# Søgning og sortering

- Søgning i et sorteret array
  - Lineær søgning
  - Binær søgning
- Sortering
  - Indsættelsessortering
  - Flettesortering

# Søgning i et sorteret array

- Søgning i sorteret array: Givet et sorteret array A og et element $x$, afgør, om der findes en indgang `i` så `A[i]` er lig med $x$.
  - Input: Sorteret array A, element x,
  - Output: Findes en indgang i, så A[i] = x? Sandt/falsk.
- Sorteret array: Et array `A[0..n-1]` er sorteret hvis
  `A[0]` $\leq$ `A[1]` $\leq$ `A[2]` $\leq \cdots \leq$ `A[n-1]`.
  (Svagt stigende / 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 |

# Lineær søgning

- Lineær søgning: Gennemløb arrayet og check alle elementer, indtil enten man finder det søgte element, eller har gennemløbet hele arrayet.
- Tid: $\Theta(n)$.
- Fordel: Virker også på ikke-sorteret array.
- Udfordring: Hurtigere algoritme for sorteret array?

| 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: Kig på `A`s midterste element, m.
  - Hvis `A[m]` $= x$ returner sand. (afslut.)
  - Hvis `A[m]` $< x$, er også `A[0]`, `A[1]`, $\ldots$, `A[m]` $< x$, fortsæt til højre for `m`.
  - Hvis `A[m]` $> x$, fortsæt til venstre for `m`.
  - Fortsæt rekursivt.
  - Når det aktive område er $\leq 0$, returner "falsk" og afslut.

| 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 – analyse 1

```
BINARYSEARCH(A,i,j,x)
   if j < i return false
   m = ⌊i+j/2⌋
   if A[m] = x return true
   elseif A[m] < x return BINARYSEARCH(A,m+1,j,x)
   else return BINARYSEARCH(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?
- Analyse 1. Samme analyse som Toppunkt3.
  - Hvert rekursivt kald tager konstant tid.
  - Hver gang halveres intervallet.
  - Når intervallet er $\leq 1$ er der $\leq 1$ runde til, at vi stopper.
  - Derfor: tiden er $\Theta(\log n)$.

# Binær søgning – analyse 2

- Analyse 2. Binær søgning på array af længde $n$ tager $T(n)$ tid.
  - Løs rekursionsligningen for $T(n)$.

$$T(n) = \begin{cases} T(n/2) + c_1 & \text{hvis } n > 1 \\ c_2 & \text{hvis } n \leq 1 \end{cases}$$

$$\begin{aligned} T(n) &= T(n/2) + c_1 \\ &= T(n/4) + c_1 + c_1 \\ &= T(n/8) + c_1 + c_1 + c_1 \\ &\cdots \\ &= T(n/2^k) + k \cdot c_1 \\ &\cdots \\ &= T(n/2^{\log_2(n)}) + \log_2(n) \cdot c_1 \\ &= T(1) + c_1 \cdot \log_2(n) \\ &= c_2 + c_1 \cdot \log_2(n) \text{ som er } \Theta(\log n) \end{aligned}$$

# Søgning i et sorteret array – konklusion

- Lineær søgning i lineær tid, vi skriver $\Theta(n)$,
- Binær søgning i logaritmisk tid, vi skriver $\Theta(\log n)$.

# Søgning og sortering

- Søgning i et sorteret array
  - Lineær søgning
  - Binær søgning
- Sortering
  - Indsættelsessortering
  - Flettesortering

# Sortering – problem

- Sortering. Givet et array A[0..n-1] returner et sorteret array B[0..n-1] med samme værdier som A.

| 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 |

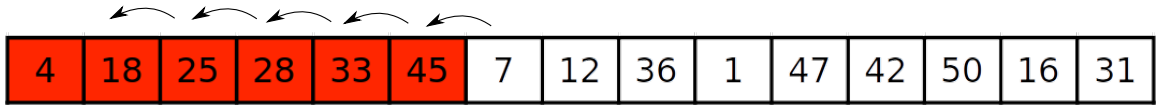
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 7 | 12 | 16 | 18 | 25 | 28 | 31 | 33 | 36 | 42 | 45 | 47 | 50 |

# Sortering – motivation

- Oplagte anvendelser.
  - Sortere en liste af navne, organisere et bibliotek over lydfiler, vise resultater fra en websøgning, vise sociale mediers feed i kronologisk rækkefølge.
- Mindre oplagte.
  - Datakompression, computergrafik, bioinformatik, anbefalingssystemer.
- Når data er sorteret kan man hurtigt/nemt:
  - Søge, finde median, identificere duplikater, finde tætteste par, finde statistiske perifere observationer (outliers).

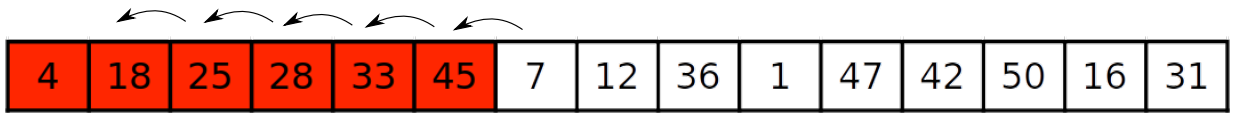
# Sortering – indsættelsessortering

- Indsættelsessortering: Start med usorteret array A.
- Kig på elementerne fra venstre til højre i n-1 runder.
- Runde i:
  - Delarray A[0..i-1] er sorteret.a
  - 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 elementerne fra højre til venstre.



# Sortering – indsættelsessortering

```
INSERTIONSORT(A, n)
    for i = 1 to n-1
        j = i
        while j > 0 and A[j-1] > A[j]
            swap A[j] og A[j-1]
            j = j - 1
```




- Tid? For at indsætte `A[i]` bruger vi $c \cdot i$ tid, hvor $c$ er en konstant.
  Samlet tid $T(n)$: $1 \cdot c + 2 \cdot c + 3 \cdot c + 4 \cdot c + \ldots + (n-1) \cdot c$

$$T(n) = \sum_{i=1}^{n-1} i \cdot c = c \sum_{i=1}^{n-1} i = c \cdot \frac{n \cdot (n-1)}{2} \text{ som er } \Theta(n^2)$$

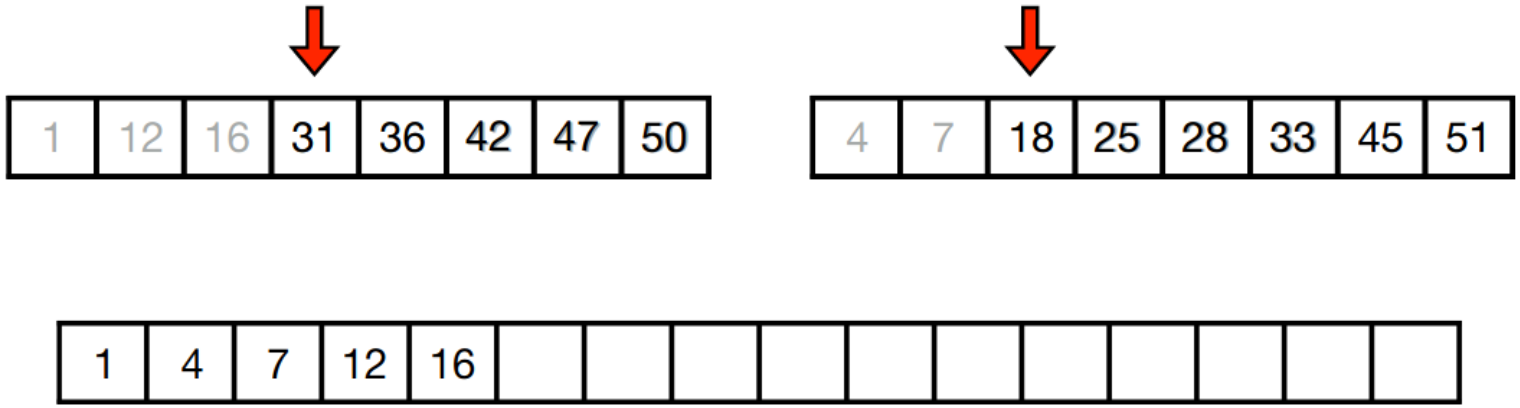
- Udfordring: Kan vi sortere hurtigere?

# Sortering – flettesortering

- Flettesortering. (Merge sort.)
  - Idé: Rekursiv sortering ved at flette sorterede delarrays.
  - Flette?

# Fletning

- Flette: Kombinere to sorterede arrays til ét forenet sorteret array.
- Idé: Gennemløb begge sorterede arrays (venstre mod højre).
  - Til enhver tid betragter vi højest to elementer, et fra hvert array.
  - Sammenlign og vælg det mindste element.
  - Dette skrives på første ledige plads i vores output,
  - Ryk frem i det array hvor det kom fra.
  - Fortsæt ind til alle elementer er kopieret til output.
- Analogi: lynlås.



# Fletning

- Tid: Hvor hurtigt kører fletning på to arrays $A_1$ og $A_2$?
  - Hvert skridt i algoritmen tager $\Theta(1)$ (dvs. konstant) tid.
  - I hvert skridt flytter vi et element frem i et af arraysene.
  - I alt: $\Theta(|A_1| + |A_2|)$ tid.
  - At flette tager tid proportionalt med det samlede antal elementer i $A_1$ og $A_2$.

# Sortering – flettesortering

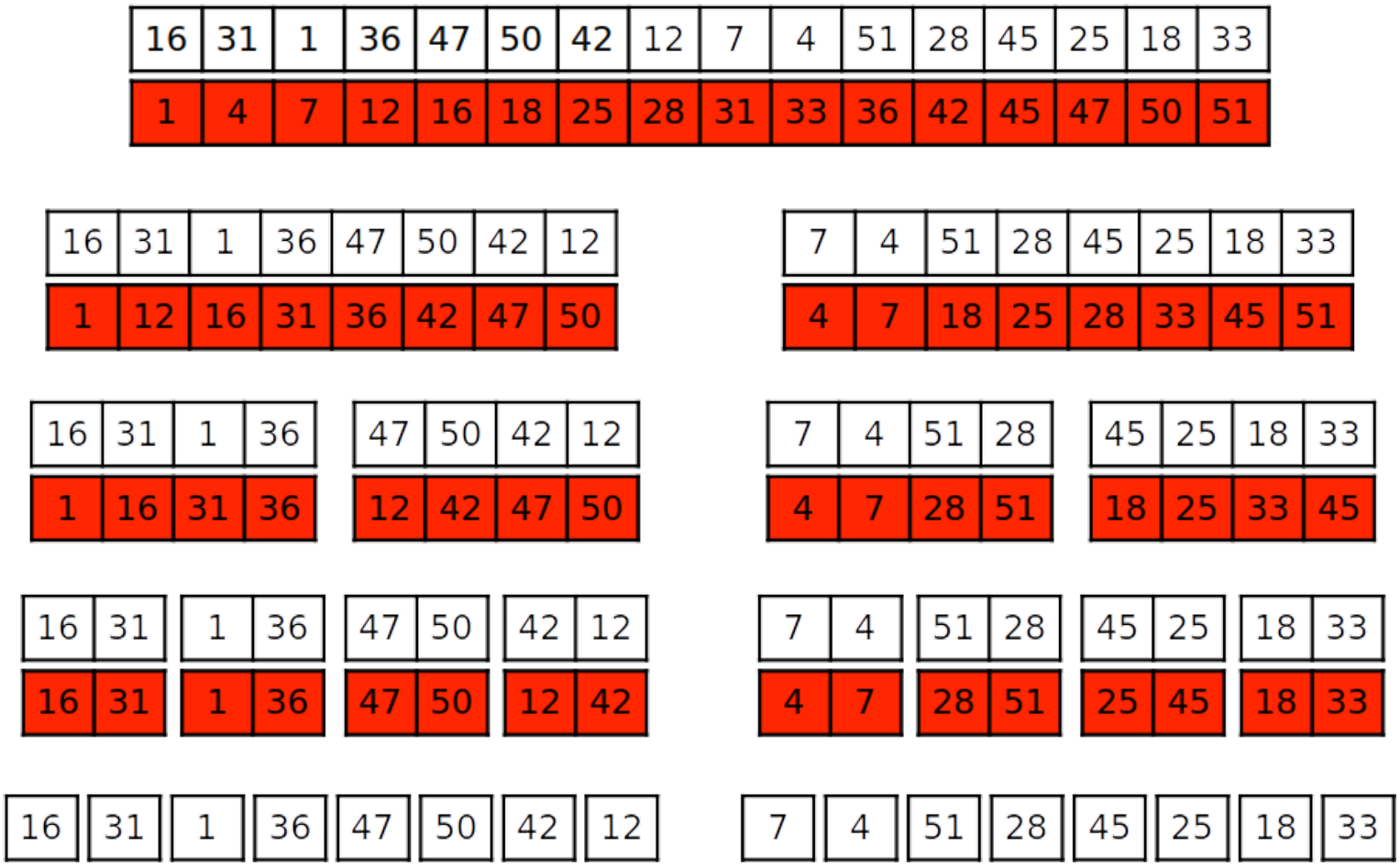
- Flettesortering af et array $A$.
- Hvis $|A| \leq 1$ er $A$ allerede sorteret så returner $A$.
- Ellers:
  - Del $A$ i to halvdele.
  - Sorter hver halvdel rekursivt.
  - Flet de to sorterede halvdele sammen, og returner resultatet.

| 16 | 31 | 1 | 36 | 47 | 50 | 42 | 12 | 7 | 4 | 51 | 28 | 45 | 25 | 18 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 7 | 12 | 16 | 18 | 25 | 28 | 31 | 33 | 36 | 42 | 45 | 47 | 50 | 51 |

| 16 | 31 | 1 | 36 | 47 | 50 | 42 | 12 |
|---|---|---|---|---|---|---|---|
| 1 | 12 | 16 | 31 | 36 | 42 | 47 | 50 |

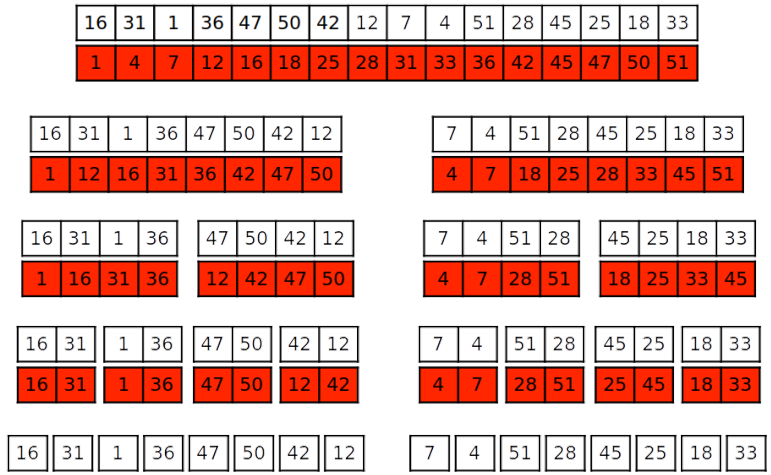
| 7 | 4 | 51 | 28 | 45 | 25 | 18 | 33 |
|---|---|---|---|---|---|---|---|
| 4 | 7 | 18 | 25 | 28 | 33 | 45 | 51 |

# Sortering – flettesortering – eksempel



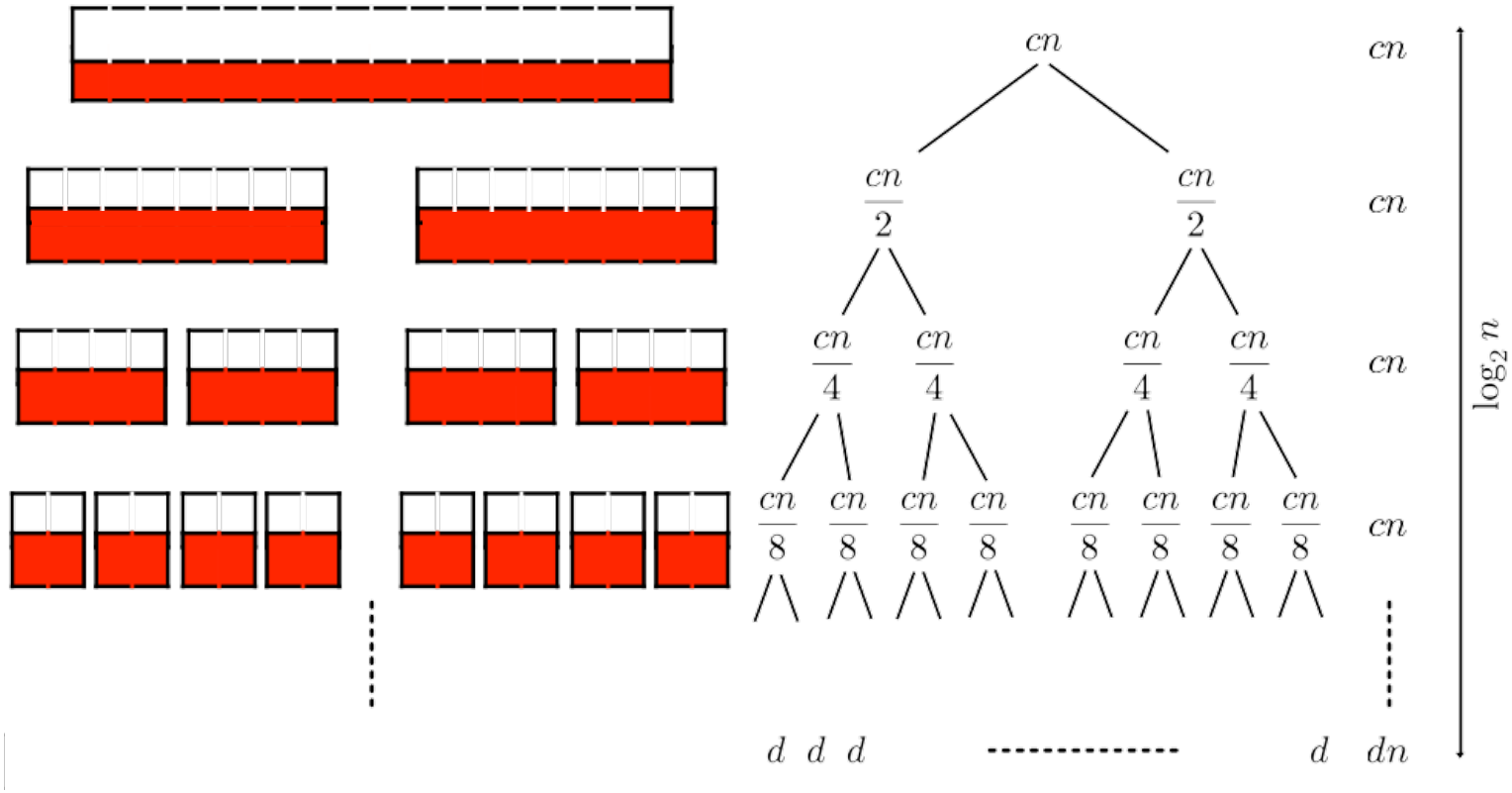
# Sortering – flettesortering

```
MERGESORT(A,n)
    n = A.length
    if n>1
        m =⌊n/2⌋
        A1 = A[0..m]
        A2 = A[m+1..n-1]
        MERGESORT(A1)
        MERGESORT(A2)
        A = MERGE(A1,A2)
```




- Tid: Kørselstid $T(n)$, når input $A$ har længde $n$.
- Hvad er $T(n)$?
- Idé: Opstil rekursionstræ og udregn $T(n)$. (Tavle.)

# Sortering – flettesortering




$$T(n) = c \cdot n \cdot \log_2(n) + d \cdot n, \text{ som er } \Theta(n \log n)$$

# Sortering – konklusion

- Vi kan sortere et array
  - i $\Theta(n^2)$ tid med indsættelsessortering,
  - i $\Theta(n \log n)$ tid med flettesortering.

# Del og hersk

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

# Søgning og sortering

- Søgning i et sorteret array
  - Lineær søgning
  - Binær søgning
- Sortering
  - Indsættelsessortering
  - Flettesortering