# TDDC91,TDDE22,725G97 Lektion 2
Sortering

Magnus Nielsen
magnus.nielsen@liu.se

25 september 2018

## Utomhus - Första 45

Genomgång av algoritmen för ***insertionsort***:
Den första positionen i vår osorterade datamängd anses vara sorterad. Vi tittar på positionen efter, och om det är mindre än föregående elementet swappar vi plats på dem, annars lämnar vi dem. Efter detta steg består vår sorterade del av två element.
Vi fortsätter på nästa position, och swappar ner det elementet till rätt position i den sorterade delen av vår datamängd. Öka återigen på positionen, och reptera föregående steg. Upprepa tills hela datamängden är sorterad.

**Namnförklaring**: Vi stoppar in ett element i taget i den sorterade delen av datamängden (insertion).

Låt studenterna ställa sig på ett led. Hjälp dem gärna "slumpa" sig lite extra vid behov. Behåll gärna en slumpmässigt utvald student utanför "arrayen" som assistent. Låt dem nu, med hjälp av din asssistent, sortera sig efter denna algoritm baserat på något kriterie (längd, skostorlek, etc : **ytterligare förslag mottages tacksamt**). Håll dig själv utanför själva sorteringen så att studenterna själva får aktivt deltaga, samt att du kan hjälpa till att korrigera eventuella fel, eller om problem uppstår. Om de behöver hjälp igång ska de självfallet få det, men de bör själva göra arbetet till största delen.

Diskussion: Efter sorteringen, ge studenterna några minuter att diskutera vad tidskomplexiteten kan tänkas vara (och be om motivation, även om någon direkt kommer fram till rätt svar!).
Svar:
$O(n^2)$ då vi itererar hela datamängden (n element, förvisso med -1 för varje inre iteration, men konstanter är inte intressanta) n gånger.
$\Omega(n^2)$ då vi fortfarande måste utföra samtliga iterationer, även på en helt sorterad datamängd.
Alltså: $\Theta(n^2)$

Genomgång av algoritmen för ***selectionsort***:
Vi startar på första positionen i datamängden vi avser sortera, och sätter det som minsta element. Vi söker igenom hela den resterande datamängden och jämför element för element, om elementet vi tittar på är mindre än det som för närvarande är minst, så sparar vi det nya istället.
När vi tagit oss igenom hela datamängden utför vi en swap på det första och det minsta elementet.

Vi flyttar fram ett steg i datamängden, och utför samma sak som tidigare. Denna process repeterar vi tills datamängden är sorterad.

**Namnförklaring**: Vi väljer ut ett element (selection), och stoppar in på nuvarande position.

Likt de tidigare algoritmerna, låt studenterna få sortera sig och erbjud hjälp vid behov. Blanda upp ordningen, eller sortera efter ett nytt kreterium.

Diskussion: Efter sorteringen, ge studenterna några minuter att diskutera vad tidskomplexiteten kan tänkas vara (och be om motivation, även om någon direkt kommer fram till rätt svar!).
Svar:
$O(n^2)$ då vi itererar hela datamängden (n element, förvisso med -1 för varje inre iteration, men konstanter är inte intressanta) n gånger.
$\Omega(n^2)$ då vi fortfarande måste utföra samtliga iterationer, även på en helt sorterad datamängd.
Alltså: $\Theta(n^2)$

# Inomhus - Första 45

Genomgång av algoritmen för ***insertionsort***:
Den första positionen i vår osorterade datamängd anses vara sorterad. Vi tittar på positionen efter, och om det är mindre än föregående elementet swappar vi plats på dem, annars lämnar vi dem. Efter detta steg består vår sorterade del av två element.
Vi fortsätter på nästa position, och swappar ner det elementet till rätt position i den sorterade delen av vår datamängd. Öka återigen på positionen, och reptera föregående steg. Upprepa tills hela datamängden är sorterad.

**Namnförklaring**: Vi stoppar in ett element i taget i den sorterade delen av datamängden (insertion).

Diskussion: Efter sorteringen, ge studenterna några minuter att diskutera vad tidskomplexiteten kan tänkas vara (och be om motivation, även om någon direkt kommer fram till rätt svar!).
Svar:
$O(n^2)$ då vi itererar hela datamängden (n element, förvisso med -1 för varje inre iteration, men konstanter är inte intressanta) n gånger.
$\Omega(n^2)$ då vi fortfarande måste utföra samtliga iterationer, även på en helt sorterad datamängd.
Alltså: $\Theta(n^2)$

Genomgång av algoritmen för ***selectionsort***:
Vi startar på första positionen i datamängden vi avser sortera, och sätter det som minsta element. Vi söker igenom hela den resterande datamängden och jämför element för element, om elementet vi tittar på är mindre än det som för närvarande är minst, så sparar vi det nya istället.
När vi tagit oss igenom hela datamängden utför vi en swap på det första och det minsta elementet.

Vi flyttar fram ett steg i datamängden, och utför samma sak som tidigare. Denna process repeterar vi tills datamängden är sorterad.

**Namnförklaring**: Vi väljer ut ett element (selection), och stoppar in på nuvarande position.

Med hjälp av **placeholder**, demonstrera visuellt i lugnt takt och förklara varje steg av sorteringen.

Diskussion: Efter sorteringen, ge dem några minuter att diskutera vad tidskomplexiteten kan tänkas vara (och be gärna om motivation!).
Svar:
$O(n^2)$ då vi itererar hela datamängden (n element, förvisso med -1 för varje inre iteration, men konstanter är inte intressanta) n gånger.
$\Omega(n^2)$ då vi fortfarande måste utföra samtliga iterationer, även på en helt sorterad datamängd.
Alltså: $\Theta(n^2)$

## Svar till lektionsuppgifterna.

1. Insertion sort
2. Bubble sort ("uppbubbling" av största elementet)
3. Bubble sort ("nedbubbling" av minsta elementet)
4. Selection sort
5. Quick sort
6. Vi måste räkna lite:
   (a) Vi har $10^7$ element, och ska göra 1000 fullständiga traverseringar av listan. Det ger oss ungefär $1000 \times 10^7 = 10^{10} = 10 miljarder$ operationer
   (b) Konverteringen kostar oss ca $10^7$ operationer.
   I datavetenskapen refererar logaritmuttrycket till $bas_2$ logaritmer. Vi har då:
   $10^7 \times \log_2(10^7) \approx 2.3 \times 10^8$ operationer för att sortera arrayen.
   Uppskattat antal operationer blir ungefär: $2.3 \times 10^8 + 10^7 = 240 miljoner$.
   Anledningen till att vi kan bortse från kostnaden för att plocka fram de 1000 största (dyraste) elementen i arrayen är att de kommer vara från slutet av arrayen, och således behöver vi göra 1000 operationer för att plocka fram dem. 1000 operationer är försumbara i sammanhanget.

   Vad har vi lärt oss? Att inte stirra oss blinda på tiskomplexiteten, vi måste även överväga användningsområdet.