Divide-and-Conquer 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).

NB: dette er en generel algoritme-udviklingsmetode, med mange anvendelser.

For rekursive algoritmer: korrekthed bevises via induktion på inputstørrelse.

I detaljer: Korrekthed af store størrelser følger af korrekthed for små størrelser, samt handlingerne involveret i at konstruere en løsning for den store størrelse ud fra løsningerne for de mindre størrelser. Basistilfældet for rekursionen er også basistilfældet for induktionsbeviset.

Divide-and-Conquer eksempler

Mergesort:

- Del input op i to dele X og Y (trivielt).
- Sorter hver del for sig (rekursion).
- Merge de to sorterede dele til én sorteret del (reelt arbejde).

Basistilfælde: $n \leq 1$ (trivielt).

Quicksort:

- Del input op i to dele X og Y så $X \leq Y$ (reelt arbejde).
- Sorter hver del for sig (rekursion).
- Returner X efterfulgt af Y (trivielt)

Basistilfælde: $n \leq 1$ (trivielt).

Et andet eksempel er inorder gennemløb af et binært søgetræ.

Divide-and-Conquer mere generelt

- Del problem op i del-problemer (nyt hver gang)
- Løs del-problemer med rekursion (fast del)
- Konstruer løsning til problem ud fra løsningerne til del-problemerne (nyt hver gang).

Basistilfælde $n = O(1)$: Løs direkte (nyt hver gang, men altid simpelt).

Vi vil nu kigge på hvordan man vurderer køretiden for Divide-and-Conquer algoritmer (rekursive algoritmer).

Divide-and-Conquer, udført arbejde

Hvis basistilfælde ($n = O(1)$):

- Arbejde

Hvis ikke basistilfælde:

- Arbejde
- Rekursivt kald
- Arbejde
- Rekursivt kald
- Arbejde

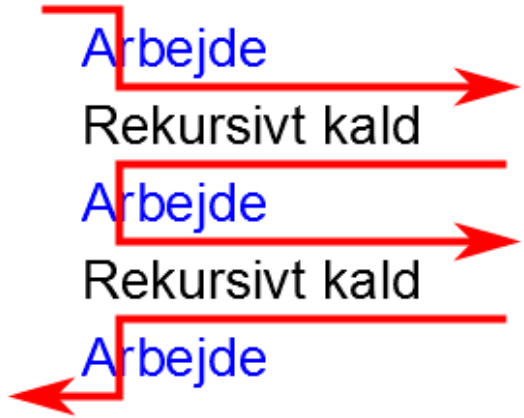
(Der behøver ikke altid være to rekursive kald. Nogle rekursive algoritmer har bare eet, og nogle har flere end to).

Divide-and-Conquer, udført arbejde

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

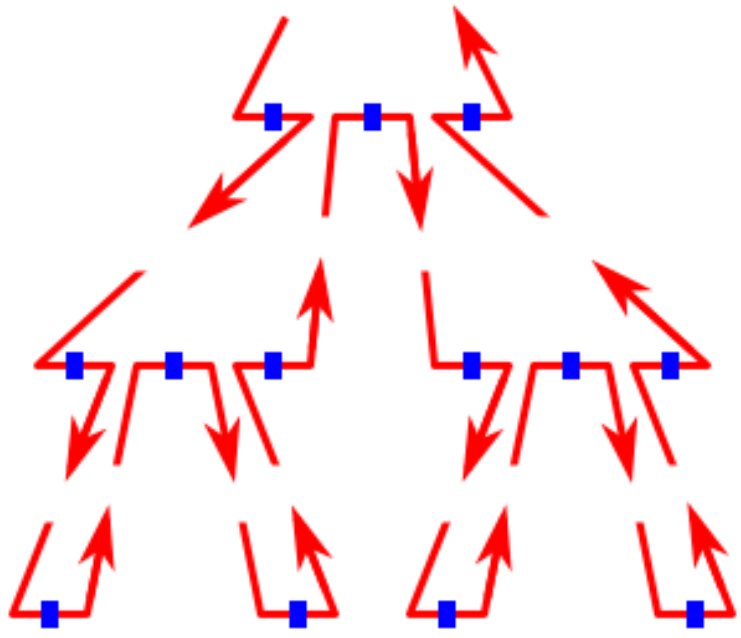
Basistilfælde

Ikke basistilfælde



Divide-and-Conquer, udført arbejde

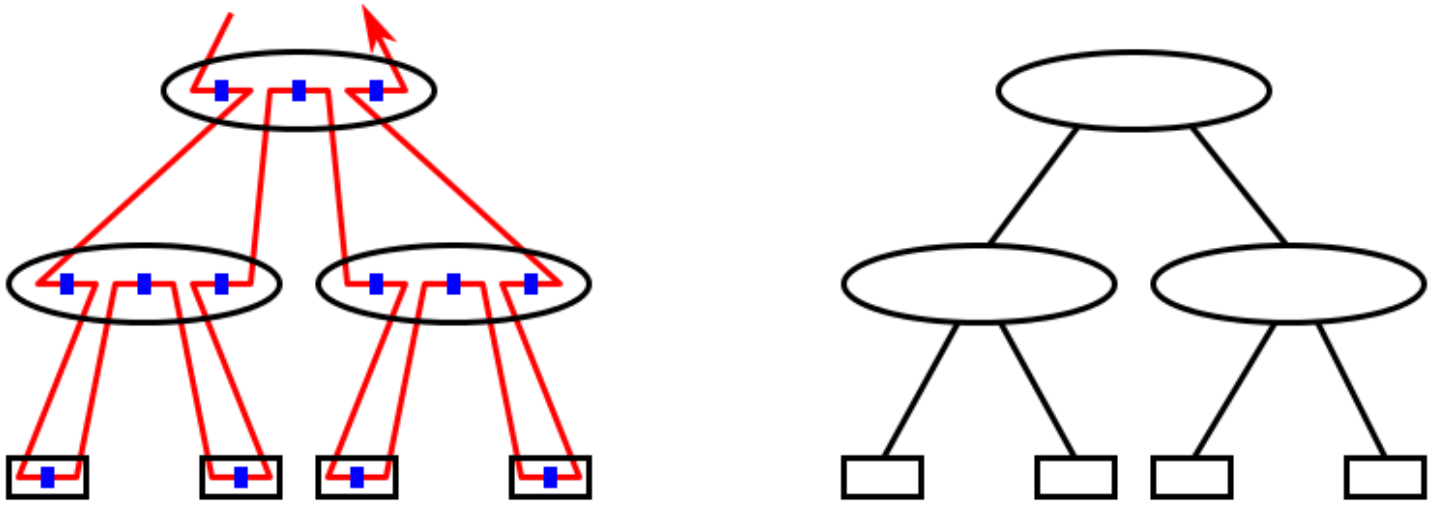
Globalt flow of control:



Vi ønsker af finde samlet arbejde = sum af blå bidder.

Rekursionstræer

Globalt flow of control = rekursionstræer:



Én knude = ét kald af algoritmen.

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

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

Rekursionsligninger

Køretiden for rekursive algoritmer kan beskrives ved rekursionsligninger.

Hvis en rekursiv algoritme for problemer af størrelse n laver a rekursive kald, alle på delproblemer af størrelse n/b, og laver $\Theta(f(n))$ lokalt arbejde, må der for køretiden $T(n)$ gælde:

$$T(n) = \begin{cases} aT(n/b) + \Theta(f(n)) & \text{hvis } n > 1 \\ \Theta(1) & \text{hvis } n \leq 1 \end{cases}$$

Sidste linie er altid den samme og udelades derfor ofte.

Eksempel: Mergesort har to rekursive kald af størrelse $n/2$ og laver $\Theta(n)$ lokalt arbejde. Dens rekursionsligning er derfor

$$T(n) = 2T(n/2) + \Theta(n)$$

Rekursionstræer og beregning af køretid

For en rekursiv algoritme (eller en rekursionsligning), annoter knuderne i rekursionstræet (for algoritmen eller ligningen) med

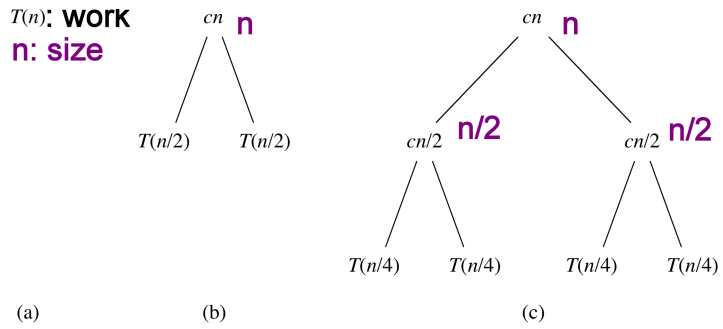
- Input størrelsen for kaldet til knude.
- Det resulterende arbejde udført i denne knude (men ikke i rekursive kald under knuden – de bliver selv annoteret med deres udførte arbejde).

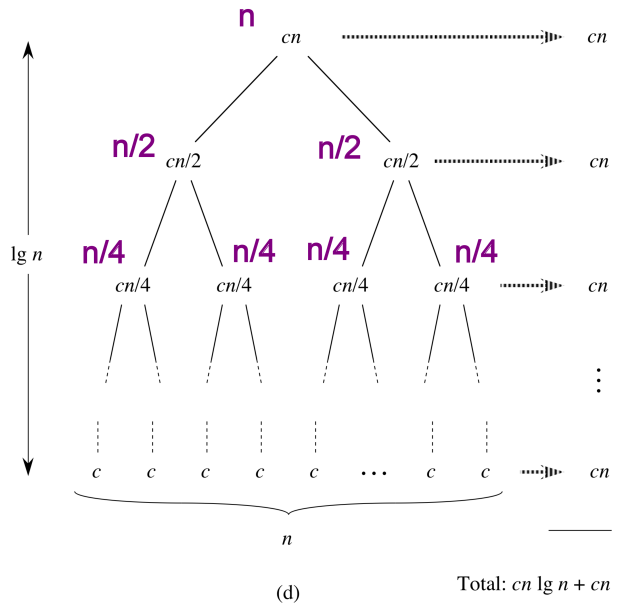
Find derefter summen af alt arbejde i knuder. Ofte:

- Sum hvert lag i rekursionstræet sammen for sig.
- Sum de resulterende værdier for alle lag. Find først højden af træet (antal lag).

Eksempler følger.

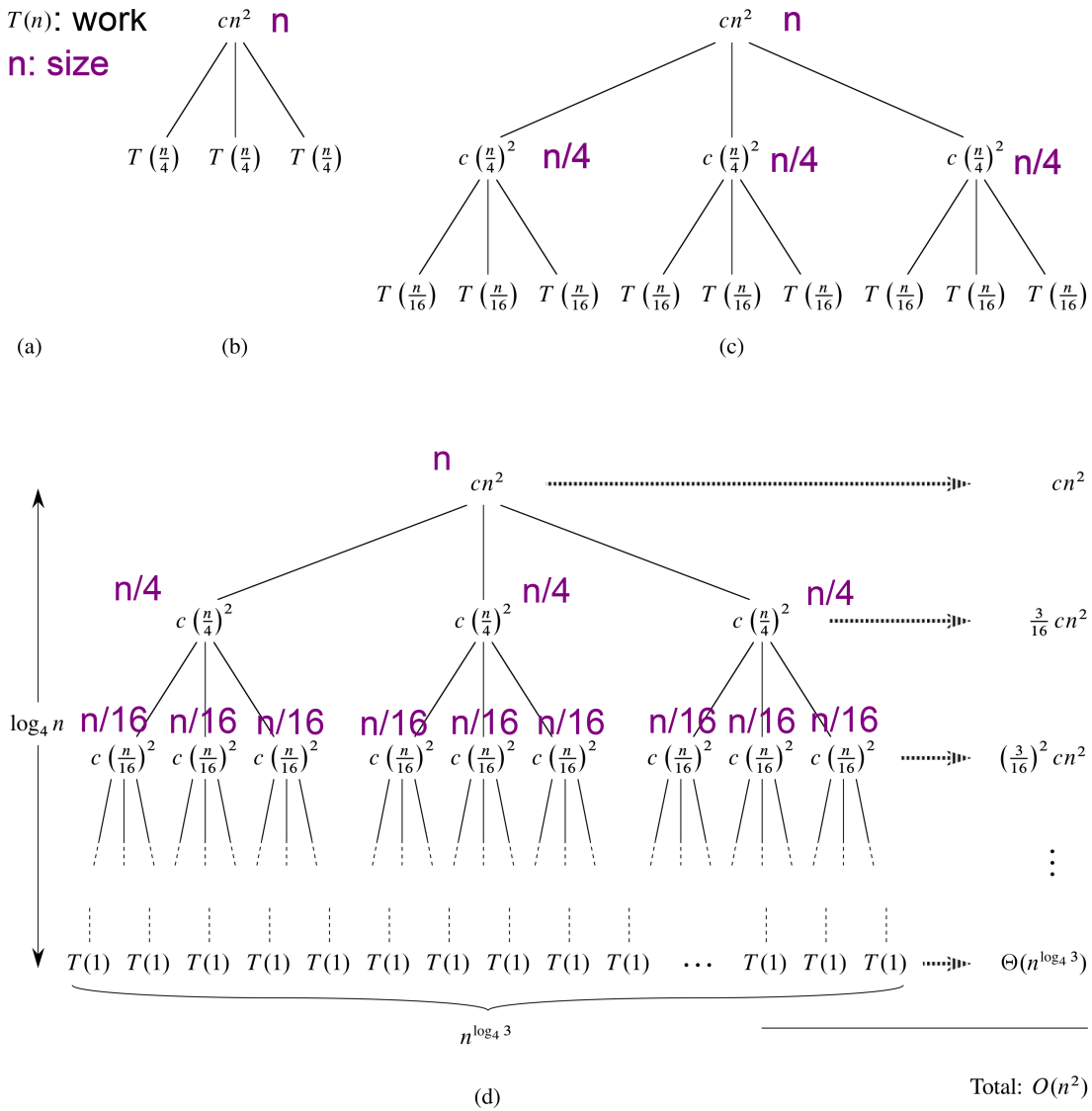
Divide-and-Conquer eksempler





$$T(n) = 2T(n/2) + cn$$

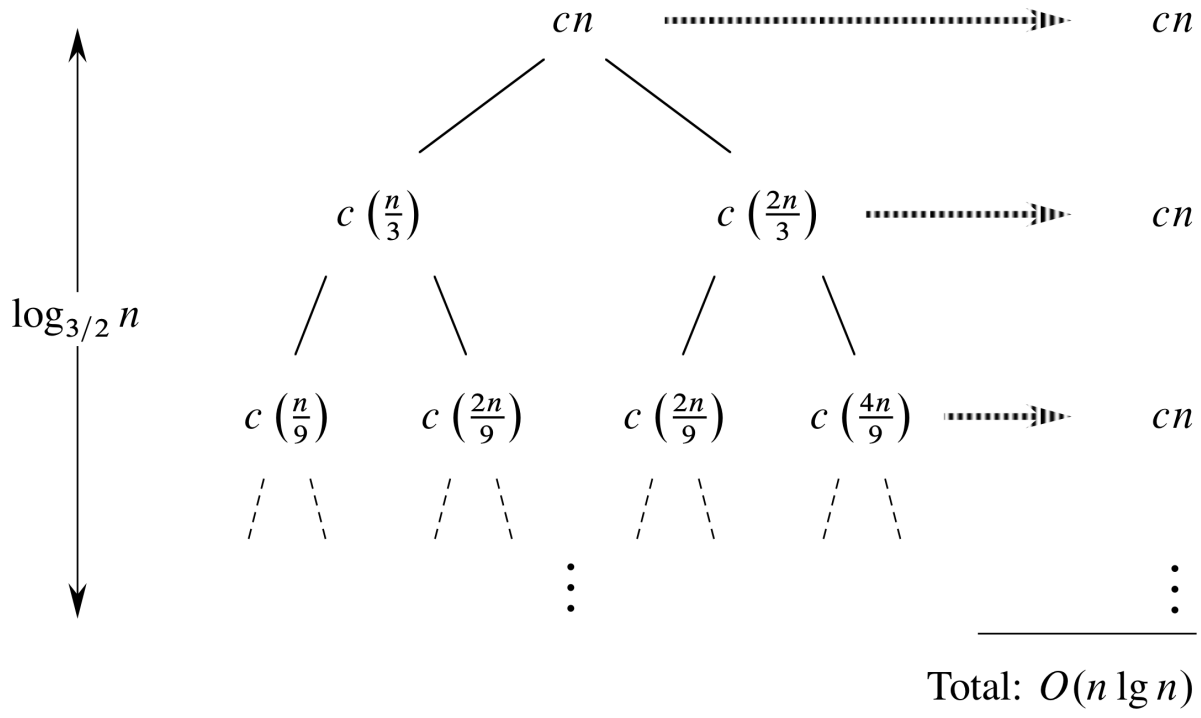
Divide-and-Conquer eksempler



(a) (b) (c) (d)

$$T(n) = 3T(n/4) + cn^2$$

Divide-and-Conquer eksempler



$$T(n) = T(n/3) + T(2n/3) + cn$$

Divide-and-Conquer eksempler

Ofte gælder én af flg.:

- Alle lag har lige stor sum, hvorved den samlede sum er antal lag (træets højde) gange denne sum.
- Lagenes sum aftager eksponentiel nedad gennem lagene, hvorved øverste lag dominerer.
- Lagenes sum vokser eksponentiel nedad gennem lagene (aftager eksponentielt opad gennem lagene), hvorved nederste lag dominerer. For at finde dette lags sum, skal man kende træets højde.

En generisk løsning af disse tre cases er præcis indholdet af bogens sætning side 94, kaldet Master Theorem.

De fleste rekursive algoritmer har en køretid, som beskrives ved en rekursionsligning, der passer ind i Master Theorem.

Ellers må man ræsonnere ud fra rekursionstræet (det kan man også gøre selv om Master Theorem kan bruges, naturligvis).

Master Theorem

Rekursionsligningen

$$T(n) = aT(n/b) + f(n)$$

har følgende løsning, hvor $\alpha = \log_b a$:

1. Hvis $f(n) = O(n^{\alpha-\epsilon})$ for et $\epsilon > 0$ gælder $T(n) = \Theta(n^{\alpha})$.
2. Hvis $f(n) = \Theta(n^{\alpha})$ gælder $T(n) = \Theta(n^{\alpha} \log n)$.
3. Hvis $f(n) = \Omega(n^{\alpha+\epsilon})$ for et $\epsilon > 0$ gælder $T(n) = \Theta(f(n))$.[1]

[1] I case 3 skal der også gælde, at der findes et $c < 1$ og et n_0 således at $a \cdot f(n/b) \leq c \cdot f(n)$ når $n \geq n_0$.