

Divide-and-Conquer algoritmer

Divide-and-Conquer algoritmer

Det samme som [rekursive algoritmer](#).

Divide-and-Conquer algoritmer

Det samme som [rekursive algoritmer](#).

1. Opdel problem i mindre delproblemer (af [samme type](#)).
2. Løs delproblemerne ved rekursion (dvs. kald algoritmen selv, men med de mindre input).
3. Konstruer en løsning til problemet ud fra løsningen af delproblemerne.

Basistilfælde: Problemer af størrelse $O(1)$ løses direkte (uden rekursion).

Divide-and-Conquer algoritmer

Det samme som [rekursive algoritmer](#).

1. Opdel problem i mindre delproblemer (af [samme type](#)).
2. Løs delproblemerne ved rekursion (dvs. kald algoritmen selv, men med de mindre input).
3. Konstruer en løsning til problemet ud fra løsningen af delproblemerne.

Basistilfælde: Problemer af størrelse $O(1)$ løses direkte (uden rekursion).

NB: dette er en [generel algoritme-udviklingsmetode](#), med mange anvendelser.

Divide-and-Conquer

Hvis basistilfælde ($n = O(1)$):

- ▶ Lokalt arbejde

Hvis ikke basistilfælde:

- ▶ Lokalt arbejde
- ▶ Rekursivt kald
- ▶ Lokalt arbejde
- ▶ Rekursivt kald
- ▶ Lokalt arbejde

(Der behøver ikke altid være to rekursive kald. Nogle rekursive algoritmer har bare eet, og nogle har flere end to).

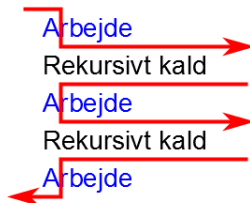
Divide-and-Conquer, flow of control

Flow of control (lokalt set, for eet kald af algoritmen):

Basistilfælde

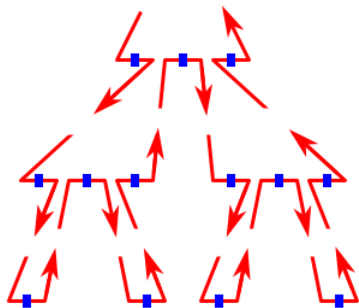


Ikke basistilfælde



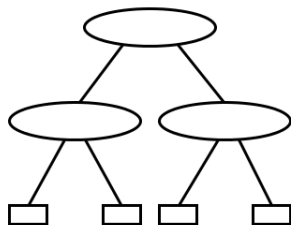
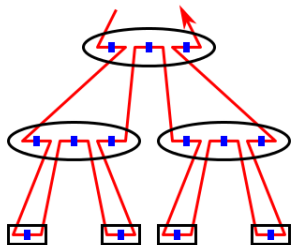
Divide-and-Conquer, udført arbejde

Globalt flow of control:



Rekursionstræer

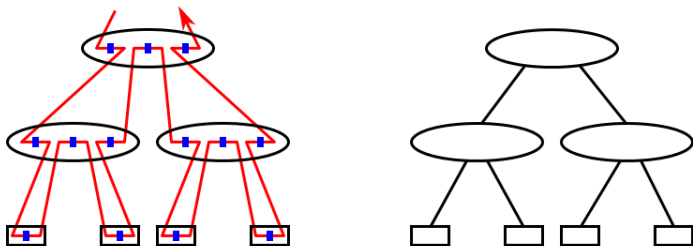
Globalt flow of control = rekursionstræer:



Een knude = eet kald af algoritmen.

Rekursionstræer

Globalt flow of control = rekursionstræer:



Een knude = eet kald af algoritmen.

Husk: alle kald på en sti mod roden er i gang samtidig – hvert kalds variable og state opbevares (af operativsystemet) på en stak, så kaldenes udførelse blandes ikke sammen.

- ▶ Kald af barn i rekursionstræet = push på stak.
- ▶ Afslutning af et barns udførelse = pop fra stak.