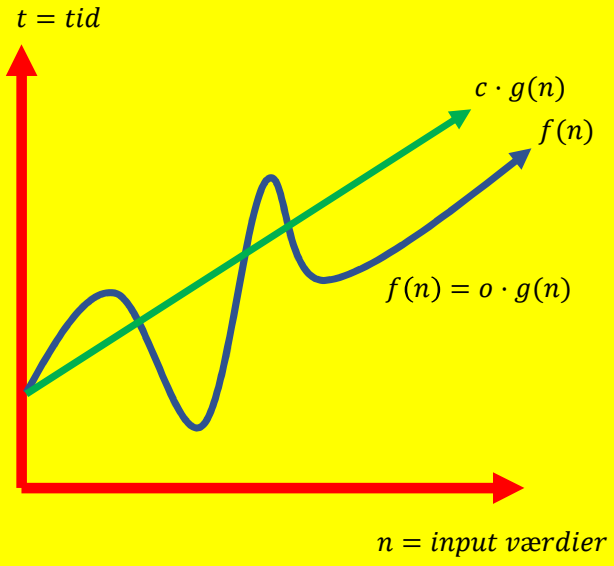
Hvad er Algorime og dets karakteristika!

- En algoritme er en lsite af mængde entydige funktioner som afgøre en afgørende proces.
  - Man kan se eksempelvis i VOLE, hvor man kan lave en rækkefølge.
  - Man kan også udføre det parallelt, hvor vi kort inde på adskillige computere og arkitekturer og hvordan grafikkort fungerede.
  - Man har også cause and effekt, det er når der sker noget så skal der ske noget lignende.
  - Ordered: Der skal være et input før jeg kan gøre noget.

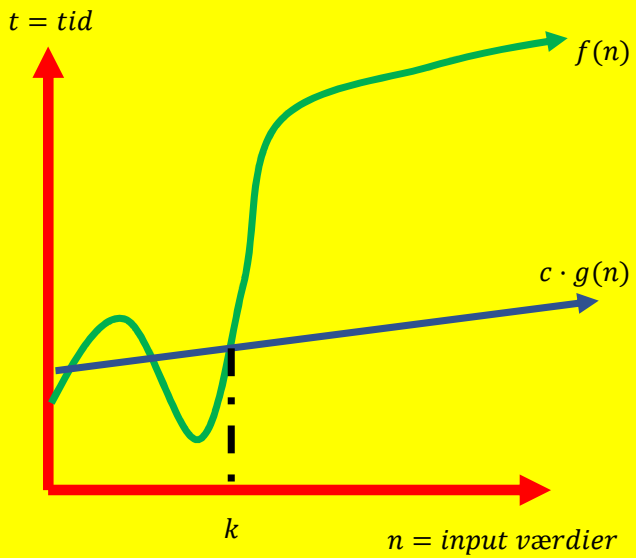
Asymptotic Notation

- Det er den Matematiske Måde, at repræsentere tidskompleksitet!
- Man prøver at analysere en Algoritmes tidseksevering uden den generelle eksekvering af algoritmen!
- Hvor mange gange kalder algoritmen sig selv?

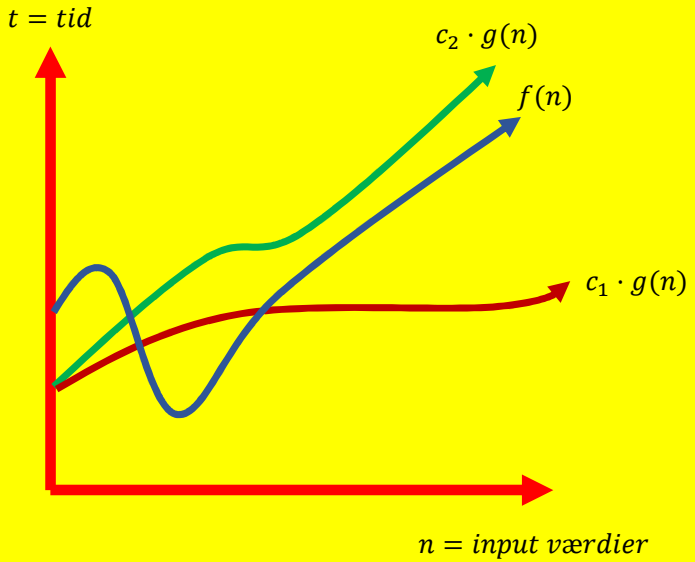
3-Eksempler på Asymptotic Notation



$f(n) \leq c \cdot g(n)$
$c > 0$
$n \geq R$
$R \geq 0$
$f(n) = 2n^2 + n$
$f(n) = o \cdot ( )$
$2n^2 + n \leq c \cdot g(n^2)$
Løsningen giver os følgende resultat!
$2n^2 + n \leq 3 \cdot n^2$
I tilfældet, kan det ses at $c\ og\ n^2$ har en forhold.
$n \leq n^2\ og\ 1 < n$
$n \geq 1$



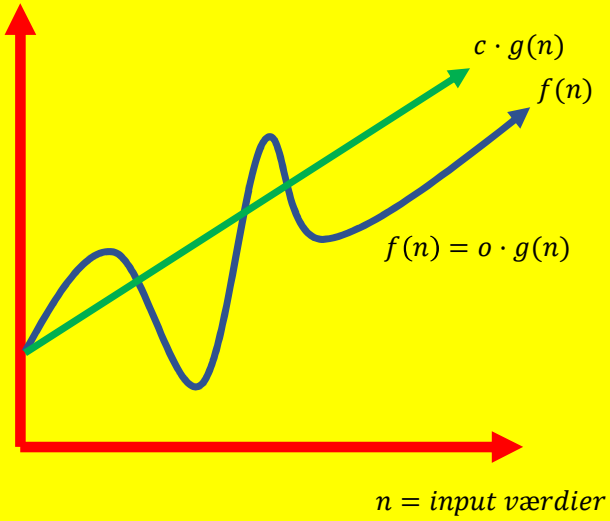
$f(n) = \Omega \cdot g(n)$
$f(n) \geq \Omega \cdot g(n)$
Løsningen giver os følgende resultat!
$2n^2 + n \geq c \cdot n^2$

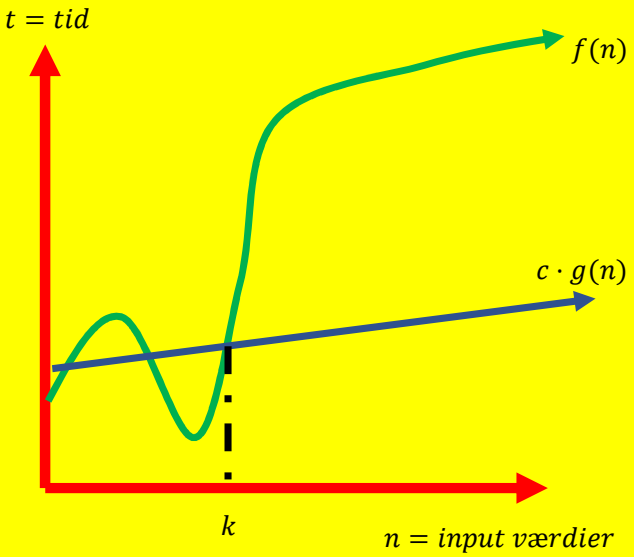


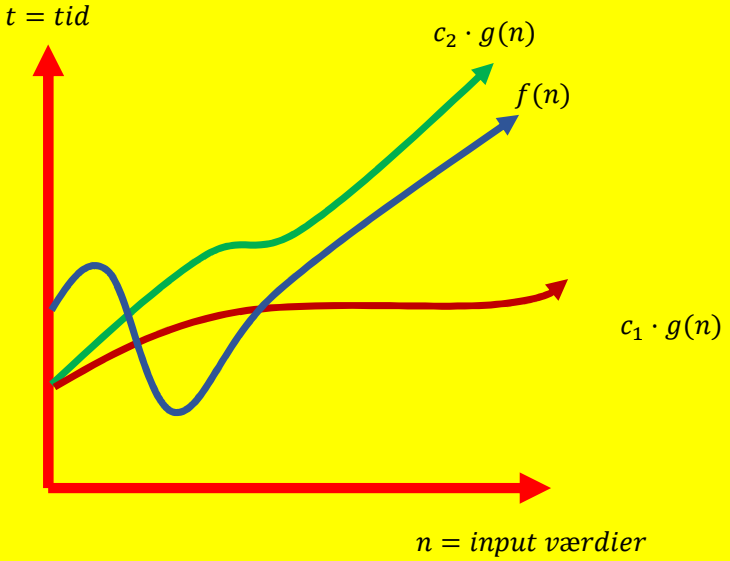
$g(n) \leq f(n) \leq c_2 \cdot g(n)$
Løsningen giver os følgende resultat!
$2n^2 \leq 2n^2 + n \leq 3n^2$

Udregningseksempler

Big (O)	Big Omega (Ω)	Thea (θ)
$2n^2 + n \leq 3 \cdot n^2$	$2n^2 + n \geq c \cdot n^2$	$2n^2 \leq 2n^2 + n \leq 3n^2$
Eksempel med n=5	Eksempel med n=2 og c=2	Eksempel med n=2
$2 * 5^2 + 5 \leq 3 \cdot 5^2$	$2 \cdot 2^2 + 2 \geq 2 \cdot 2^2$	$2 \cdot 2^2 \leq 2 \cdot 2^2 + 2 \leq 3 \cdot 2^2$
$55 \leq 75$	$10 \geq 8$	$8 \leq 10 \leq 12$







Forskellige egenskaber af Asymptotic Notation

De 5 notationer!	Refleksive	Symmetrisk	Transitive
	Refleksive er der hvor vi tjekker en funktion, eller en værdi passer den selv! $a = a$?	Hvis det ene er større end det andet, eller omvendt! $a > b$ $b > a$	Hvis det ene er større end det andet, så betyder det at den første ting er mindre end den sidste ting! $a \leq b \leq c$ $a \leq c$
Big (O); $f(n) \leq c \cdot g(n),\ a \leq b$	Yes	No	Yes
Big Omega (**Ω**); $f(n) \geq c \cdot g(n), a \geq b$	Yes	No	Yes
Theta (θ); $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$	Yes	Yes	Yes
Small (o); $f(n) < c \cdot g(n),\ a < b$	No	No	Yes
Big (o); $f(n) > c \cdot g(n),\ a > b$	No	No	Yes

Search Algoritmer!

Search	Beskrivelse
Binary Search $-\boldsymbol{log_2 n}$	Går igennem stamtræ/data fra midten og leder efter et destinationstal
Sekvential Search $-\boldsymbol{o(n)}$	Går igennem en usorteret liste fra starten og itererer igennem Listen!
QuckSort $-\boldsymbol{o} \cdot (\boldsymbol{n} \cdot \boldsymbol{logn})$	I bedste tilfælde er kompleksiteten $-\boldsymbol{o} \cdot (\boldsymbol{n} \cdot \boldsymbol{logn})$ men kigger man nærmere ændrer den sig til $\boldsymbol{o} \cdot (\boldsymbol{n^2})$ ved worst case. Eksempel er, at når man går igennem listen, skal man implementere $(\boldsymbol{n-1})$
MergeSort $-\boldsymbol{o} \cdot (\boldsymbol{n} \cdot \boldsymbol{logn})$	Det er den i bedste tilfælde, som bruges som Formel.
InsertionSort $-\boldsymbol{o} \cdot (\boldsymbol{n^2})$	I bedste tilfælde er det o(n) men i average og worst case bliver det $\boldsymbol{o} \cdot (\boldsymbol{n^2})$, men i bedste tilfælde er det o(n).
BubbleSort	I alle cases af kompleksistet anvendes formlen $\boldsymbol{o} \cdot (\boldsymbol{n^2})$
HeapSort $-\boldsymbol{o} \cdot (\boldsymbol{n} \cdot \boldsymbol{logn})$	Vi bruger datastrukturer, hvor vi har den maksimale element og den minimale element. Vi bruger $\boldsymbol{o} \cdot (\boldsymbol{n} \cdot \boldsymbol{logn})$ i alle tilfælde!
Sekvential Search $-\boldsymbol{o(n^2)}$	Her bruges den samme formel i alle tilfælde.
Complete Bindry Tree $-\boldsymbol{o} \cdot (\boldsymbol{logn})$	Det betyder at man har en stamtræ hvor alle noder findes på samme Level. vi plejer altid at gå igen nem imod Venstre og så højre . husk altid at ende betyder antallet af elementer.

Search Algoritmer!

Search	Beskrivelse
Insertion in Heap $(\boldsymbol{logn})$	Hvis man indsætter en element/value inde i et heap, så tager det tid af Ordre af 1. O(1). Derefter skal man arrangere tingene. Ma forsøger at tage den maksimale element i ruten eller ved rødderne.
Construct Heap $(\boldsymbol{n \cdot logn})$	Hvis man ønsker, at konstruere en Heap, hvor meget tid kommer det til at tage? Vi så, at ved at sætte en element i kompleksiteten logn, men for at sætte flere elementer bliver det n*logn.
Delete fra Heap $\boldsymbol{log_2 n}$	Er der hvor vi sletter. Element fra rødder og sætter nyt på. Det er derfor man siger $\boldsymbol{log_2 n}$
Huffmann Coding $\boldsymbol{log_2 n}$	Der er ikke så meget tidskompleksitet
Primes Matrix $\boldsymbol{o(n^2)}$ Krisball	Er der hvor vi finder den korteste afstand mellem tingene. Hvis Matrix bruges er det $\boldsymbol{o(n^2)}$, men er det Heaps under Prins, så bliver det: $\boldsymbol{o[(V+E) \cdot log(V)]}$
DFS og BFS	Breadth for Search, de har kompleksiteten $\boldsymbol{O \cdot (V+E)}$
All-Pair-Shortest $\boldsymbol{o(n^3)}$	Vi plejer, at snakke om den korteste form ved enhver par. FLOYD-VARSEL's METODE

Search Algoritmer!

Search	Beskrivelse
DIJKSTRA-Algoritme	Er der hvor vi bruger dem til single-source, eller single pair shortest path. Det betyder, at man ønsker at gå fra en Element til en ønsket Destination. Den minimale tid eller path til den ønskede sted er: $o \cdot (v^2)$

Rekursive

- Her går det ud på at inddele problemet oppe i flere små dele. I denne sammenhæng har man en funktion power, hvor man siger at x i y'en er det samme som at tage 5 gange 5 i anden.
- Hvis y er eksempel i lige med 0, så skal man ikke gøre mere og derved returnere 1.
- Ellers skal man bare returnere x gange med opløftet i x, og y-1.

- Rekursive Strukturer bliver ved med, at kalde sig selv. Man kan ikke nøjes med, at have en lykke fordi ellers bliver den ved med at køre og derved stopper den ikke og bruger computerens memory celler.
- Derfor er det vigtigt, at komme ud af lykke hvor man kalder på sig selv flere gange, med andre ord skal der være en stop proces/funktion.

Rekursive

- I den binære liste, ønsker vi at have sorteret listen på forhånd. Man kan se, at vi leder efter en værdi i en liste, hvorefter man leder efter Larry i den mindre liste og derved leder man efter John i en endnu mindre liste. (Kig på Slide 15 fra Mortens Præsentation!).
- Det samme procedure foregår i de næste 3 slides.

- Man har også denne her ringe spil, hvor man bygger på at man gøre tingene mindre ved at sætte ring ovenpå i hinanden. Hvis man eksempelvis ønsker, at rykke alle ringene over til C, så kan det ses at vi starter med at rykke de 3 ting over til B og derved rykke den røde over til C. Til sidst kan vi se, at vi har rykke de 3 ringe oven på den røde fra B til C.

- Man kan se, at man efterfølgende kan se kodet på Python, at hvis man har man følgende kode.

Computersystemer 5

Lavet af: Vivek Misra

Håber I kan lide det ☺