AlgDat 12

Forelesning 8
Algoritmeanalyse

Gunnar Misund

Mål

- Hvor effektiv er en gitt algoritme? To hovedaspekter:
  - **Tidsforbruk**
    Hvor lang tid tar en algoritme på å løse et problem i forhold til en annen algoritme?
  - **Plassforbruk**
    Hvor mye internminne (evt. også eksternminne, dvs. disk etc.) bruker algoritmen i prosessen med å løse et problem av en viss størrelse?
- I begge tilfellene er vi opptatt av hvordan tid- og plassforbruket øker når vi øker størrelsen på problemet algoritmene skal løse.
- Problemets størrelse, eller størrelsen på input, uttrykkes ofte ved parameteren N, som er et positivt heltall

Worst case

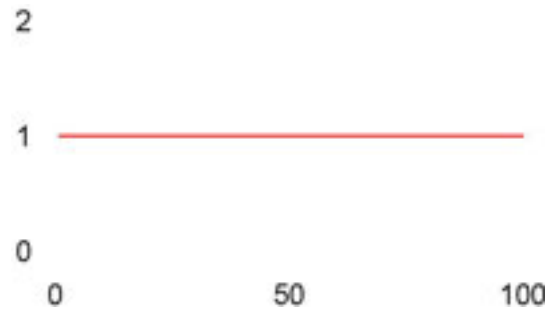
- I algoritmeanalysen er vi generelt ikke interessert i å vite nøyaktig hvor mye tid en algoritme bruker, men prøver heller å angi hvilken størrelsesorden løsningen ligger i.
- Det er i praksis to måter å tilnærme seg algoritmeanalysen.
  - Enten er vi interessert i hvordan algoritmen oppfører seg i en "normal", eller "gjennomsnittlig" situasjon,
  - eller så er vi interessert i hvor effektiv algoritmen er for de "verste tilfeller" av input.
  - I det siste tilfellet ønsker vi en slags garanti for at en metode ikke overstiger en viss øvre grense når det gjelder bruk av tid eller minne. Det er denne typen analyse ("worst case") som er den vanligste, og den som er den enkleste å utføre.

Big-Oh

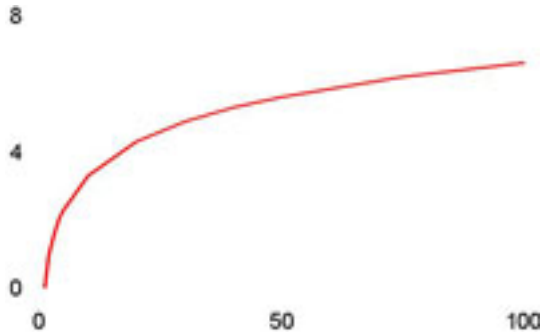
- "Store O"-notasjon (Big-Oh) er en grov måte å angi tids- og plassforbruk på.
- **Definisjon Big-Oh:**
  - Vi sier at T(N) er O(F(N))
  - hvis det finnes positive konstanter *c* og N_0
  - slik at
  - T(N) <= *c*F(N) når N >= N_0.

Viktige funksjonsklasser

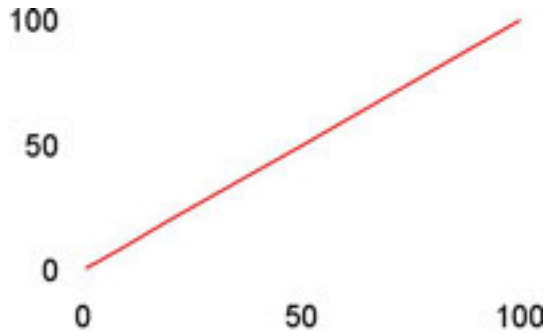
Konstant O(1)



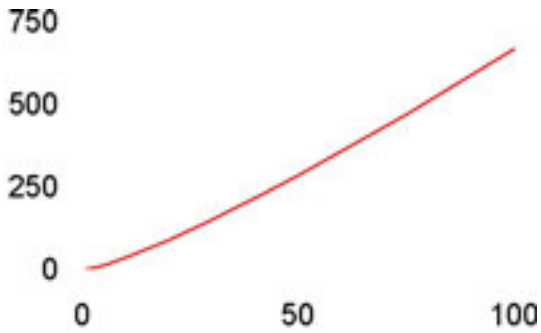
Logaritmisk O(logN)



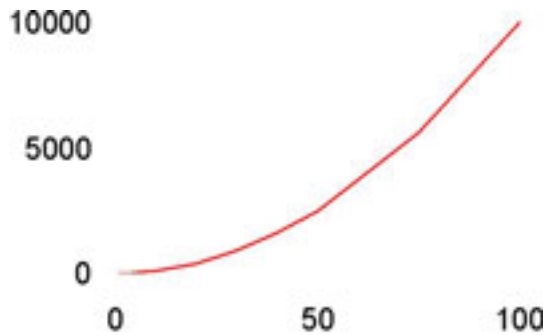
Lineær O(N)



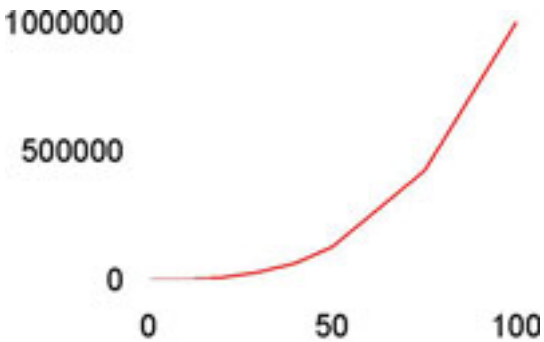
Loglineær O(NlogN)



Kvadratisk $O(N^2)$



Kubisk $O(N^3)$



Gunnar Misund

Eksponensial- og Logaritmefunksjoner

- Eksponentialfunksjoner:
  - $f(x) = a^x$, a: grunntall/base
  - $f(x) = 2^x$:
    - 2 * 2 *...*2, x ganger
    - $f(3) = 2^3 = 8$
- Logaritmefunksjoner (inverse eksponentialfunksjoner)
  - $f(x) = \log_a x$, a: grunntall/base
    - $\log_2 8 = 3$, fordi 2*2*2 = 8, eller fordi: 8/2/2/2 = 1

      (antallet ganger vi kan halvere 8 for å få 1
  - Sagt på en annen måte: $\log_a a^x = x$
  - $\log_2 8 = \log_2 2^3 = 3$

Eksempler

- F(N) = 3N - 42 => O(N)
  - Velg f.eks c = 4: F(N) < 4N, for alle N
- F(N) = 1000N + 3 => O(N)
  - Velg f.eks c = 1010: F(N) < 1010N, for alle N

O-analyse i praksis, I

$F(N) = 15N^2 + 6N + 8\log N$

1. Stryk alle ledd med lavere grad (orden) enn det med høyest grad(jfr funksjonsklassene på forrige slide)
   - $F(N) = 15N^2 + 6N + 8\log N$
2. Stryk den konstante faktoren i leddet med høyest grad
   - $F(N) = 15N^2 + 6N + 8\log N$
3. Dette gir Big-Oh orden
   - $O(N^2)$

O-analyse i praksis, II

```
public void selectionSort(int[] arr) {
    int min = 0;
    for (int i = 0; i < arr.length; i++) {
        min = pass;
        for (int j = i; j < arr.length; j++) {
            if (arr[j] < arr[min]) {
                min = j;
            }
        }
        swap(arr, i, min);
    }
}
```

1. Finn kodebiten som blir utført flest ganger (som regel: studer løkkene!)
2. Finn ut hvor mange ganger den blir utført (i verste tilfelle) i forhold til størrelse på input
3. Bruk "reglene" for å finne funksjonsklassen dette tilsvarer

O-analyse i praksis, III

- Mål kjøretider for et passende sett med økende input
- Gjett på hvilken orden algoritmen har, la oss si $O(N^2)$
- For hver måling T(N), del med "ordensfunksjonen", i vårt eksempel $T(N)/N^2$
- Plott dette resultatet; Hvis vi får ut en tinærmet horisontal graf, er hypotese korrekt. Hvis ikke, prøv med en annen ordensfunksjon