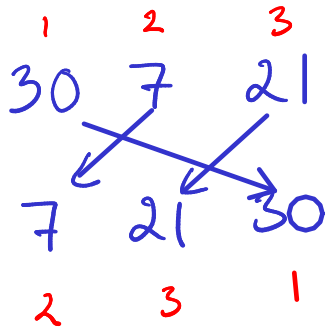
Sortering i lineær tid

(under visse omstændigheder)

og en nedre grænse for sammenligningssortering

Nedre grænse for sammenligningsbaseret sortering

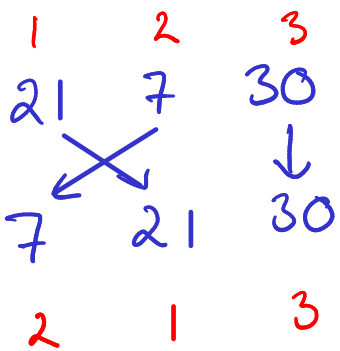
Nedre grænse for *alle* sorteringsalgoritmer. Kræver en præcis definition af sorteringsalgoritme.



$30 > 7 \quad 21$ — 1 : 2

$7 \quad 30 > 21$ — 1 : 3

$7 \leq 21 \quad 30$ — 2 : 3



$21 > 7 \quad 30$ — 1 : 2

$7 \quad 21 \leq 30$ — 1 : 3

Nedre grænse for sammenligningsbaseret sortering

Nedre grænse for *alle* sorteringsalgoritmer. Kræver en præcis definition af sorteringsalgoritme.

Sammenligningbaseret: elementer kan sammenlignes med andre elementer, men ikke deltage i andre operationer.

- Grundlæggende handling: sammenligning af to elementer i input.
- Grundlæggende svar: opstilling som skal laves for at få sorteret orden. (permutation af input)
- ID for elementer: deres *oprindelige* position (index) i input.

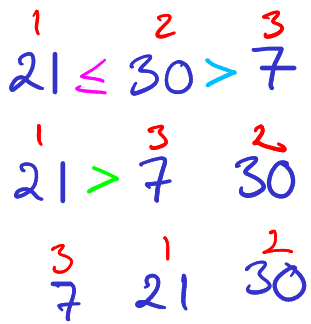
Bemærk: hvis vi starter med at annotere alle input-elementer med deres oprindelige position, kan vi i en konkret algoritme altid følge med i, hvilke to ID'er, som sammenlignes.

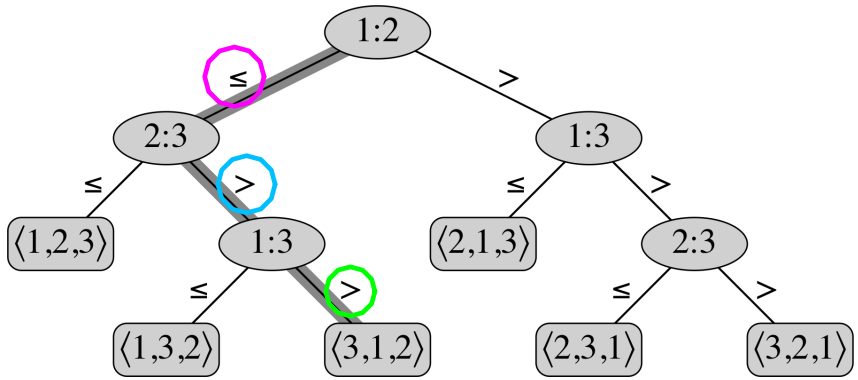
Annotering af input:

$$51, 27, 99, 61, 18, 37, \cdots \rightarrow (51, 1), (27, 2), (99, 3), (61, 4), (18, 5), (37, 6), \ldots$$

Nedre grænse for sammenligningsbaseret sortering

Præcis model som definerer begrebet "sammenligningsbaserede sorteringsalgoritmer":





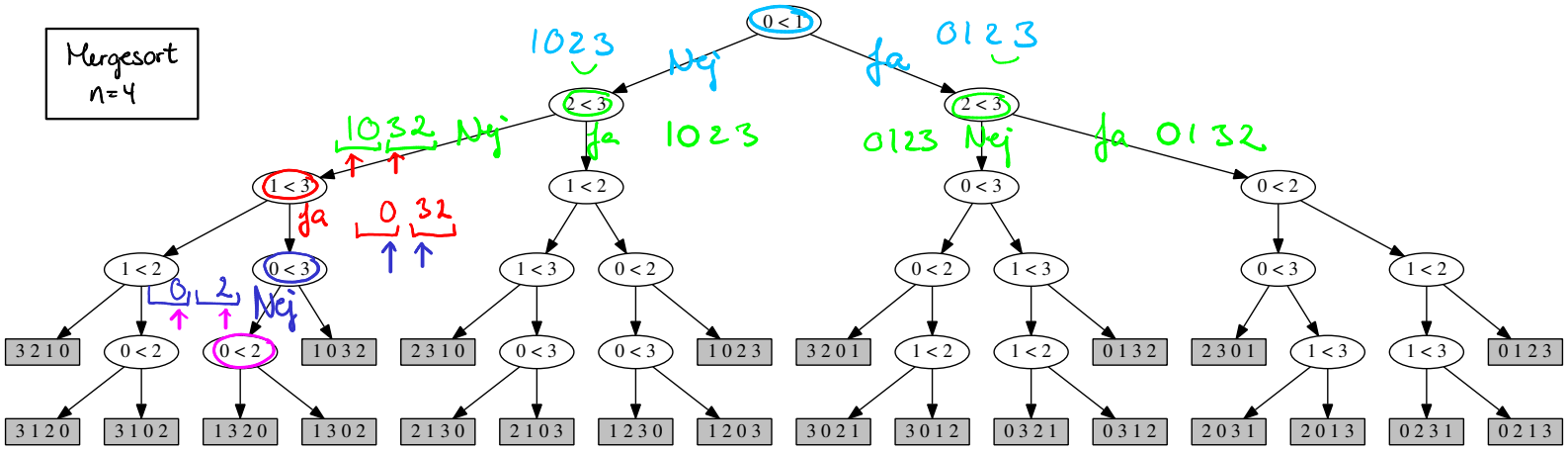
6 blade, da der er 6 permutationer af 3 elementer

Labels for indre knuder: ID'er (dvs. oprindelige indeks i input) for to input-elementer, som sammenlignes.

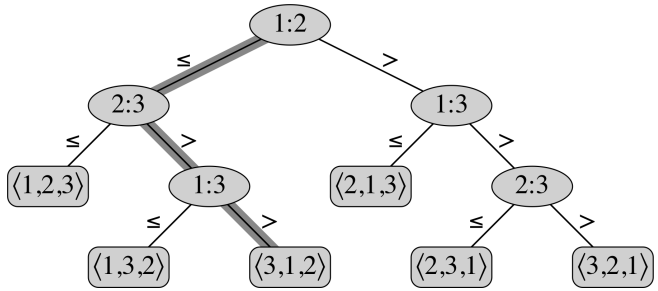
Labels for blade (svar når algoritmen stopper): hvilken opstilling som skal laves for at få sorteret orden (angivet med liste af ID'er, dvs. af oprindelige indekser for input-elementer).

Worst-case køretid: længste rod-blad sti = træets højde.

Bemærk: Insertionsort, selectionsort, mergesort, quicksort, heapsort kan alle beskrives sådan.


Mergesort
n=4
0 < 1
1023
Nej
ja
0123
2 < 3
2 < 3
1032
Nej
ja
1023
0123
Nej
ja
0132
1 < 3
1 < 2
0 < 3
0 < 3
0 < 2
ja
0
32
1 < 2
0 < 3
1 < 3
0 < 2
0 < 2
1 < 3
0 < 3
1 < 2
0
2
Nej
3210
0 < 2
0 < 2
1032
2310
0 < 3
0 < 3
1023
3201
1 < 2
1 < 2
0132
2301
1 < 3
1 < 3
0123
3120
3102
1320
1302
2130
2103
1230
1203
3021
3012
0321
0312
2031
2013
0231
0213

Sammenligningsbaseret sortering



2^0
2^1
2^2
2^3

For en fast samling af n elementer er der $n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot \ldots \cdot n$ forskellige input (rækkefølger af elementer).

Hvis algoritmen (træet) skal kunne sortere alle disse, skal der være mindst $n!$ blade - ellers vil der være to forskellig input som leder til samme svar, og for det ene input må svaret være forkert.

Et træ af højde h har højst 2^h blade (da det fulde træ af højde h har det).

$$2^h \geq \text{ antal blade } \geq n!$$

$$h \geq \log(n!) = \log(1 \cdot 2 \cdot 3 \ldots \cdot n)$$

$$= \log(1) + \log(2) + \ldots \underbrace{\log(n/2) \cdots + \log(n)}_{\geq \frac{n}{2} \text{ led}} \geq \frac{n}{2} \cdot \log(\frac{n}{2}) = \frac{n}{2}(\log(n) - 1)$$

($\geq \log(n/2)$, $\geq \log(n/2)$)

$$h = \Omega(n \log n)$$

Counting sort

Elementer heltal: elementer kan bruges som array-indekser ($\neq$ at bruge sammenligninger på elementer).

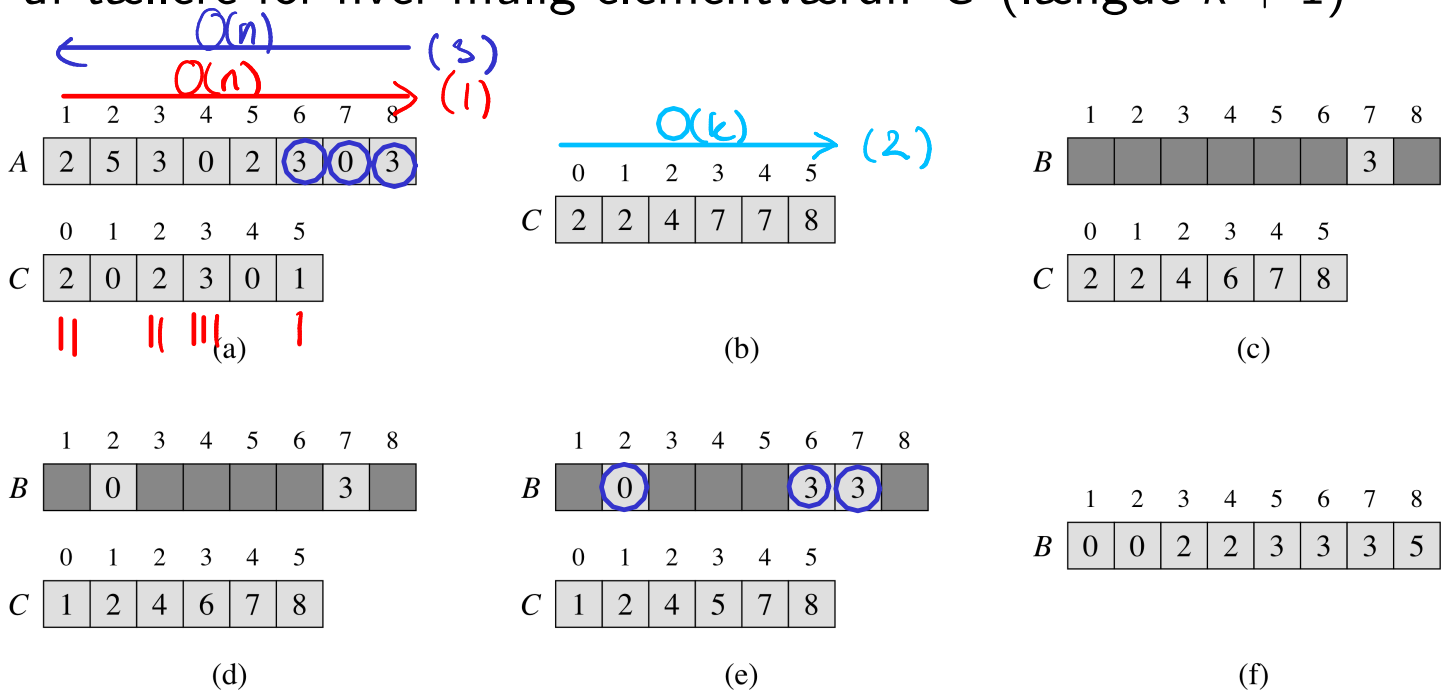
Counting sort: Sorterer n heltal af størrelse mellem 0 og k (inkl.).

Inputarray: A (længde n)

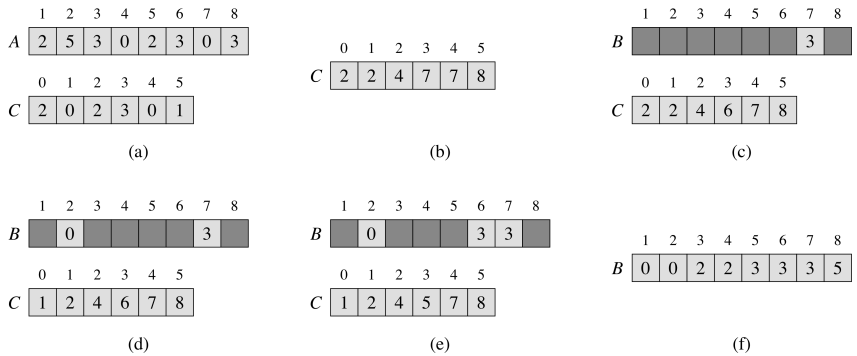
Outputarray: B (længde n)

Array af tællere for hver mulig elementværdi: C (længde $k+1$)

$k=5$



Counting sort




```
COUNTING-SORT(A,B,k)
  for i = 0 to k
    C[i] = 0
  for j = 1 to A.length
    C[A[j]] + +
  for i = 1 to k
    C[i] = C[i] + C[i − 1]
  for j = A.length downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] − −
```

$O(n)$

$O(k)$

$O(n)$

$O(n)$, hvis $k \in O(n)$

$O(n^3)$, hvis $k = n^3$

Tid: $O(n+k)$

Bemærk: stabil (da sidste løkke løber baglæns gennem både A og B), dvs at elementer med ens værdier beholder deres indbyrdes plads.

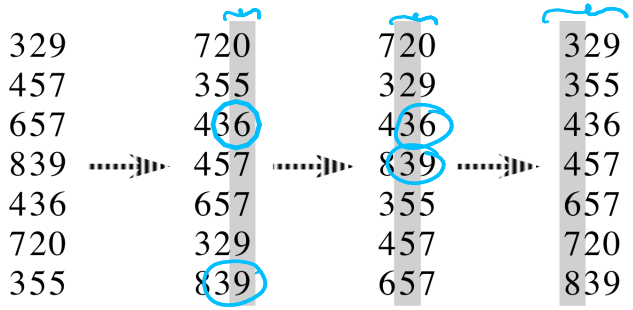
Radix sort

Radix sort: Sorterer n heltal alle med d cifre i base (radix) k.

(dvs. cifrene er heltal i $\{0, 1, 2, \ldots, k-1\}$)

På figuren nedenfor er der 7 heltal med 3 cifre i base 10.

RADIX-SORT(A,d)
  for $i = 1$ **to** d
    use a stable sort to sort A on digit i from right



Tid: $O(d(n+k))$ hvis der bruges Counting Sort i **for**-løkken.

Korrekthed:

Efter i'te iteration af **for**-løkken er A sorteret hvis man kun kigger på de i cifre mest til højre.

Radix sort

Eksempel: heltal i 10-talsystemet med bredde 12

486 239 123 989

Countingsort sorterer disse i tid $O(n + 10^{12})$

Dette er $O(n)$ hvis $n \geq 10^{12} = 1.000.000.000.000$

Se som 2-cifrede tal i base 10^6 (bemærk: sorteret orden er den samme)

486 239	123 989

Radixsort sorterer disse i tid $O(2(n + 10^6))$

Dette er $O(n)$ hvis $n \geq 10^6 = 1.000.000$

Se som 4-cifrede tal i base 10^3 (bemærk: sorteret orden er den samme)

486	239	123	989

Radixsort sorterer disse i tid $O(4(n + 10^3))$

Dette er $O(n)$ hvis $n \geq 10^3 = 1.000$

Hvis alle tal er $< n^d$, kan de skrives skrives med d cifre i base n:

$c_{d-1} n^{d-1} + c_{d-2} n^{d-2} + \ldots + c_1 n + c_0$

O.v.s.:

Hvis alle tal er $\leq n^d$, $d \in O(1)$, får man køretid $O(d(n+n)) = O(n)$ ved at bruge base n.

Radix sort

Eksempel: heltal i 2-talsystemet med bredde 32 (dvs. binære tal med 32 bits)

11011001 10011000 01101000 10110101

Countingsort sorterer disse i tid $O(n + 2^{32})$

Dette er $O(n)$ hvis $n \geq 2^{32} = 4.294.967.296$

Se som 2-cifrede tal i base 2^{16} (bemærk: sorteret orden er den samme)

11011001 10011000	01101000 10110101

Radixsort sorterer disse i tid $O(2(n + 2^{16}))$

Dette er $O(n)$ hvis $n \geq 2^{16} = 65.536$

Se som 4-cifrede tal i base 2^8 (bemærk: sorteret orden er den samme)

11011001	10011000	01101000	10110101

Radixsort sorterer disse i tid $O(4(n + 2^8))$

Dette er $O(n)$ hvis $n \geq 2^8 = 256$