

Divide-and-Conquer algoritmer

Divide-and-Conquer algoritmer

Det samme som [rekursive algoritmer](#).

Grundidé:

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](#).

Grundidé:

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).

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

For hver ny algoritme skal punkt 1 og punkt 3 udvikles. Punkt 2 er altid det samme. Løsning for basistilfældet skal også udvikles, men er som regel trivielt.

Generel struktur af Divide-and-Conquer kode

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 ét, og nogle har flere end to).

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).

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).

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 tredje eksempel er **inorder gennemløb** af et binært søgetræ: To rekursive kald (på højre og venste undertræ) med $O(1)$ arbejde imellem sig (udskrivning af nøgle i aktuelle knude).

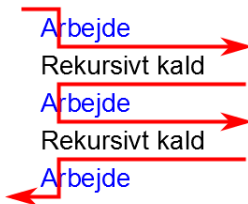
Hvad sker der når Divide-and-Conquer algoritmer udføres?

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

Basistilfælde

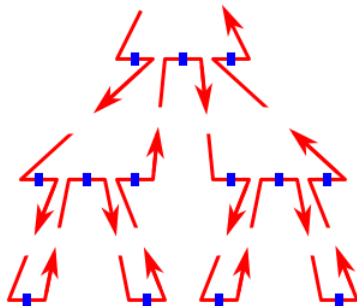


Ikke basistilfælde



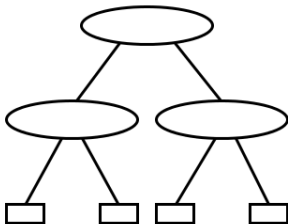
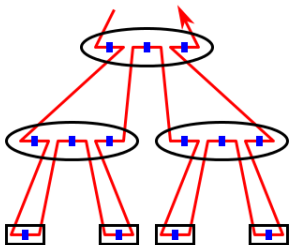
Hvad sker der når Divide-and-Conquer algoritmer udføres?

Her ses det globale flow of control, som opstår:



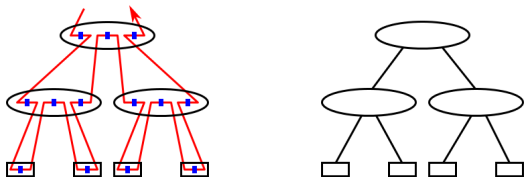
Rekursionstræer

Denne struktur for flow of control kan illustreres via træer, som har én knude for hvert kald af koden. Basistilfældene er blade, ikke-basistilfælde er indre knuder, og træets fanout (antal børn af knuder) er lig med antal rekursive kald i koden.



Disse træer kaldes rekursionstræer.

Hvad sker der når Divide-and-Conquer algoritmer udføres?



På ethvert givet tidspunkt er den samlede udførelse (den røde sti) nået til en eller anden knude v .

På dette tidspunkt er det kaldet, som v svarer til, der udføres handlinger fra. Alle kald svarende til knuder på stien fra v til roden er sat på pause. Hvert af disse kalds tilstand (indhold af deres variable, hvilken kommando de er nået til, m.m.) opbevares af computeren på en stak, så kaldenes udførelse ikke blandes sammen. Kald svarende til alle andre knuder i træet er enten helt færdige eller slet ikke begyndt.

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

Korrekthed af Divide-and-Conquer algoritmer

Korrekthed af Divide-and-Conquer algoritmer argumenteres nedefra og op i rekursionstræet:

- ▶ Argumentér for, at base case kald svarer korrekt (som regel trivielt).
- ▶ For et ikke-base case kald, argumentér for, at hvis de rekursive kald svarer korrekt, så vil dette sammen med det lokale arbejde gøre, at der svares korrekt i det aktuelle kald.

Dvs. at korrekthed af kald med store input følger af korrekthed af kald med mindre input samt handlingerne involveret i at konstruere en løsning for det store input ud fra løsningerne for de mindre input.

[Formelt kan man også sige, at korrekthed vises via induktion på inputstørrelsen. Basistilfældet for rekursionen er basistilfældet for induktionsbeviset.]

Selve argumentet er individuelt for hver algoritme. Det bliver som regel udviklet sammen med algoritmen (det er i praksis svært at få ideen til algoritmen uden at have ideen til, hvorfor den virker).

Rekursionsligninger

Vi vil nu kigge på **køretider** for rekursive algoritmer. Vi skal først se, hvordan disse kan beskrives ved **rekursionsligninger**.

Rekursionsligninger

Vi vil nu kigge på **køretider** for rekursive algoritmer. Vi skal først se, hvordan disse kan beskrives ved **rekursionsligninger**.

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

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

Rekursionsligninger

Vi vil nu kigge på **køretider** for rekursive algoritmer. Vi skal først se, hvordan disse kan beskrives ved **rekursionsligninger**.

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

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

Sidste linie er altid den samme og udelades derfor ofte. Ofte er det også underforstået, at vi bruger asymptotisk notation (så vi skriver $f(n)$ i stedet for $\Theta(f(n))$ og 1 i stedet for $\Theta(1)$).

Rekursionsligninger

Vi vil nu kigge på **køretider** for rekursive algoritmer. Vi skal først se, hvordan disse kan beskrives ved **rekursionsligninger**.

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

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

Sidste linie er altid den samme og udelades derfor ofte. Ofte er det også underforstået, at vi bruger asymptotisk notation (så vi skriver $f(n)$ i stedet for $\Theta(f(n))$ og 1 i stedet for $\Theta(1)$).

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

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

Køretid for Divide-and-Conquer algoritmer

En rekursionsligning beskriver strukturen af en rekursiv algoritme og giver en rekursiv beskrivelse af dens køretid (dvs. at $T(n)$ beskrives ud fra T på mindre inputstørrelse).

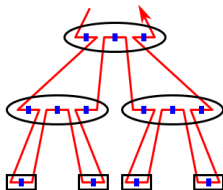
Men vi ønsker naturligvis også at finde et direkte udtryk (en funktion af n) for algoritmens køretid $T(n)$.

Køretid for Divide-and-Conquer algoritmer

En rekursionsligning beskriver strukturen af en rekursiv algoritme og giver en rekursiv beskrivelse af dens køretid (dvs. at $T(n)$ beskrives ud fra T på mindre inputstørrelse).

Men vi ønsker naturligvis også at finde et direkte udtryk (en funktion af n) for algoritmens køretid $T(n)$.

En rekursiv algoritmes samlede arbejde er lig summen af **lokalt arbejde** i algoritmens rekursionstræ:



Dvs. at vi ønsker at finde denne sum.

Rekursionstræsmetoden til beregning af køretid

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

- ▶ **Input størrelsen** for kaldet til knude.
- ▶ Det resulterende **arbejde udført i denne knude**.

Rekursionstræsmetoden til beregning af køretid

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

- ▶ **Input størrelsen** for kaldet til knude.
- ▶ Det resulterende **arbejde udført i denne knude**.

Find derefter summen af arbejdet i alle knuder på følgende måde:

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

Eksempler følger.

Eksempel 1

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

$$a = 2$$

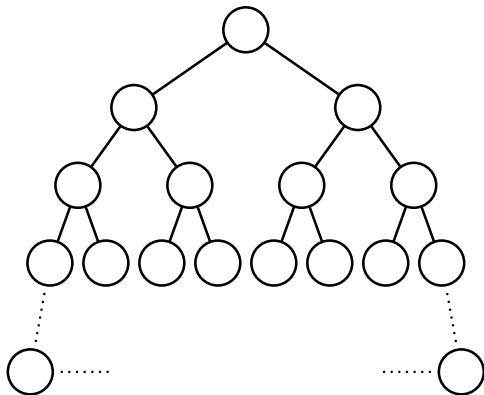
$$b = 2$$

$$f(n) = n$$

Eksempel 1

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

$$a = 2, b = 2, f(n) = n$$

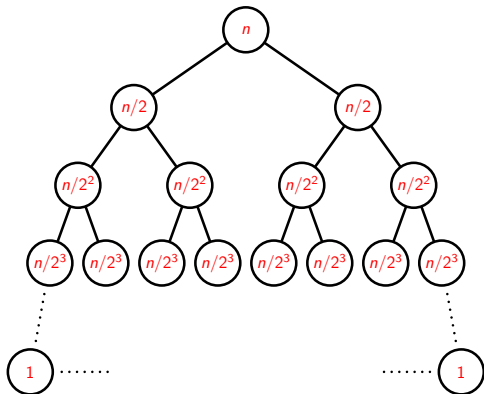


Tegn træ med fanout $a = 2$.

Eksempel 1

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

$$a = 2, b = 2, f(n) = n$$

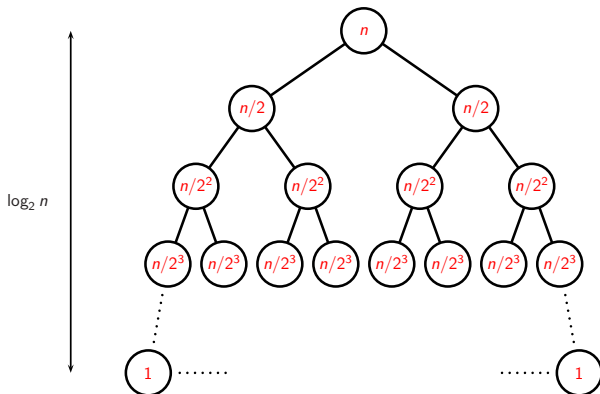


Indsæt input-størrelser i knuder ($= n/b^{\text{dybde}}$).

Eksempel 1

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

$$a = 2, b = 2, f(n) = n$$

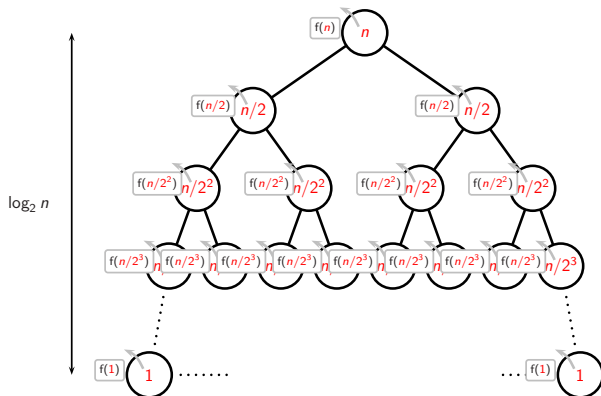


Find højden ud fra $n/b^{\text{højde}} = 1 \Leftrightarrow b^{\text{højde}} = n \Leftrightarrow \text{højde} = \log_b n$.

Eksempel 1

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

$$a = 2, b = 2, f(n) = n$$

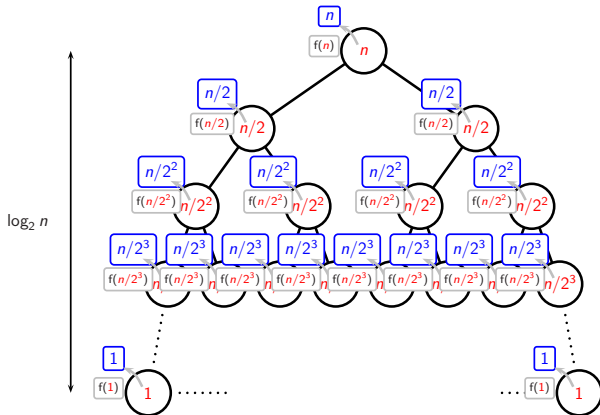


Anvend funktionen f for at konvertere input størrelse til arbejde.

Eksempel 1

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

$$a = 2, b = 2, f(n) = n$$

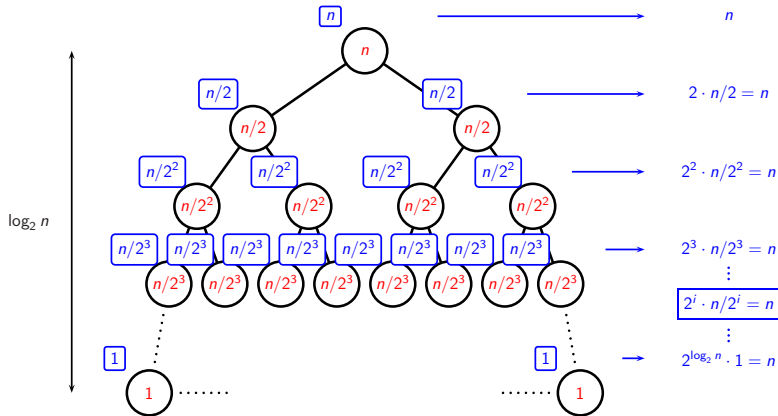


Anvend funktionen f for at konvertere input størrelse til arbejde.

Eksempel 1

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

$$a = 2, b = 2, f(n) = n$$

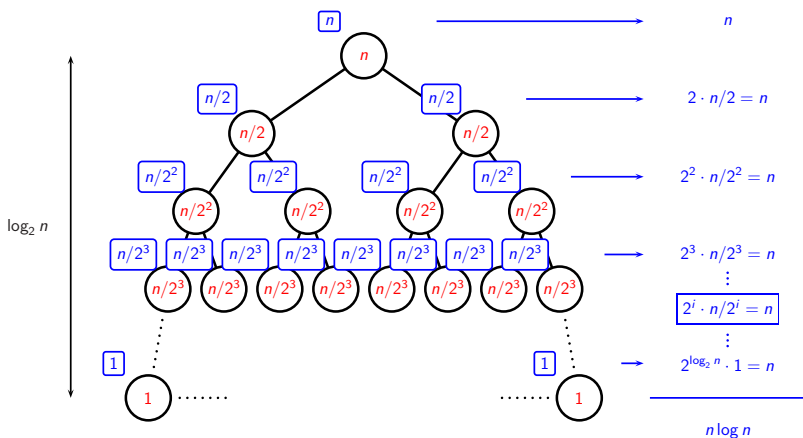


Sum hvert lag sammen og udtryk som funktion af lag nummeret i .

Eksempel 1

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

$$a = 2, b = 2, f(n) = n$$



Sum alle lag sammen for at finde samlet arbejde: $n \log n$.

Nogle matematiske facts til brug for de næste eksempler

Vi har tidligere vist at

$$1 + 2 + 4 + 8 + \dots + 2^k = 2^{k+1} - 1 = 2 \cdot 2^k - 1 = \Theta(2^k).$$

Mere generelt gælder følgende sætning for $c > 1$:

$$1 + c + c^2 + c^3 + \dots + c^k = \frac{c^{k+1} - 1}{c - 1} = c^k \cdot \frac{c - 1/c^k}{c - 1} = \Theta(c^k).$$

[Bevis (ikke pensum): Sæt $S = 1 + c + c^2 + c^3 + \dots + c^k$, så haves at $S(c - 1) = S \cdot c - S$ er lig $(c + c^2 + c^3 + \dots + c^{k+1}) - (1 + c + c^2 + c^3 + \dots + c^k) = c^{k+1} - 1$, så $S = (c^{k+1} - 1)/(c - 1)$.]

Denne sætning kan genfortælles som, at hvis elementerne i en sum ændrer sig eksponentielt, så er summen domineret af det største led (eksponentielt voksende: sidste led, eksponentielt faldende: første led).

Nogle matematiske facts til brug for de næste eksempler

Andre facts:

- ▶ $(a^b)^c = a^{bc} = (a^c)^b$
- ▶ $a^{\log_b n} = (b^{\log_b a})^{\log_b n} = (b^{\log_b n})^{\log_b a} = n^{\log_b a}$
- ▶ $\log_b a = \log_c a / \log_c b$ (f.eks. $= \ln a / \ln b$).

[Bevis (ikke pensum) for den sidste: $\log_b a = \log_c a / \log_c b \Leftrightarrow \log_c b \cdot \log_b a = \log_c a \Leftrightarrow c^{\log_c b \cdot \log_b a} = c^{\log_c a} \Leftrightarrow (c^{\log_c b})^{\log_b a} = a \Leftrightarrow b^{\log_b a} = a$]

Bemærk at det sidste fact kan skrives som $\log_b x = \frac{1}{\log_c b} \cdot \log_c x$, dvs. at logaritmer med forskellige grundtal (her b og c) er en konstant faktor fra hinanden: $\log_b x = \Theta(\log_c x)$.

Eksempel 2

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

$$a = 3$$

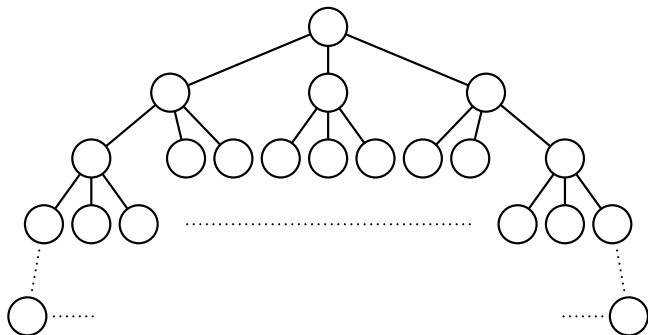
$$b = 2$$

$$f(n) = n$$

Eksempel 2

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

$$a = 3, b = 2, f(n) = n$$

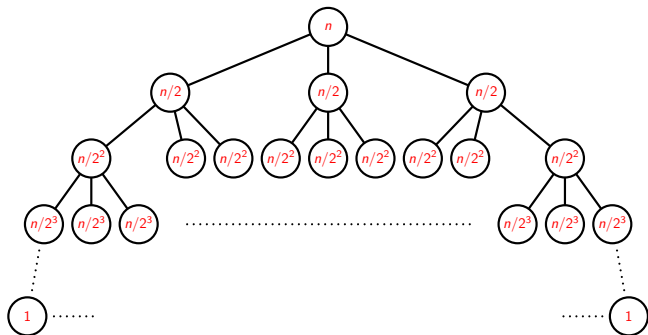


Tegn træ med fanout $a = 3$.

Eksempel 2

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

$$a = 3, b = 2, f(n) = n$$

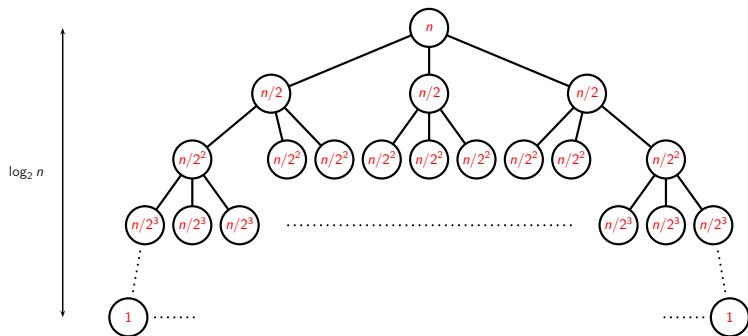


Indsæt input-størrelser i knuder ($= n/b^{\text{dybde}}$).

Eksempel 2

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

$$a = 3, b = 2, f(n) = n$$

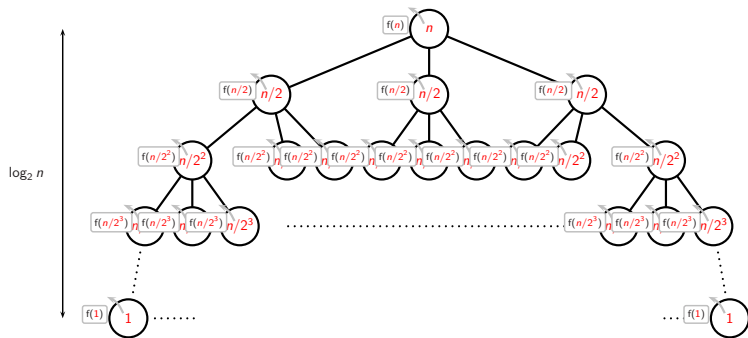


Find højden ud fra $n/b^{\text{højde}} = 1 \Leftrightarrow b^{\text{højde}} = n \Leftrightarrow \text{højde} = \log_b n$.

Eksempel 2

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

$$a = 3, b = 2, f(n) = n$$

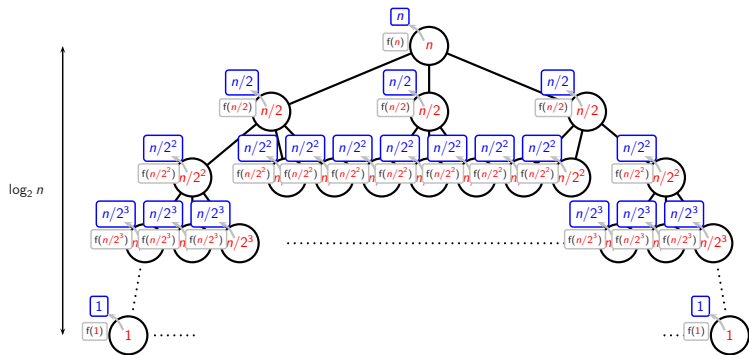


Anvend funktionen f for at konvertere input størrelse til arbejde.

Eksempel 2

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

$$a = 3, b = 2, f(n) = n$$

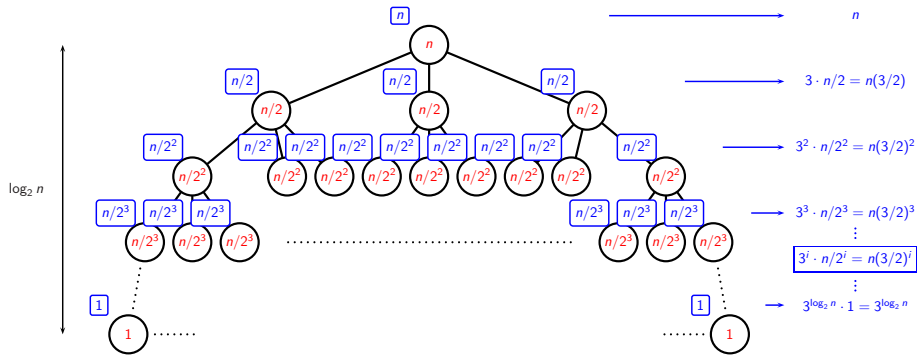


Anvend funktionen f for at konvertere input størrelse til arbejde.

Eksempel 2

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

$$a = 3, b = 2, f(n) = n$$

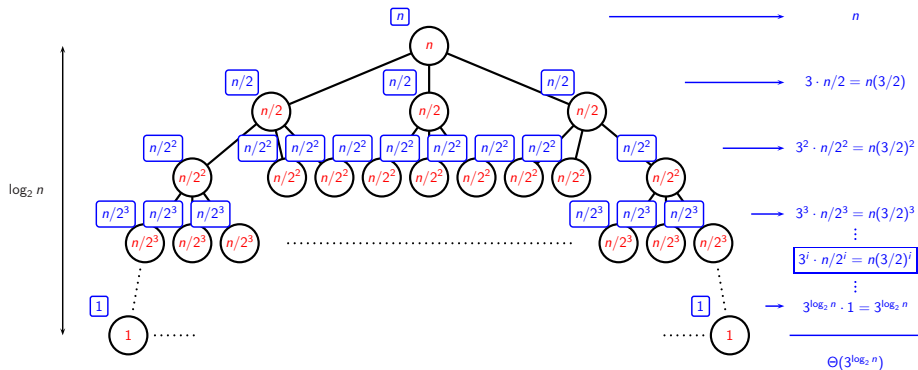


Sum hvert lag sammen og udtryk som funktion af lag nummeret i .

Eksempel 2

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

$$a = 3, b = 2, f(n) = n$$



Ekspontielt voksende \Rightarrow sidste lag dominerer \Rightarrow samlet arbejde er $\Theta(3^{\log_2 n}) = \Theta(n^{\log_2 3}) = \Theta(n^{\ln 3 / \ln 2}) = \Theta(n^{1.5849\dots})$.

Eksempel 3

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

$$a = 3$$

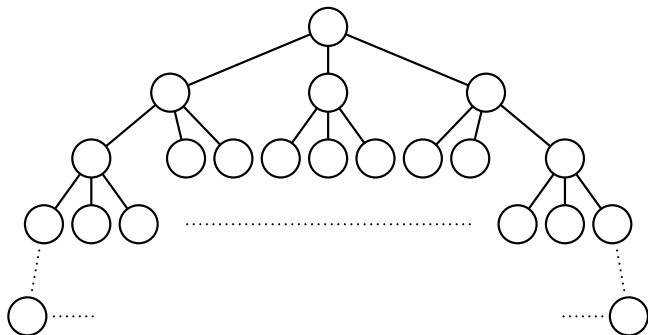
$$b = 4$$

$$f(n) = n^2$$

Eksempel 3

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

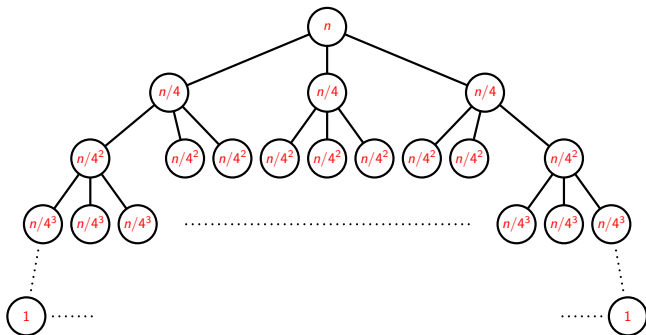
$$a = 3, b = 4, f(n) = n^2$$



Tegn træ med fanout $a = 3$.

Eksempel 3

$$T(n) = 3T(n/4) + n^2$$
$$a = 3, b = 4, f(n) = n^2$$

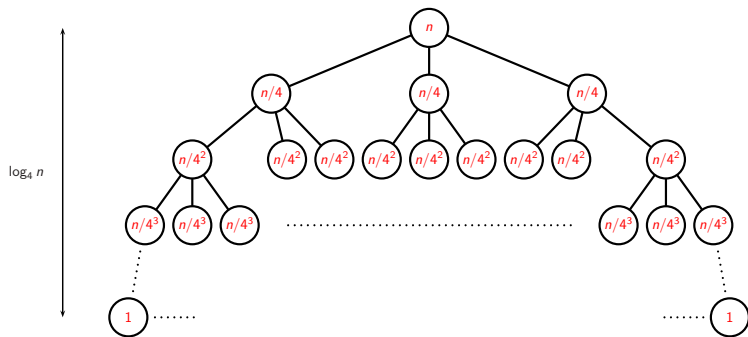


Indsæt input-størrelser i knuder ($= n/b^{\text{dybde}}$).

Eksempel 3

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

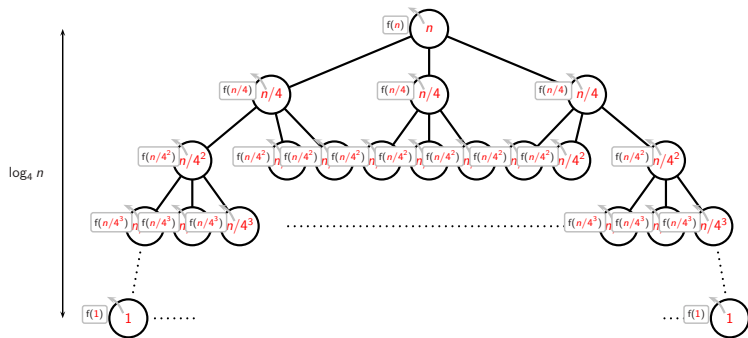
$$a = 3, b = 4, f(n) = n^2$$



Find højden ud fra $n/b^{\text{højde}} = 1 \Leftrightarrow b^{\text{højde}} = n \Leftrightarrow \text{højde} = \log_b n$.

Eksempel 3

$$T(n) = 3T(n/4) + n^2$$
$$a = 3, b = 4, f(n) = n^2$$

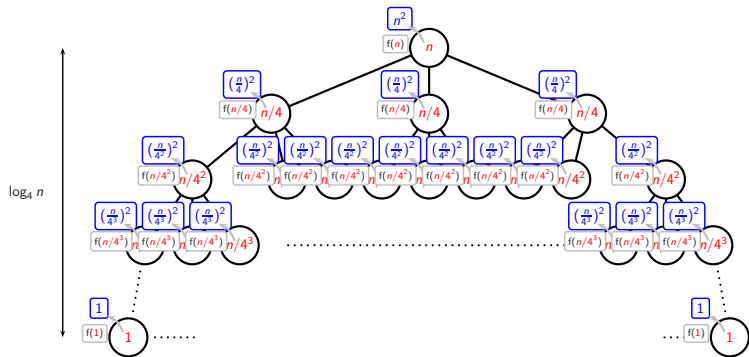


Anvend funktionen f for at konvertere input størrelse til arbejde.

Eksempel 3

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

$$a = 3, b = 4, f(n) = n^2$$

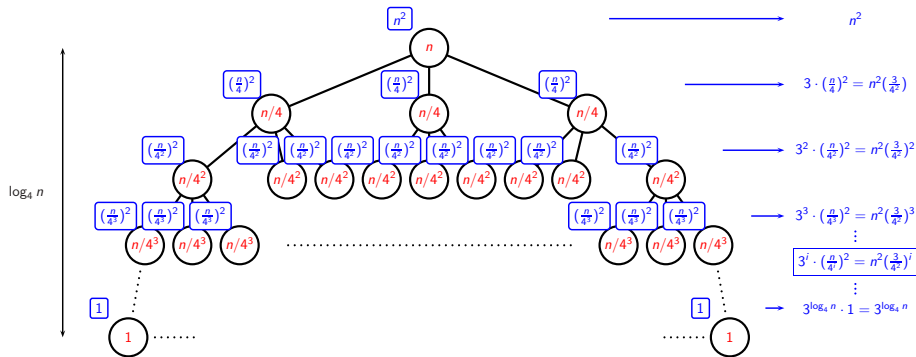


Anvend funktionen f for at konvertere input størrelse til arbejde.

Eksempel 3

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

$$a = 3, b = 4, f(n) = n^2$$

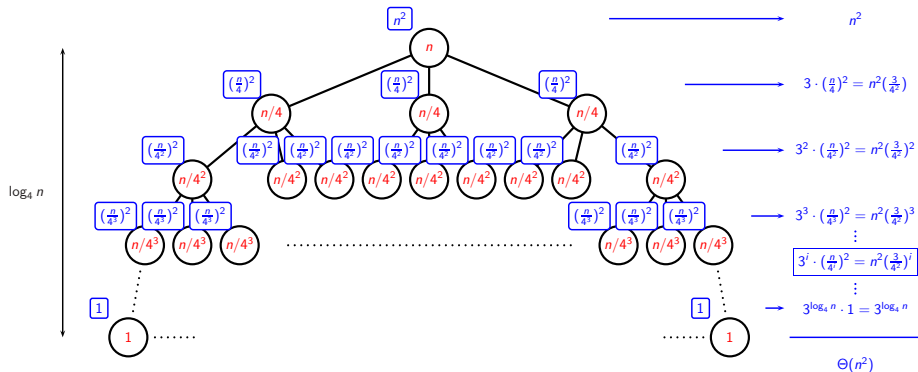


Sum hvert lag sammen og udtryk som funktion af lag nummeret i .

Eksempel 3

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

$$a = 3, b = 4, f(n) = n^2$$



Ekspontielt aftagende \Rightarrow første lag dominerer \Rightarrow samlet arbejde er $\Theta(n^2)$.

Divide-and-Conquer eksempler

Disse tre eksempler dækker mange tilfælde. Dvs. at ofte gælder én af følgende:

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

Divide-and-Conquer eksempler

Disse tre eksempler dækker mange tilfælde. Dvs. at ofte gælder én af følgende:

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

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.

Hvis rekursionsligningen ikke passer i Master Theorem, må man forsøge rekursionstræsmetoden direkte (det kan man også gøre selv om Master Theorem ikke 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 $T(n) = \Theta(f(n))$.

Ekstra betingelse: I case 3 skal der også gælde, at der findes et $c < 1$ og et n_0 som opfylder at $a \cdot f(n/b) \leq c \cdot f(n)$ når $n \geq n_0$.

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 $T(n) = \Theta(f(n))$.

Ekstra betingelse: I case 3 skal der også gælde, at der findes et $c < 1$ og et n_0 som opfylder at $a \cdot f(n/b) \leq c \cdot f(n)$ når $n \geq n_0$.

Case afgøres af forholdet mellem voksehastighederne for $f(n)$ og n^α . Er de ens, har vi case 2. Er $f(n)$ mindre (med mindst en faktor n^ϵ for et $\epsilon > 0$), har vi case 1. Er $f(n)$ større (med mindst en faktor n^ϵ for et $\epsilon > 0$), har vi case 3 (hvis ekstrabetingelsen er opfyldt.).

Master Theorem brugt på de tre eksempler

Eksempel 1: $T(n) = 2T(n/2) + n$

- ▶ $a = 2, b = 2, f(n) = n, \alpha = \log_2 2 = 1, f(n) = n = \Theta(n^\alpha) = \Theta(n^1)$
- ▶ Så vi er i case 2, så der gælder $T(n) = \Theta(n^\alpha \log n) = \Theta(n \log n)$.

Master Theorem brugt på de tre eksempler

Eksempel 1: $T(n) = 2T(n/2) + n$

- ▶ $a = 2, b = 2, f(n) = n, \alpha = \log_2 2 = 1, f(n) = n = \Theta(n^\alpha) = \Theta(n^1)$
- ▶ Så vi er i case 2, så der gælder $T(n) = \Theta(n^\alpha \log n) = \Theta(n \log n)$.

Eksempel 2: $T(n) = 3T(n/2) + n$

- ▶ $a = 3, b = 2, f(n) = n, \alpha = \log_2 3 = \ln(3)/\ln(2) = 1.5849\dots, f(n) = n = O(n^{\alpha-\epsilon}) = O(n^{1.5849\dots-\epsilon})$ for f.eks. $\epsilon = 0.1$.
- ▶ Så vi er i case 1, så der gælder $T(n) = \Theta(n^\alpha) = \Theta(n^{1.5849\dots})$.

Master Theorem brugt på de tre eksempler

Eksempel 1: $T(n) = 2T(n/2) + n$

- ▶ $a = 2, b = 2, f(n) = n, \alpha = \log_2 2 = 1, f(n) = n = \Theta(n^\alpha) = \Theta(n^1)$
- ▶ Så vi er i case 2, så der gælder $T(n) = \Theta(n^\alpha \log n) = \Theta(n \log n)$.

Eksempel 2: $T(n) = 3T(n/2) + n$

- ▶ $a = 3, b = 2, f(n) = n, \alpha = \log_2 3 = \ln(3)/\ln(2) = 1.5849\dots, f(n) = n = O(n^{\alpha-\epsilon}) = O(n^{1.5849\dots-\epsilon})$ for f.eks. $\epsilon = 0.1$.
- ▶ Så vi er i case 1, så der gælder $T(n) = \Theta(n^\alpha) = \Theta(n^{1.5849\dots})$.

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

- ▶ $a = 3, b = 4, f(n) = n^2, \alpha = \log_4 3 = \ln(3)/\ln(4) = 0.7924\dots, f(n) = n^2 = \Omega(n^{\alpha+\epsilon}) = \Omega(n^{0.7924\dots+\epsilon})$ for f.eks. $\epsilon = 0.1$.
- ▶ Så vi er i case 3, så der gælder $T(n) = \Theta(f(n)) = \Theta(n^2)$.

[I det sidste eksempel skal vi også checke den ekstra betingelse: Med $c = 3/16 < 1$ og $n_0 = 1$ opfyldes at $a \cdot f(n/b) = 3(n/4)^2 = 3/16 \cdot n^2 \leq c \cdot f(n) = c \cdot n^2$ for alle $n \geq n_0$.]

Master Theorem kan ikke altid bruges

Master Theorem kan ses som en forudberegnet rekursionstræsmetode, der dækker mange rekursionsligninger.

Mht. løsning af rekursionsligninger er det til eksamen i DM507 nok at kunne følgende:

- ▶ Bruge Master Theorem i situationer, hvor det kan anvendes.
- ▶ Genkende situationer, hvor Master Theorem ikke kan anvendes.

Master Theorem kan ikke altid bruges

Master Theorem kan ses som en forudberegnet rekursionstræsmetode, der dækker mange rekursionsligninger.

Mht. løsning af rekursionsligninger er det til eksamen i DM507 nok at kunne følgende:

- ▶ Bruge Master Theorem i situationer, hvor det kan anvendes.
- ▶ Genkende situationer, hvor Master Theorem ikke kan anvendes.

Af hensyn til det andet punkt giver vi nu nogle eksempler på rekursionsligninger, hvor Master Theorem *ikke* kan anvendes.

For de viste tilfældet *er* det faktisk muligt at gennemføre rekursionstræsmetoden og finde køretiden på den måde. Vi viser også hvordan (men dette er ikke pensum til eksamen).

Eksempel 4

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

Vi har her: $a = 2$, $b = 2$, $f(n) = n \log n$ og $\alpha = \log_2 2 = 1$. Da

$$\frac{n^\alpha}{f(n)} = \frac{n}{n \log n} = \frac{1}{\log n} \rightarrow 0 \text{ for } n \rightarrow \infty$$

er $n^\alpha = o(f(n))$, hvilket er det samme som

$$f(n) = \omega(n^\alpha).$$

Så vi kan ikke være i Case 2 af Master Theorem, da det kræver

$$f(n) = \Theta(n^\alpha).$$

Eksempel 4

Da

$$\frac{f(n)}{n^{\alpha+0.0001}} = \frac{n \log n}{n^{1.0001}} = \frac{\log n}{n^{0.0001}} \rightarrow 0 \text{ for } n \rightarrow \infty$$

er

$$f(n) = o(n^{\alpha+0.0001}).$$

Det samme vil gælde med ethvert $\epsilon > 0$ i stedet for 0.0001. Så vi kan ikke være i Case 3 af Master Theorem, da det kræver et $\epsilon > 0$ hvor

$$f(n) = \Omega(n^{\alpha+\epsilon}).$$

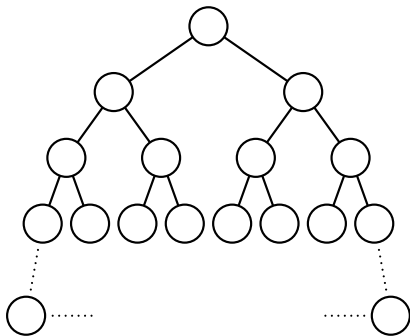
Denne rekursionsligning falder imellem Case 2 og Case 3: $f(n)$ vokser hurtigere end n^α , men ikke nok hurtigere til at være i Case 3. Vi kan derfor ikke bruge Master Theorem.

Vi viser nu, hvordan vi alligevel kan løse rekursionsligningen med rekursionstræsmetoden.

Eksempel 4

$$T(n) = 2T(n/2) + n \log n,$$

$$a = 2, b = 2, f(n) = n \log n$$

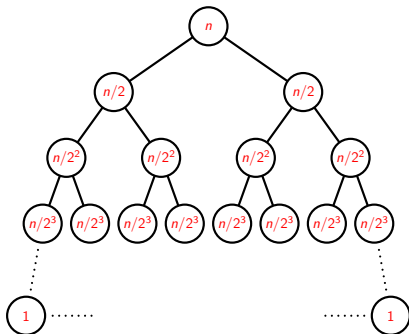


Tegn træ med fanout $a = 2$.

Eksempel 4

$$T(n) = 2T(n/2) + n \log n,$$

$$a = 2, b = 2, f(n) = n \log n$$

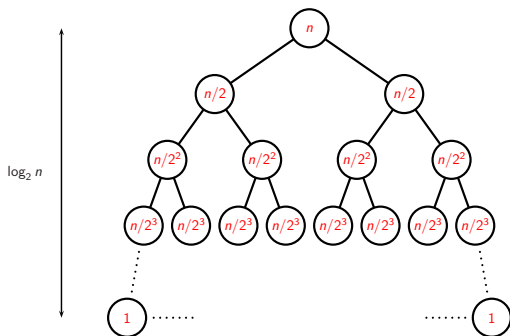


Indsæt input-størrelser i knuder ($= n/b^{\text{dybde}}$).

Eksempel 4

$$T(n) = 2T(n/2) + n \log n,$$

$$a = 2, b = 2, f(n) = n \log n$$

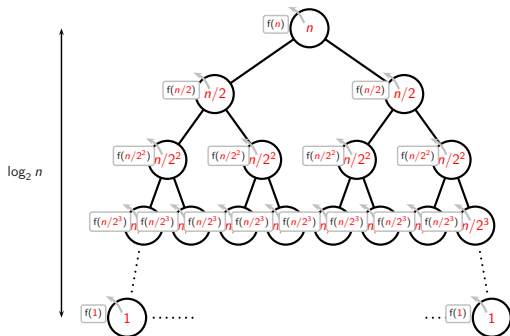


Find højden ud fra $n/b^{\text{højde}} = 1 \Leftrightarrow b^{\text{højde}} = n \Leftrightarrow \text{højde} = \log_b n$.

Eksempel 4

$$T(n) = 2T(n/2) + n \log n,$$

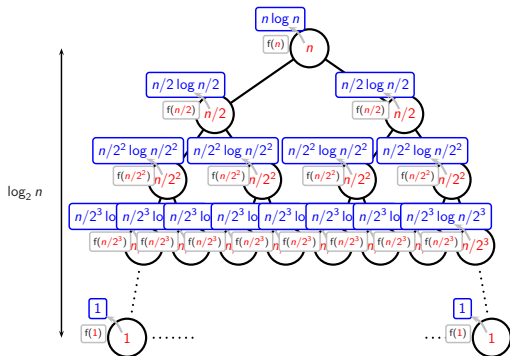
$$a = 2, b = 2, f(n) = n \log n$$



Anvend funktionen f for at konvertere input størrelse til arbejde.

Eksempel 4

$$T(n) = 2T(n/2) + n \log n, \quad a = 2, b = 2, f(n) = n \log n$$

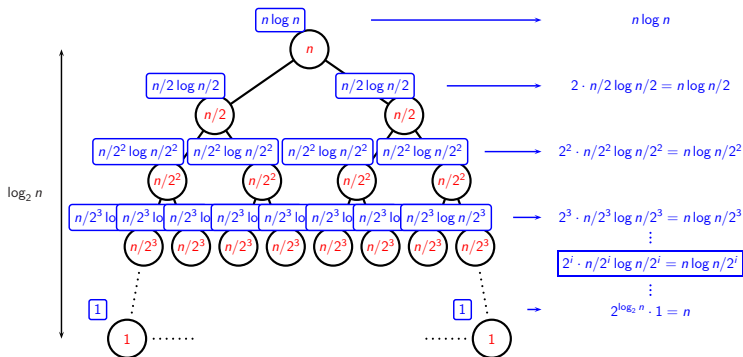


Anvend funktionen f for at konvertere input størrelse til arbejde.

Eksempel 4

$$T(n) = 2T(n/2) + n \log n,$$

$$a = 2, b = 2, f(n) = n \log n$$

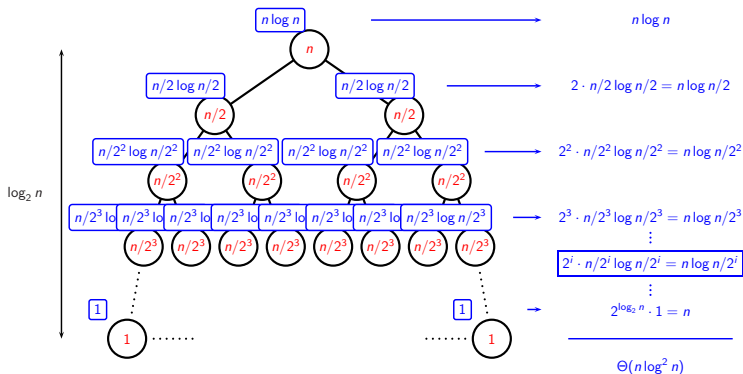


Sum hvert lag sammen og udtryk som funktion af lag nummeret i .

Eksempel 4

$$T(n) = 2T(n/2) + n \log n,$$

$$a = 2, b = 2, f(n) = n \log n$$



Sum alle lag sammen for at finde samlet arbejde. Svaret er $\Theta(n \log^2 n)$, hvilket vi vil argumentere for på næste side.

Eksempel 4

Vi har set at det i 'te lag i rekursionstræet laver $n \log(n/2^i)$ arbejde, hvilket kan skrives som $n(\log(n) - \log(2^i)) = n(\log(n) - i)$. Dette udtryk falder, når i stiger, så alle lag laver *højst* det samme arbejde som det første lag ($i = 0$). Det første lag laver $n \log n$ arbejde og der er $\log n$ lag i alt, så det samlede arbejde er $O(n \log^2 n)$.

Vi ser nu på den øverste halvdel af lagene, dvs. på lag i for $i = 1, 2, \dots, k = \frac{1}{2} \log n$.

Arbejdet for lag i falder, når i stiger, så hvert af disse lag laver *mindst* det samme arbejde som det sidste af dem (lag k). Arbejdet for lag k er

$$n(\log(n) - k) = n(\log(n) - \frac{1}{2} \log n) = \frac{1}{2} n \log n.$$

Der er derfor $k = \frac{1}{2} \log n$ lag, som hver laver mindst $\frac{1}{2} n \log n$ arbejde. Det samlede arbejde er derfor $\Omega(n \log^2 n)$.

I alt har vi vist at det samlede arbejde er $\Theta(n \log^2 n)$.

Eksempel 5

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

Denne rekursionsligning angiver køretiden for en rekursiv algoritme, som laver $O(n)$ lokalt arbejde og som laver to rekursive kald, et med størrelse $n/3$ og et med størrelse $2n/3$.

Denne rekursionsligning er desværre *ikke* af den type, som Master Theorem omhandler. For dem skal alle rekursive kald have samme størrelse (n/b):

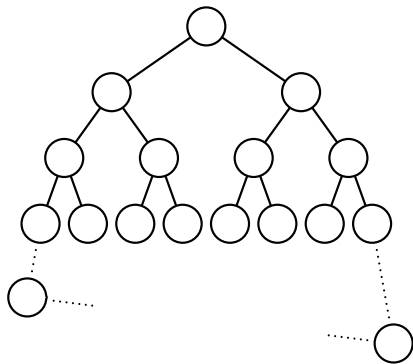
$$T(n) = aT(n/b) + f(n)$$

Vi kan derfor ikke bruge Master Theorem.

Vi viser nu, hvordan vi alligevel kan løse rekursionsligningen med rekursionstræsmetoden.

Eksempel 5

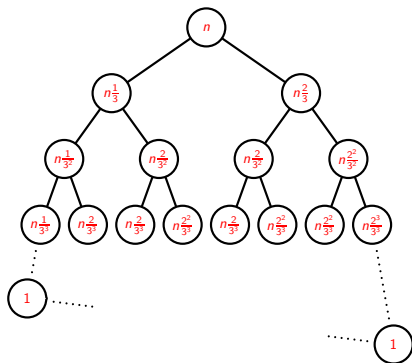
$$T(n) = T(n/3) + T(2n/3) + n$$



Tegn træ med fanout 2 (da der er to rekursive kald).

Eksempel 5

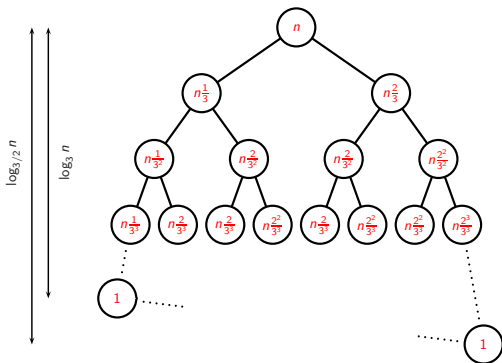
$$T(n) = T(n/3) + T(2n/3) + n$$



Indsæt input-størrelser i knuder: gang med $1/3$ (det ene barn) eller $2/3$ (det andet barn) når man går nedad.

Eksempel 5

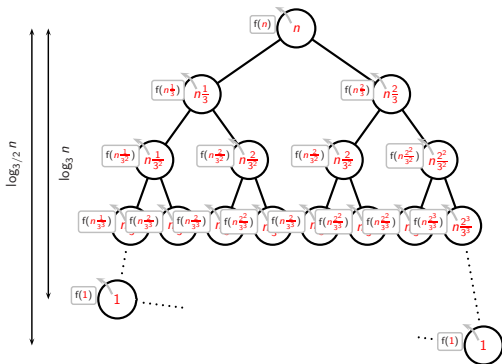
$$T(n) = T(n/3) + T(2n/3) + n$$



Find længste og korteste sti til blad: $n(\frac{1}{3})^k = 1 \Leftrightarrow n = 3^k \Leftrightarrow \log_3 n = k$
og $n(\frac{2}{3})^k = 1 \Leftrightarrow n = (\frac{3}{2})^k \Leftrightarrow \log_{3/2} n = k$.

Eksempel 5

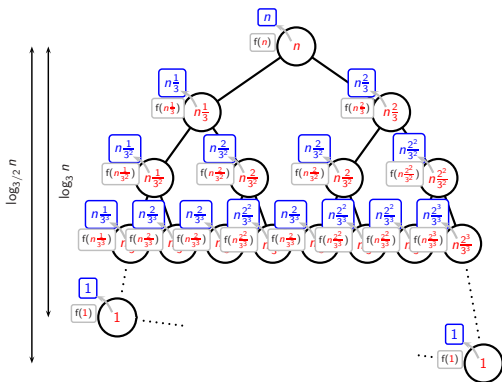
$$T(n) = T(n/3) + T(2n/3) + n$$



Anvend funktionen f for at konvertere input størrelse til arbejde.

Eksempel 5

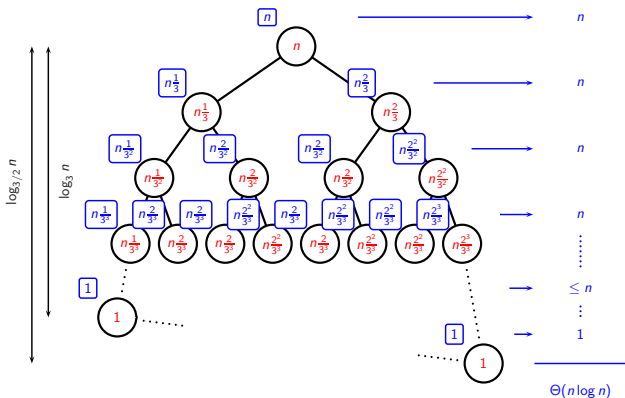
$$T(n) = T(n/3) + T(2n/3) + n$$



Anvend funktionen f for at konvertere input størrelse til arbejde.

Eksempel 5

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



Sum hvert lag sammen og sum derefter alle lag sammen for at finde samlet arbejde. Svaret er $\Theta(n \log n)$, hvilket vi vil argumentere for på næste side.

Eksempel 5

En knude med input af størrelse x laver $f(x) = x$ arbejde. Den har to børn med input af størrelse $x/3$ og $2x/3$, som derfor laver $f(x/3) = x/3$ og $f(2x/3) = 2x/3$ arbejde. Da $x/3 + 2x/3 = x$, ser vi at knudens arbejde er lig summen af den børns arbejde.

Deraf følger at summen af arbejde i lag k er lig summen af arbejdet i lag $k + 1$, hvis lag $k + 1$ er et fuldt lag (alle knuder i lag k har to børn).

Så alle fulde lag har samme sum af arbejde. For rodens lag er denne sum klart n , så for alle fulde lag er summen af arbejdet i laget lig n . Der er $\log_3 n$ fulde lag, så det samlede arbejde er $\Omega(n \log n)$.¹

For ikke-fulde lag kan summen kun være mindre (blade har ingen børn, så deres input-størrelse overføres ikke til næste lag), dvs. højst n . Der er $\log_{3/2} n$ lag i alt (fulde og ikke-fulde), så det samlede arbejde er $O(n \log n)$.¹

Alt i alt har vi vist at det samlede arbejde er $\Theta(n \log n)$.

¹Her bruges at logaritmer med forskellige grundtal er en konstant faktor fra hinanden, jvf. matematisk fact på side 15.

Floors og ceilings i rekursionsligninger

Der er en detalje, som vi indtil nu ikke har snakket om. Vi bruger Mergesort som eksempel.

Rekursionsligningen for Mergesort har vi skrevet som

$$T(n) = 2T(n/2) + n. \quad (1)$$

Men de to rekursive kald kan jo ikke være helt ens i størrelse hvis n er ulige. Dvs. at rekursionsligningen faktisk hedder

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n. \quad (2)$$

Betyder denne forskel noget for vores beregning af køretiden?

Svaret er nej, som vi skal se.

Floors og ceilings i rekursionsligninger

For en sti ned gennem rekursionstræet lader vi n_i betegne størrelsen af input på lag nummer i (rodens lag sættes til nummer 0).

For rekursionsligningen $T(n) = 2T(n/2) + n$ gælder:

$$n_0 = n$$

$$n_1 = n_0/2 = n/2$$

$$n_2 = n_1/2 = (n/2)/2 = n/2^2$$

$$n_3 = n_2/2 = (n/2^2)/2 = n/2^3$$

\vdots

$$n_i = n/2^i$$

Floors og ceilings i rekursionsligninger

Vi ser nu på rekursionsligningen $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$.

Eftersom

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1.$$

får vi:

$$n_0 = n$$

$$n_1 \leq \lceil n_0/2 \rceil < n_0/2 + 1 = n/2 + 1$$

$$n_2 \leq \lceil n_1/2 \rceil < n_1/2 + 1 < (n/2 + 1)/2 + 1 = n/2^2 + 1/2 + 1$$

$$n_3 \leq \lceil n_2/2 \rceil < n_2/2 + 1 < (n/2^2 + 1/2 + 1)/2 + 1 = n/2^3 + 1/2^2 + 1/2 + 1$$

⋮

$$n_i < n/2^i + 1/2^{i-1} + \dots + 1/2 + 1$$

Floors og ceilings i rekursionsligninger

Vi ser nu på rekursionsligningen $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$.

Eftersom

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1.$$

og vi får også:

$$n_0 = n$$

$$n_1 \geq \lfloor n_0/2 \rfloor > n_0/2 - 1 = n/2 - 1$$

$$n_2 \geq \lfloor n_1/2 \rfloor > n_1/2 - 1 > (n/2 - 1)/2 - 1 = n/2^2 - 1/2 - 1$$

$$n_3 \geq \lfloor n_2/2 \rfloor > n_2/2 - 1 > (n/2^2 - 1/2 - 1)/2 - 1 = n/2^3 - 1/2^2 - 1/2 - 1$$

\vdots

$$n_i > n/2^i - 1/2^{i-1} - \dots - 1/2 - 1$$

Floors og ceilings i rekursionsligninger

Så for rekursionsligningen $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$ har vi vist

$$n_i < n/2^i + (1/2^{i-1} + \dots + 1/2 + 1),$$

$$n_i > n/2^i - (1/2^{i-1} + \dots + 1/2 + 1).$$

Udtrykket i parentes er lig $1 + c + c^2 + \dots + c^{i-1}$ for $c = 1/2$, og er derfor (jvf. slide med matematiske facts på side 14) lig med:

$$\frac{(1/2)^i - 1}{1/2 - 1} = \frac{(1/2)^i - 1}{-1/2} = 2 - (1/2)^{i-1} < 2.$$

For rekursionsligningen $T(n) = 2T(n/2) + n$ viste vi

$$n_i = n/2^i.$$

Floors og ceilings i rekursionsligninger

Så i de to rekursionstræer for

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$$

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

afviger inputstørrelserne n_i i knuderne på et vilkårligt lag i kun med plus/minus 2.

For alle de funktioner f , som vi møder, gælder $f(n+2) = O(f(n))$.

For sådanne funktioner får vi samme asymptotiske køretid, når vi analyserer rekursionsligninger med og uden floors/ceilings.

Eksempler: for $f(n) = n$ gælder $f(n+2) = n+2 = O(n) = O(f(n))$. For $f(n) = n^2$ gælder $f(n+2) = (n+2)^2 = n^2 + 2n + 4 = O(n^2) = O(f(n))$, for $f(n) = \log n$ gælder $f(n+2) = \log(n+2) \leq \log 2n = 1 + \log n = O(\log n) = O(f(n))$, og for $f(n) = 2^n$ gælder $f(n+2) = 2^{n+2} = 2^2 \cdot 2^n = 4 \cdot 2^n = O(2^n) = O(f(n))$.