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 eksempler

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 een sorteret del (reelt arbejde).

Basistilfælde: $n \leq 1$ (trivielt).

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 een 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).

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 een 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).

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

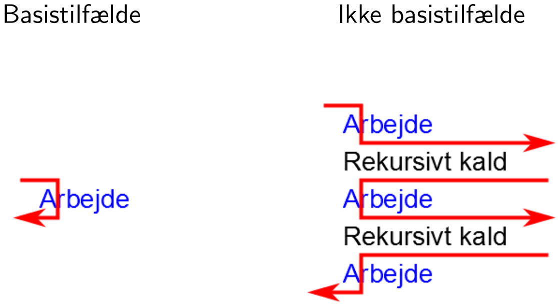
Hvis ikke basistilfælde:

- 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).

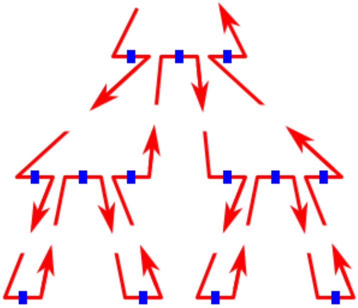
Divide-and-Conquer, udført arbejde

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



Divide-and-Conquer, udført arbejde

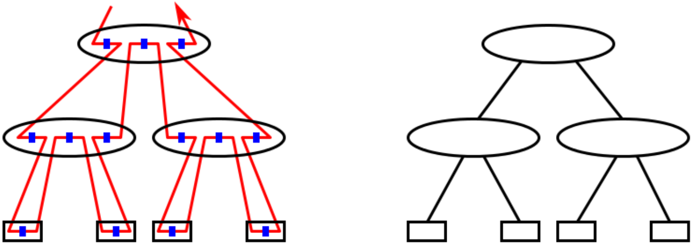
Globalt flow of control:



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

Rekursionstræer

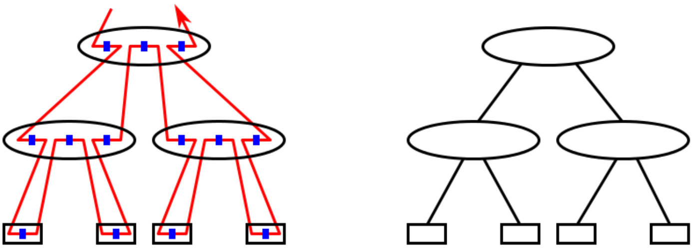
Globalt flow of control = rekursionstræer:



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 udførsel blandes ikke sammen.

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

Rekursionstræer og beregning af køretid

For en rekursiv algoritme, annoter knuderne i den rekursionstræ med

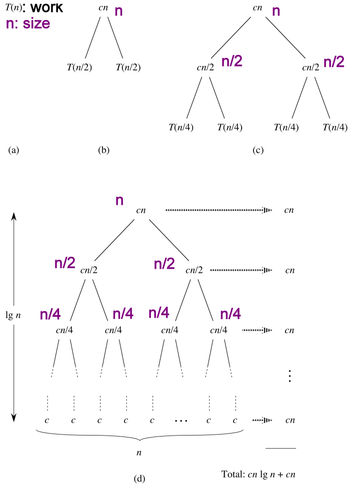
- Input størrelsen for kaldet til knude.
- Det resulterende arbejde udført i denne knude (men ikke i rekursive kald under knuden – de bliver selv annoteret med deres udførte arbejde).

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).

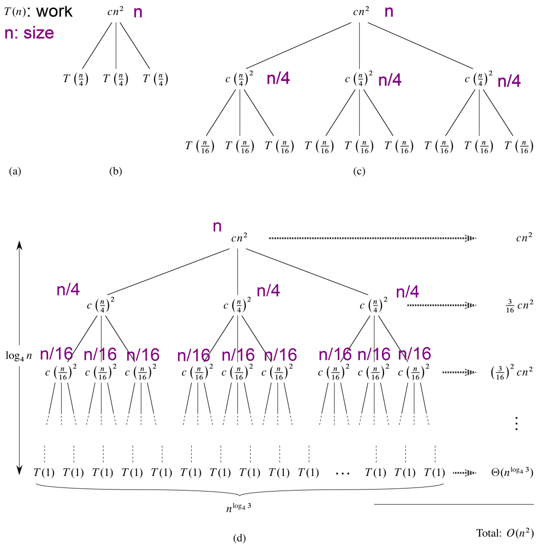
Eksempler følger.

Divide-and-Conquer eksempler



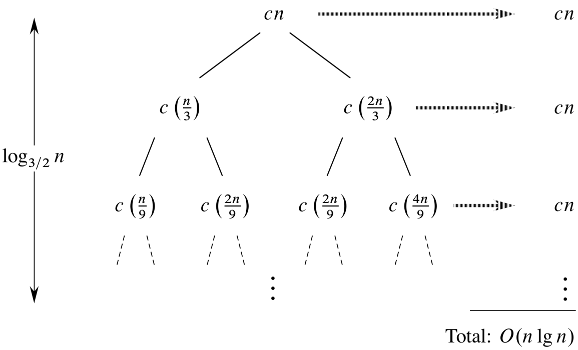
$$T(n) = 2T(n/2) + cn$$

Divide-and-Conquer eksempler



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

Divide-and-Conquer eksempler



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

Divide-and-Conquer eksempler

Ofte gælder een 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.

Divide-and-Conquer eksempler

Ofte gælder een 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).