

2.1-3 søgning og sortering Noter

CLRS 2

- Søgning
 - Linæer søgning
 - Binær søgning
- Sortering
 - Indsættelsessortering
 - Flettesortering

Søgning og Sortering

- Søgning
 - Linæer søgning
 - Binær søgning
- Sortering
 - Indsættelsessortering
 - Flettesortering

Søgning

- **Søgning.** Givet en **sorteret** tabel A og et tal x, afgør om der findes indgang i, så $A[i] = x$.
- **Sorteret tabel.** En tabel $A[0..n-1]$ er sorteret hvis $A[0] \leq A[1] \leq \dots \leq A[n-1]$ (ikke-faldende rækkefølge).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50

Linæer søgning

- **Lineær søgning.** Undersøg for alle indgange i om $A[i] = x$.
- **Tid.** $\Theta(n)$
- **Udfordring.** Kan vi udnytte at tabellen er sorteret til at gøre det bedre?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50

Binær søgning

- Binær søgning (*binary search*). Kig på midterste indgang m i A.
 - hvis $A[m] = x$ returner sand og stop.
 - hvis $A[m] < x$ fortsæt **rekursivt** på højre halvdel.
 - hvis $A[m] > x$ fortsæt **rekursivt** på venstre halvdel.
- Stop hvis tabellen har størrelse ≤ 0 og returner falsk.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50

Binær søgning

```
BINÆRSØGNING(A, i, j, x):  
    If j < i: Return False  
    m = floor((i+j)/2)  
    If A[m] == x:  
        Return True  
    Elseif A[m] < x:  
        Return BINÆRSØGNING(A, m+1, j, x)  
    Else: Return BINÆRSØGNING(A, i, m-1, x) // A[m] > x
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50

- **Tid.** Hvor hurtigt kører den?
- **Analyse 1.** Analog til analyse af rekursiv toppunktsalgoritme.
 - Et rekursivt kald tager konstant tid.
 - Hvert rekursivt kald **halverer** tabellen vi kigger på. Vi stopper når tabellen har størrelse ≤ 0 .
 - \Rightarrow Køretiden er $\Theta(\log n)$

Binær søgning

- [Analyse 2](#). Lad $T(n)$ være køretiden for binær søgning.
 - Opskriv og udregn **rekursionsligningen** for $T(n)$.

$$T(n) = \begin{cases} T(n/2) + c, & n > 1 \\ d, & n = 1 \end{cases}$$

Bemærk: c og d er ukendte konstanter. Det vigtige er relationen med $T(n/2)$!

- Hvad giver denne ligning?

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + c \\ &= T\left(\frac{n}{4}\right) + c + c \\ &= T\left(\frac{n}{2^{\log_2 n}}\right) + c \cdot \log_2 n \\ &= d + c \cdot \log_2 n \\ &= \Theta(\log n) \end{aligned}$$

Søgning

- Vi kan søge i en sorteret tabel i
 - $\Theta(n)$ tid med lineær søgning.
 - $\Theta(\log n)$ tid med binær søgning.

Søgning og Sortering

- Søgning
 - Linær søgning
 - Binær søgning
- Sortering
 - Indsættelsessortering
 - Flettesortering

Sortering

- **Sortering.** Givet en tabel $A[1..n]$ returner en tabel $B[1..n]$ med samme værdier som A men i sorteret orden.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
33	4	25	28	45	18	7	12	36	1	47	42	50	16	31

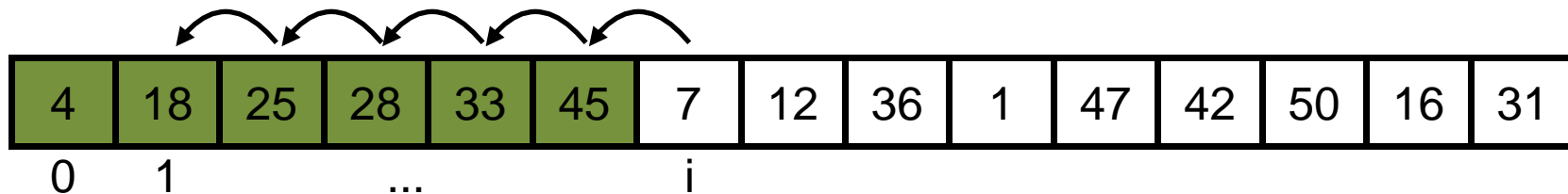
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	4	7	12	16	18	25	28	31	33	36	42	45	47	50

Anvendelser

- Oplagte.
 - Sortere en liste af navne, organisere et MP3 bibliotek, vise Google PageRank resultater, vise Facebook feed i kronologisk rækkefølge.
- Ikke oplagte.
 - Datakompression, computergrafik, bioinformatik, anbefalingssystemer (film på Netflix, bøger på Amazon, reklamer på Google,..).
- Nemme problemer for sorteret data.
 - Binær søgning, find median, identificer duplikater, find tætteste par, find statistiske perifere observationer (*outliers*).

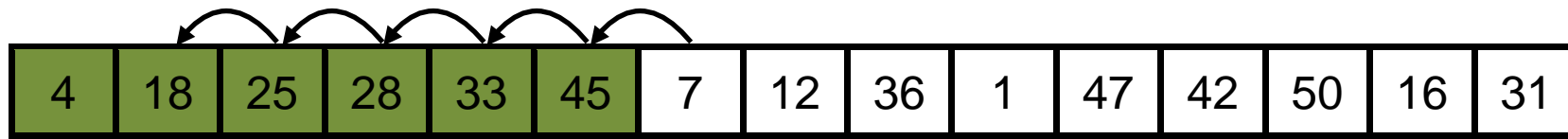
Indsættelsessortering (*insertion-sort*)

- **Indsættelsessortering**. Start med en usorteret tabel A.
- Kig på indgangene fra venstre til højre i n **runder**.
- Ved runde i:
 - Deltabel $A[0..i-1]$ er sorteret.
 - Indsæt $A[i]$ i $A[0..i-1]$ så $A[0..i]$ er sorteret.
 - For at finde rette sted til $A[i]$ sammenligner vi med indgangene fra højre til venstre.



Indsættelsessortering

```
INDSÆTTELSORTERING(A, n)
  For i = 1 to n-1:
    j = i
    While j > 0 and A[j-1] > A[j]:
      ombyt A[j] og A[j-1]
      j = j - 1
```



- **Tid.** Hvad er køretiden $T(n)$?
 - Hvad er tiden for at indsætte $A[i]$ i sorteret rækkefølge blandt $A[0..i-1]$?
 - $c \cdot i$ tid for en konstant c . (husk vi arbejder med worst-case)
 - \Rightarrow samlet tid:

$$T(n) = \sum_{i=1}^n ci = c \sum_{i=1}^n i = \frac{cn(n+1)}{2} = \Theta(n^2)$$

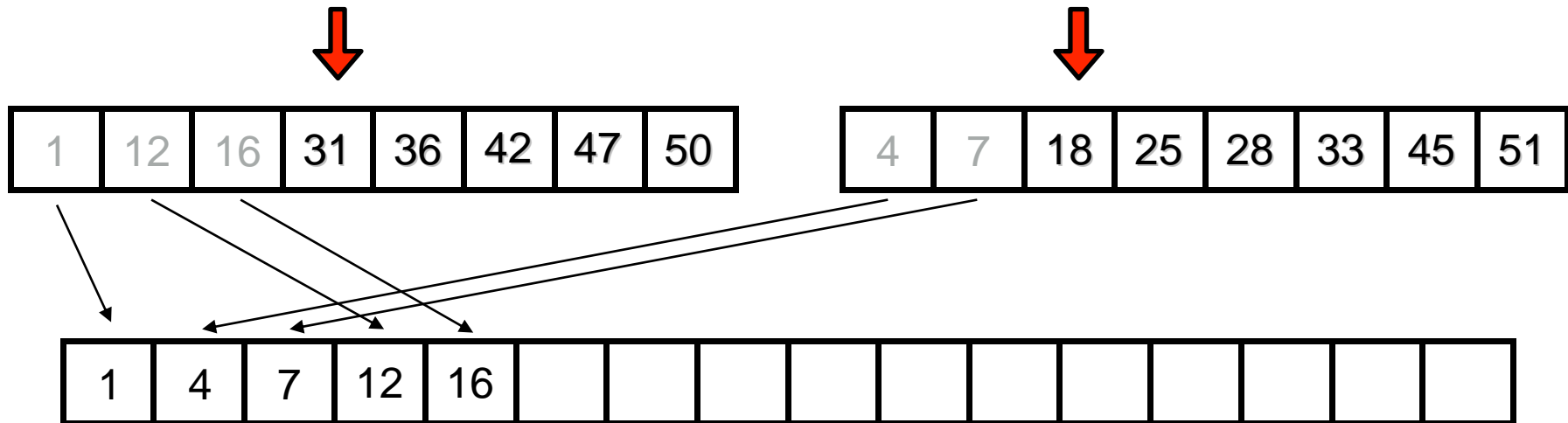
- **Udfordring.** Kan vi sortere hurtigere?

Flettesortering (*mergesort*)

- [Flettesortering](#). Hurtig sorteringsalgoritme.
- [Ide](#). Rekursiv sortering vha. **fletning** af sorterede deltabeller.

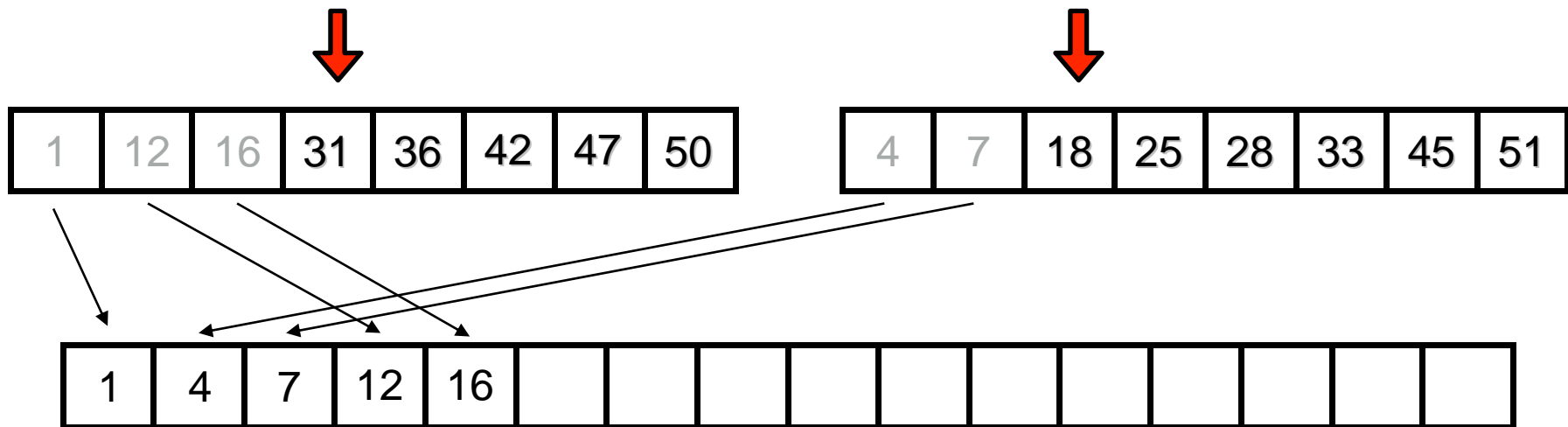
Fletning (*merge*)

- **Mål.** Kombiner to sorterede tabeller til én sorteret tabel.
- **Ide.**
 - Gennemløb begge sorterede tabeller i stigende rækkefølge:
 - I hvert skridt indsæt det mindste af de to nuværende elementer fra tabellerne i en ny tabel.
 - Ryk frem i tabellen med det mindste element.
 - Gentag indtil alle elementer er indsat i ny tabel.



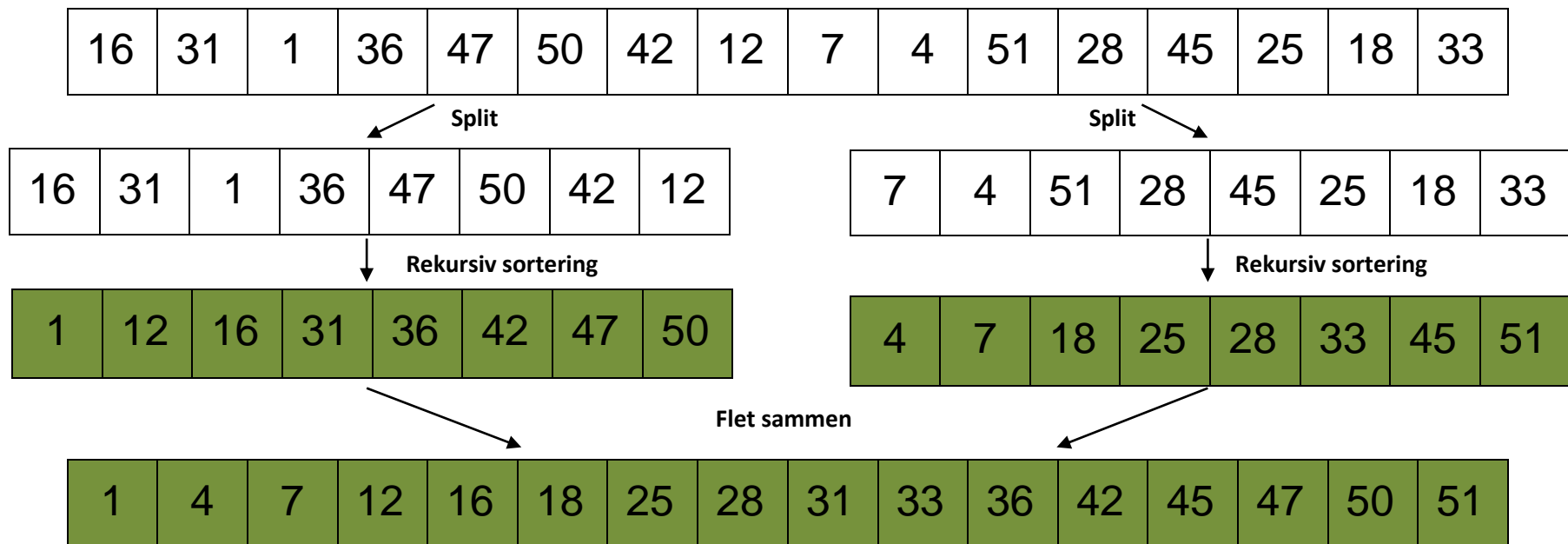
Fletning

- **Tid.** Hvor hurtigt kører fletning på to tabeller A_1 og A_2 ?
 - Hvert skridt i algoritmen tager $\Theta(1)$ tid.
 - I hvert skridt flytter vi en indgang frem i en af tabellerne.
 - $\Rightarrow \Theta(|A_1| + |A_2|)$ tid.



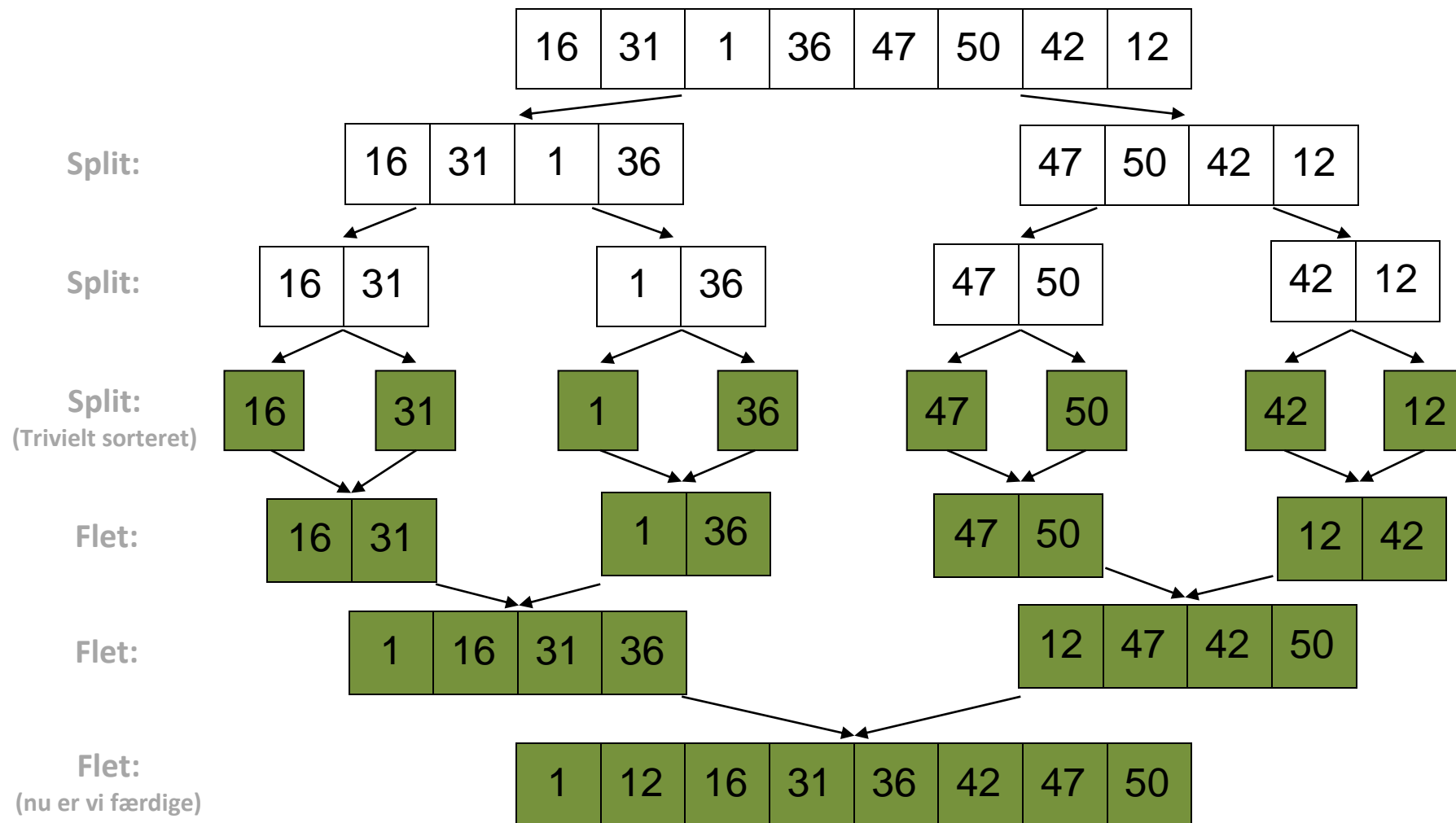
Flettesortering (*mergesort*)

- Flettesortering.
- Hvis $|A| \leq 1$, returner A.
- Ellers:
 - Del A i to halvdele.
 - Sorter hver halvdel rekursivt.
 - Flet de to halvdele sammen.



Flettesortering (*mergesort*)

- Hvordan virker rekursionen?



```
FLETTESORTERING (A, i, j)
    if i < j
        m = floor((i+j)/2)
        FLETTESORTERING (A, i, m)
        FLETTESORTERING (A, m+1, j)
    FLET (A, i, m, j)
```

- **Tid.** Lad $T(n)$ være køretiden af flettesortering på n tal..
- Hvordan kan vi udregne $T(n)$?
- **Ide.** Opstil **rekursionstræ** og udregn samlet tid for alle rekursive kald.

Flettesortering

Illustration af algoritmen:

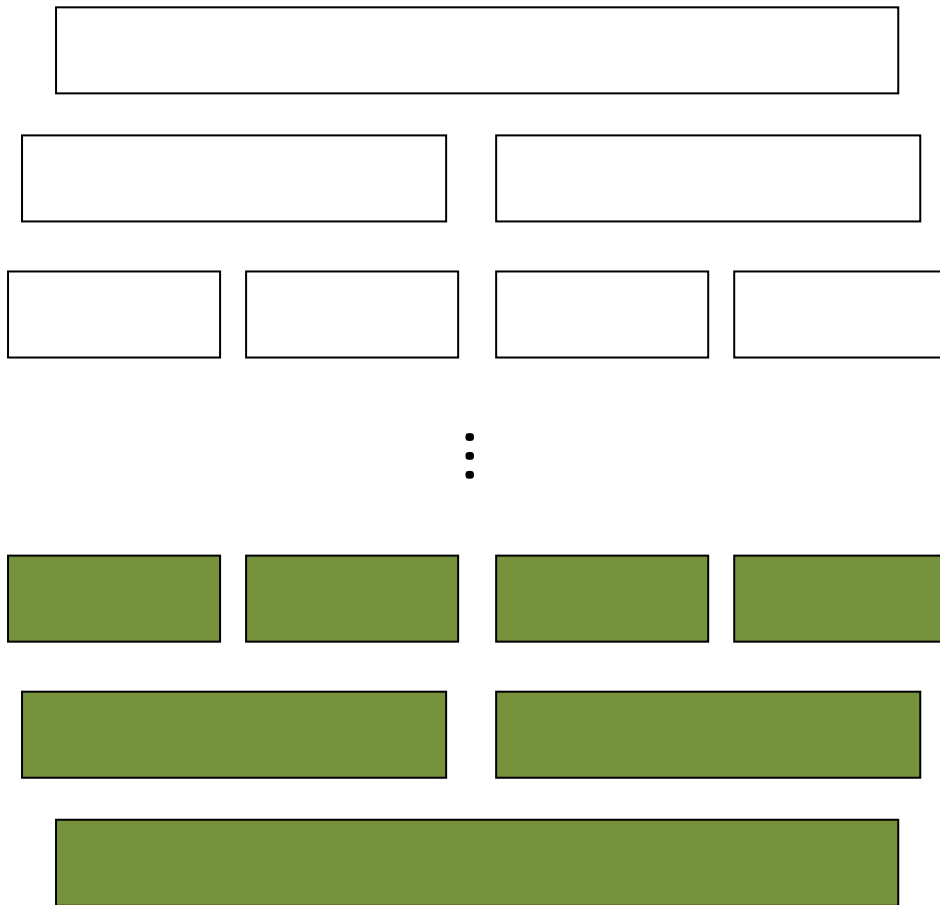
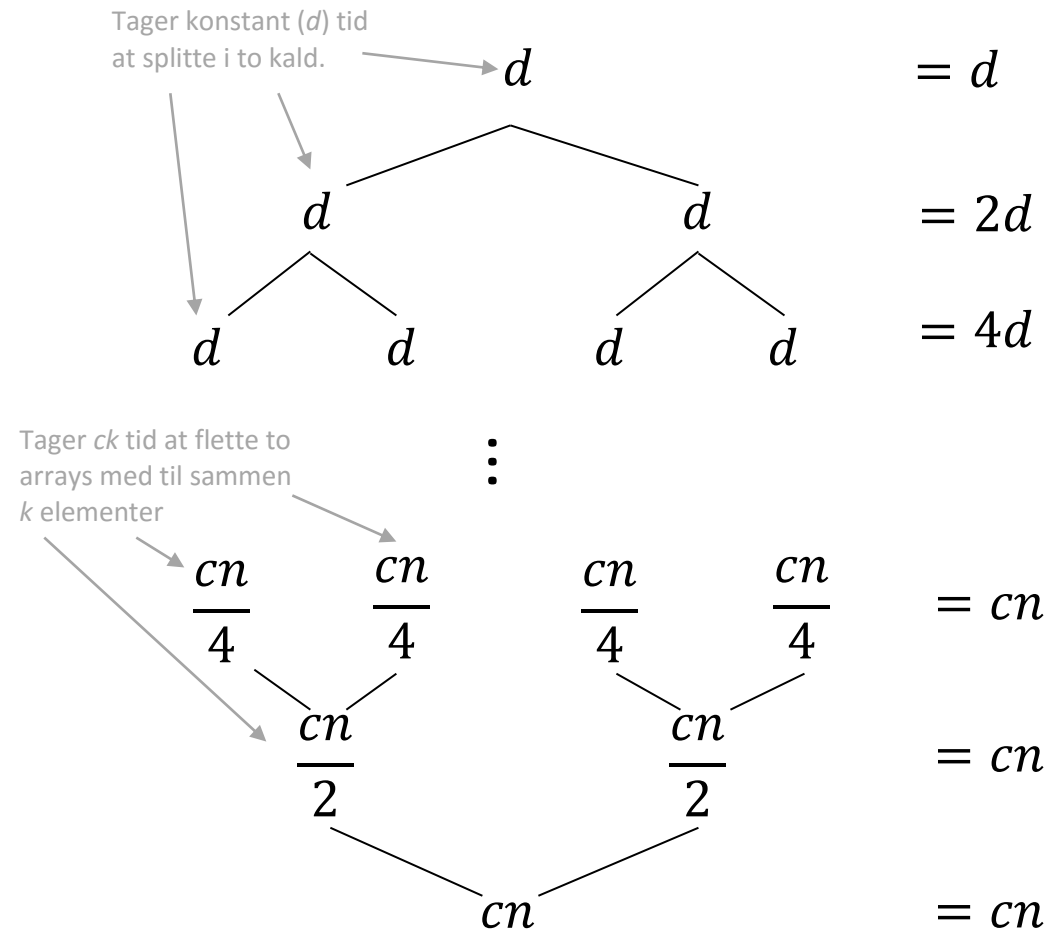


Illustration af køretid:



$$T(n) = \sum_{i=0}^{\log_2 n} 2^i d + cn \log_2 n = (2n - 1)d + cn \log_2 n = \Theta(n \log n)$$

Sortering

- Vi kan sortere en tabel i
 - $\Theta(n^2)$ tid med indsættelsessortering.
 - $\Theta(n \log n)$ tid med flettesortering.

Del og hersk (*divide-and-conquer*)

- Flettesortering er eksempel på en **del-og-hersk** algoritme.
- **Del-og-hersk.** algoritmisk **designparadigme**.
 - **Del.** Opdel problemet i et eller flere delproblemer
 - **Hersk.** Løs delproblemerne rekursivt
 - **Kombiner.** Sæt løsningerne til delproblemerne sammen til en samlet løsning for problemet.
- **Flettesortering.**
 - **Del.** Del A i to halvdele.
 - **Hersk.** Sorter hver halvdel rekursivt.
 - **Kombiner.** Flet de to halvdele sammen.

Søgning og Sortering

- Søgning
 - Linæer søgning
 - Binær søgning
- Sortering
 - Indsættelsessortering
 - Flettesortering