Insticks- och samsortering

Tom Smedsaas

Två sorteringsalgoritmer:
inštickssortering och samsortering
(mergesort)

Sortering

Vanliga algoritmer att ta upp är

- Insticksortering
- Urvalssortering
- Bubbelsortering

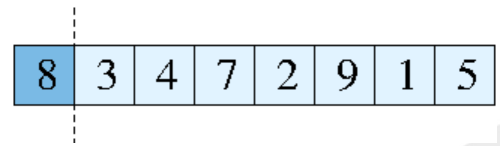
Egenskaper:

- Enkla att förstå
- Enkla att programmera
- Alla $\Theta(n^2)$ i genomsnitt

Insticksortering

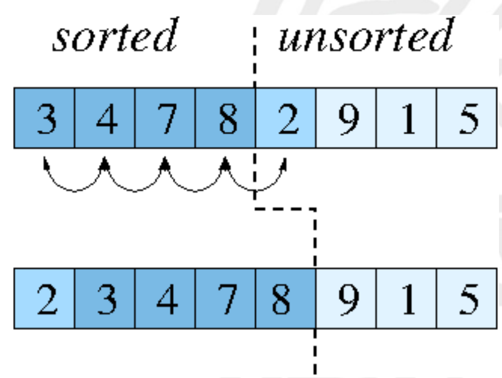
Ett vanligt sätt att beskriv insticksorteringen är att säga att listan består av två delar – en sorterad och en osorterad.

Från början innehåller den sorterade delen bara det första elementet:



För varje steg utvidgas den sorterade delen med det som står först i den osorterade delen

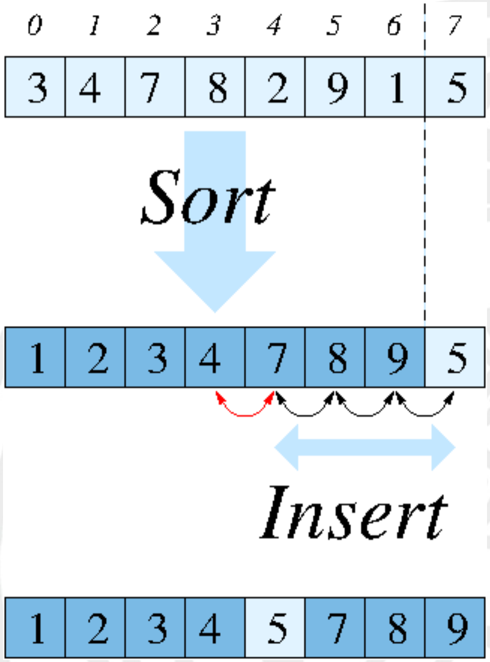
Så här kan det se ut vid den fjärde utvidgningen:



Insticksssortering rekursivt

För att sortera en lista med n element sorterar vi först de $n-1$ första elementen varefter vi infogar det sista elementet så att sorteringen behålls.

```
def ins_sort(lst, n):
    if n <= 1:
        return
    ins_sort(lst, n-1)
    i = n-1
    while i>0 and lst[i] < lst[i-1]:
        lst[i-1], lst[i] = \
            lst[i], lst[i-1]
        i -= 1
```



Instickssortering – bästa fall

```
def ins_sort(lst, n):
    if n <= 1:
        return
    ins_sort(lst, n-1)
    i = n-1
    while i>0 and lst[i] < lst[i-1]:
        lst[i-1], lst[i] = lst[i], lst[i-1]
        i -= 1
```

Låt $t(n)$ vara antalet gånger while-villkoret beräknas.

I *bästa* fall (om värdena read är sorterade):

$$t(n) = t(n-1) + 1 = (t(n-2) + 1) + 1 = \cdots = n$$

Instickssortering – värsta fall

```
def ins_sort(lst, n):
    if n <= 1:
        return
    ins_sort(lst, n-1)
    i = n-1
    while i>0 and lst[i] < lst[i-1]:
        lst[i-1], lst[i] = lst[i], lst[i-1]
        i -= 1
```

I *värsta* fall (om sorterat i omvänd ordning):

$$t(n) = t(n-1) + n \quad = \big(t(n-2) + (n-1)\big) + n \quad = \cdots$$

$$= 1 + 2 + \cdots + n = \frac{n \cdot (n+1)}{2}$$

Instickssorteringen – genomsnitt

```
def ins_sort(lst, n):
    if n <= 1:
        return
    ins_sort(lst, n-1)
    i = n-1
    while i>0 and lst[i] < lst[i-1]:
        lst[i-1], lst[i] = lst[i], lst[i-1]
        i -= 1
```

I *genomsnitt* tänker vi oss att elementen reser halva vägen:

$$t(n) = t(n-1) + n/2 \ = (t(n-2) + (n-1)/2) + n/2 \ = \cdots \ = \frac{n \cdot (n+1)}{4}$$

Således $\Theta(n^2)$ både i genomsnitt och i värsta fall.

Balansering av algoritmen

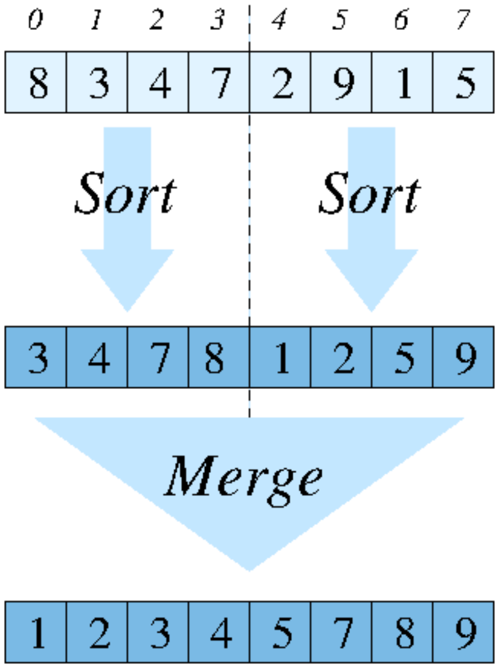
I stället för att infoga elementen ett och ett kan man ta flera åt gången.

8	3	4	7	2	9	1	5

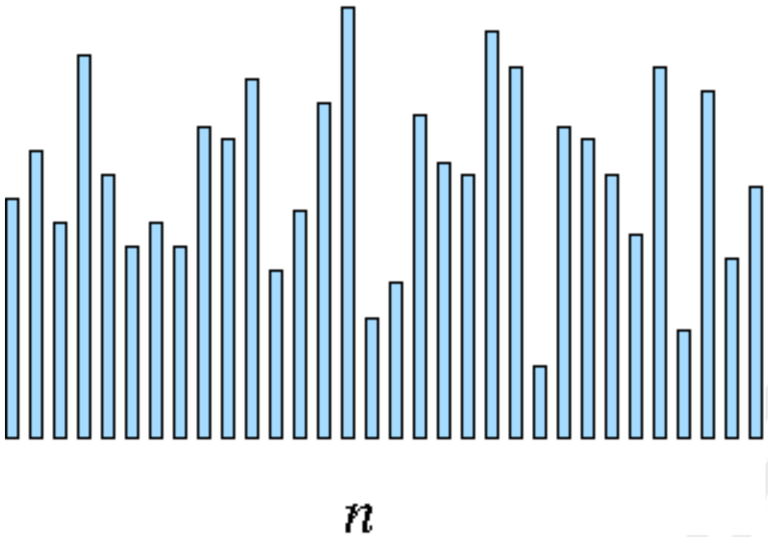
Bäst blir det om man delar i mitten:

1. Dela mängden i två lika stora delar.
2. Sortera delarna var för sig.
3. Sammanfoga delarna.

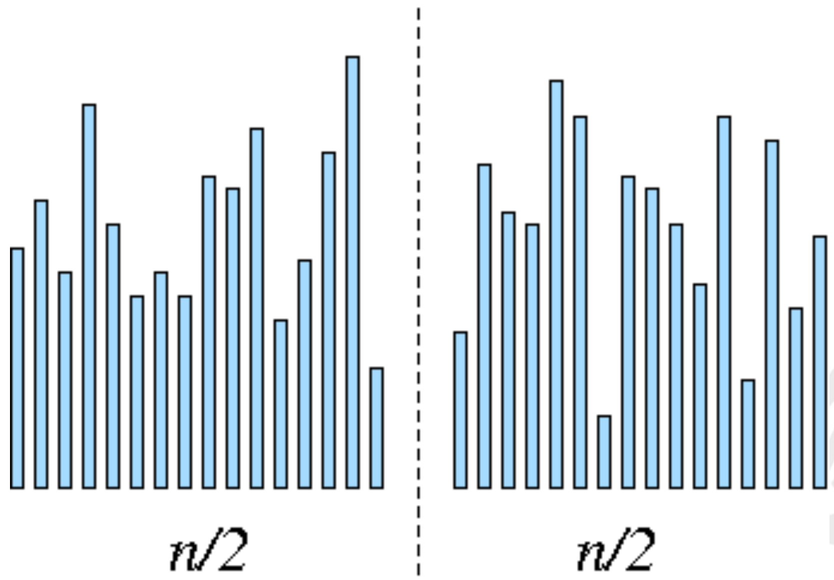
Mergesort



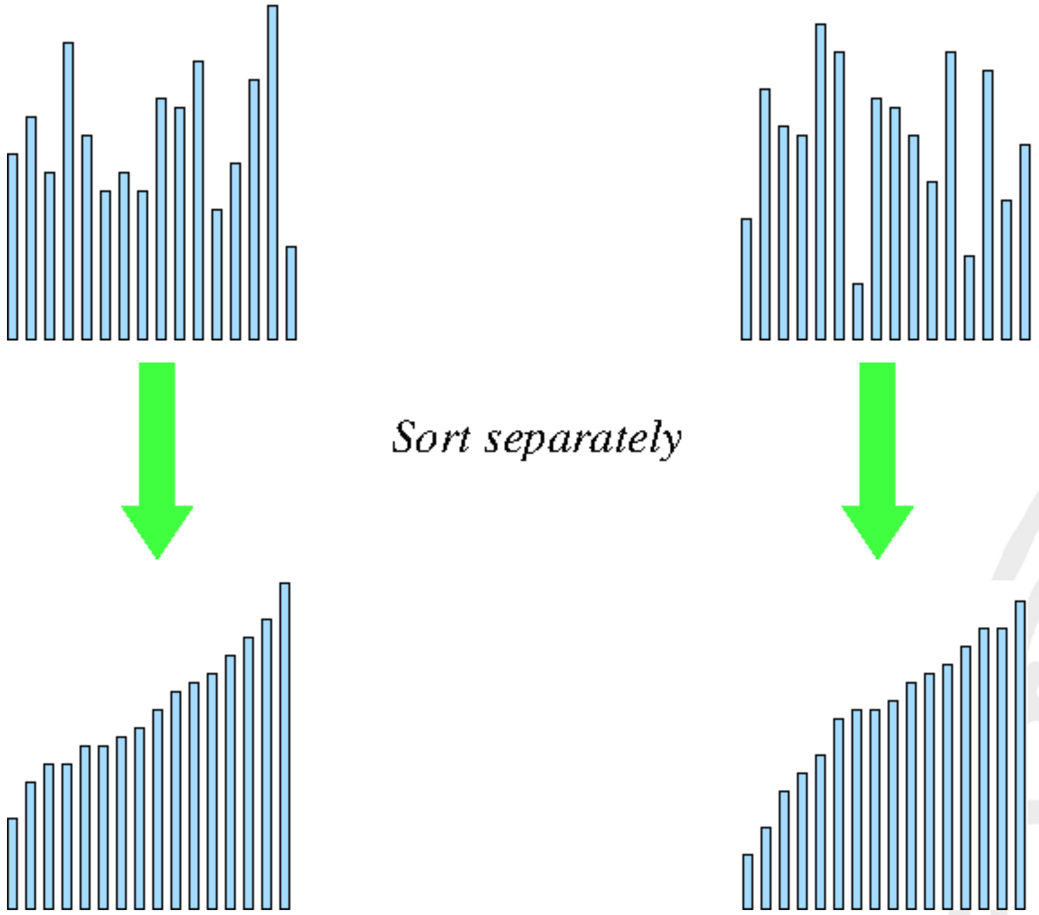
Mergesort - illustration



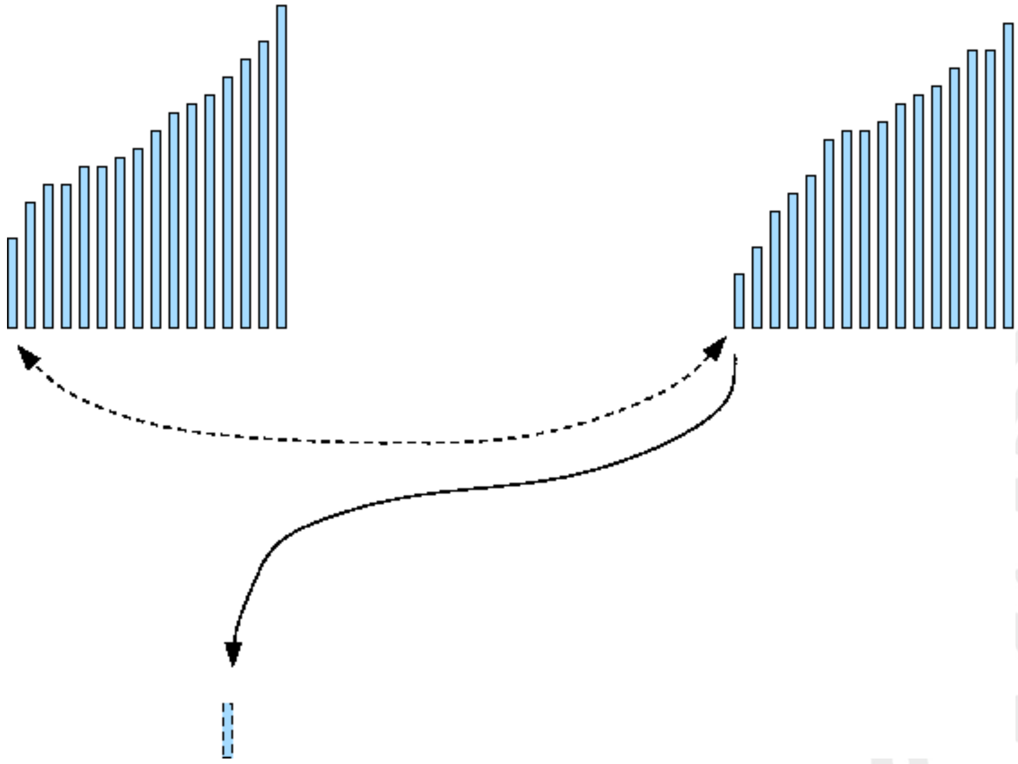
Mergesort-illustration



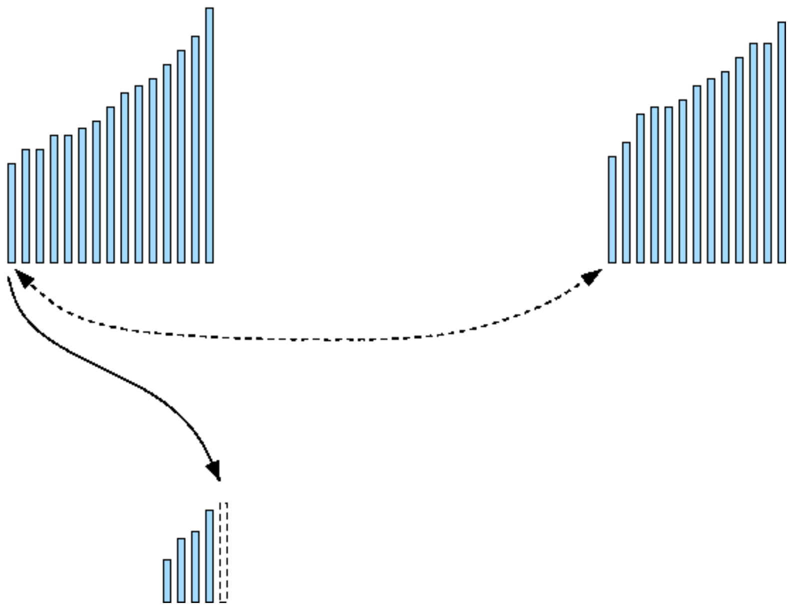
Mergesort-illustration



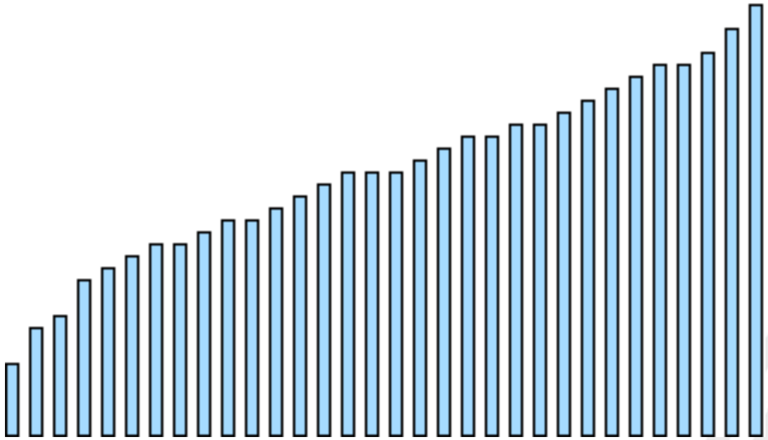
Mergesort-illustration



Mergesort-illustration



Mergesort-illustration



Analys av mergesort

Vi löser ett problem av storlek n genom att lösa *två* problem av storlek $n/2$ samt en sammanfogning av de två lösningarna.

Tiden för att sammanfoga de två lösningarna är proportionell mot n.

Låt $t(n)$ vara tiden det tar att sortera n element. Då gäller

$$t(n) = \begin{cases} c & \text{om } n = 0 \\ 2t(n/2) + d \cdot n & \text{om } n > 0 \end{cases}$$

där c och d är obekanta konstanter.

Analys av mergesort

Om n är en jämn 2-potens, $n = 2^k$, så gäller

$$
\begin{aligned}
t(n) &= 2t(n/2) + dn = \\
&= 2\left(2t(n/4) + \frac{dn}{2}\right) + dn = \\
&= 4t(n/4) + dn + dn = \\
&= 2^k t(n/2^k) + kdn = nt(1) + dn\log_2 n
\end{aligned}
$$

Således är algoritmen komplexitet $\Theta(n \log n)$.

Detta gäller även om n inte är en jämn tvåpotens.

Två frågor

- Antag att det tar 1 sekund att sortera 10^3 tal den enkla insticksorteringen. Hur lång tid kommer det då att ta att sortera 10^6 tal?
- Samma fråga för mergesort.

UPPSALA UNIVERSITET

The end