ИЗСЛЕДВАНЕ НА АЛГОРИТМИ

Асимптотични сравнения на функции

Определение 1: Нека f е реалнозначна функция, дефинирана в множеството на реалните или целите положителни числа. Функцията f се нарича *асимптотично неотрицателна*, когато $\exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : f(n) \geq 0$. Ако неравенството е строго, f се нарича *асимптотично положителна*.

Определение 2: За всяка асимптотично неотрицателна функция g въвеждаме следните класове от функции:

$$O(g) = \{f \mid \exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : 0 \leq f(n) \leq c.g(n)\}.$$

$O(g)$ е множеството от всички функции, които растат *асимптотично не по-бързо* от g.
Ще записваме това така: $f \in O(g)$, $f = O(g)$ или $f \preccurlyeq g$.

$$\Omega(g) = \{f \mid \exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : 0 \leq c.g(n) \leq f(n)\}.$$

$\Omega(g)$ е множеството от всички функции, които растат *асимптотично не по-бавно* от g.
Ще записваме това така: $f \in \Omega(g)$, $f = \Omega(g)$ или $f \succcurlyeq g$.

$$o(g) = \{f \mid \forall c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : 0 \leq f(n) < c.g(n)\}.$$

$o(g)$ е множеството от всички функции, които растат *асимптотично по-бавно* от g.
Ще записваме това така: $f \in o(g)$, $f = o(g)$ или $f \prec g$.

$$\omega(g) = \{f \mid \forall c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : 0 \leq c.g(n) < f(n)\}.$$

$\omega(g)$ е множеството от всички функции, които растат *асимптотично по-бързо* от g.
Ще записваме това така: $f \in \omega(g)$, $f = \omega(g)$ или $f \succ g$.

$$\Theta(g) = \{f \mid \exists c_1 > 0\ \exists c_2 > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : 0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)\}.$$

$\Theta(g)$ е множеството от всички функции, които растат *асимптотично еднакво* с g.
Ще записваме това така: $f \in \Theta(g)$, $f = \Theta(g)$ или $f \asymp g$.

Използва се и терминът *порядък* на асимптотично неотрицателна функция. Например
$f \in \Theta(g)$ се чете: "функциите f и g са от един и същи порядък";
$f \in o(g)$ се чете: "функцията f е от по-нисък порядък в сравнение с функцията g ".

Петте обозначения се наричат *асимптотични сравнения* или *сравнения по порядък*.
Те могат да се тълкуват като бинарни релации между асимптотично неотрицателни функции.
Непосредствено от определение 2 получаваме основните свойства на тези релации.

Свойство 1. Транзитивност: ако $f\ \sigma\ g$ и $g\ \sigma\ h$, то $f\ \sigma\ h$, за всяко $\sigma \in \{\prec, \succ, \preccurlyeq, \succcurlyeq, \asymp\}$.

Свойство 2. Θ, O и Ω са рефлексивни: $f\ \sigma\ f$ за всяко $\sigma \in \{\preccurlyeq, \succcurlyeq, \asymp\}$.

Свойство 3. Ω е антисиметрична: $f \succcurlyeq g$ и $g \succcurlyeq f \Leftrightarrow f \asymp g$. Същото важи за релацията O.

Свойство 4. Θ е симетрична: $f \asymp g \Rightarrow g \asymp f$.

Свойство 5. $f \preccurlyeq g \Leftrightarrow g \succcurlyeq f$; $f \prec g \Leftrightarrow g \succ f$.

Следващото свойство показва, че при събиране е важно само събираемото от най-висок порядък, а при умножение можем да пренебрегваме множителите, ограничени от положителни константи отгоре и отдолу (включително константните множители).

Свойство 6. а) $\max\{f,g\} \asymp f+g$. б) Ако $\exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : \mathbf{0 < c_1 \leq f(n) \leq c_2}$, то $\boldsymbol{gf \asymp g}$.

Доказателство: За всички достатъчно големи n важат неравенствата:

$$\frac{1}{2}f(n)+\frac{1}{2}g(n) \leq \max\{f(n),g(n)\} \leq f(n)+g(n) \Rightarrow \max\{f,g\} \asymp f+g;$$

$\boldsymbol{c_1 g(n) \leq f(n)\,g(n) \leq c_2\, g(n)}$, следователно $\boldsymbol{gf \asymp g}$.

Разполагаме с различни методи за сравняване на две функции. Самото определение 2 дава един метод, ала той е твърде тромав, особено при функции със сложен аналитичен израз.

Следващите две свойства са основата на друг, по-удобен метод: сравняване на функции с помощта на граничен преход.

Свойство 7. За асимптотично положителни функции f и g са в сила следната еквивалентности:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0 \Leftrightarrow f=o(g); \qquad \lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty \Leftrightarrow f=\omega(g).$$

Доказателство: За произволно $\varepsilon > 0$ и за всички достатъчно големи n имаме:

$$-\varepsilon < \frac{f(n)}{g(n)} < \varepsilon \Leftrightarrow 0 \leq f(n) < \varepsilon.g(n),$$

откъдето следва първата еквивалентност. Взимайки реципрочна стойност, получаваме втората.

Свойство 8. За асимптотично положителни функции f и g:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=c>0 \Rightarrow f=\Theta(g).$$

Доказателство: За произволно фиксирано ε, удовлетворяващо неравенствата $c > \varepsilon > 0$, и за всички достатъчно големи n важат следните формули:

$$c-\varepsilon < \frac{f(n)}{g(n)} < c+\varepsilon \Rightarrow 0 \leq (c-\varepsilon).g(n) \leq f(n) \leq (c+\varepsilon).g(n).$$

Сега свойството следва от определението на Θ при $\boldsymbol{c_1 = c-\varepsilon}$ и $\boldsymbol{c_2 = c+\varepsilon}$.

Обратното твърдение не е вярно. Например за функциите $f(n)=(2+\sin n).n$ и $g(n)=n$ имаме: $\forall n \geq 1: n \leq (2+\sin n).n \leq 3n \Rightarrow f=\Theta(g)$, но въпреки това не съществува границата $\lim\limits_{n\to\infty}\dfrac{(2+\sin n).n}{n}$.

Означение 1. Със символа $\lg n$ ще обозначаваме двоичен логаритъм, а не десетичен.

Означение 2. $\log_a n$ и $\log_b n$ за $a,b>1$ са равни по порядък, защото се различават само с константен делител:

$$\log_a n = \frac{\log_b n}{\log_b a}.$$

Затова често няма да обозначаваме основата. Например ще пишем $\Theta(\lg n)=\Theta(\log n)$ и ще използваме натурален логаритъм при диференциране (в правилото на Лопитал).

Свойство 9. Нека f и g са асимптотично положителни функции и $a > 1$. Тогава:
— ако $f \prec g$ и g клони към безкрайност, то $a^f \prec a^g$;
— ако $\log_a f \prec \log_a g$ и g клони към безкрайност, то $f \prec g$.

Доказателство:

$$1 = \lim_{n\to\infty} \frac{g(n) - f(n) + f(n)}{g(n)} = \lim_{n\to\infty} \frac{g(n) - f(n)}{g(n)} + \lim_{n\to\infty} \frac{f(n)}{g(n)} \Rightarrow$$

$$\Rightarrow \lim_{n\to\infty} \frac{g(n) - f(n)}{g(n)} = 1,$$

откъдето следва, че $g - f$ клони към плюс безкрайност. Ето защо

$$\lim_{n\to\infty} \frac{a^{f(n)}}{a^{g(n)}} = \lim_{n\to\infty} \frac{1}{a^{g(n)-f(n)}} = 0,$$

с което първата половина на свойството е доказана.

Втората половина е просто друга формулировка на първата.

Забележка: Тук е от особено значение, че g клони към безкрайност. В противен случай заключението може да не бъде вярно. Да разгледаме следния пример:

$$f(n) = \frac{1}{n}, \qquad g(n) = 1 - \frac{1}{n}.$$

Лесно се проверява, че $f \prec g$, обаче $2^f \asymp 2^g$.

Свойство 9 е основата на трети метод за сравняване на функции — чрез логаритмуване: вместо да сравняваме самите функции, сравняваме логаритмите им. Предимство на метода е, че след логаритмуване често е възможно да се опростят получените изрази. Недостатък е, че методът не може да се използва самостоятелно: той е само първата стъпка от решението на по-сложни задачи. След логаритмуване на функциите остава да сравним логаритмите им, което най-често се извършва с помощта на други методи (обикновено чрез граничен преход).

Следствие от свойство 9. Ако $f \asymp g$, то $\log f \asymp \log g$.

Когато извършваме асимптотични сравнения, често опростяваме аналитичните изрази, например с помощта на свойство 6 (някои други опростявания са показани в примерите по-долу). Всички такива опростявания могат да бъдат извършвани само на главния ред, но не и в аргумента на функция!

Например от свойство 6 следва, че $2n \asymp n$. Обаче не е вярно, че $10^{2n} \asymp 10^n$. Напротив, с граничен преход се доказва, че $10^{2n} \succ 10^n$.

Както показва следствието от свойство 9, функцията логаритъм е изключение, тоест имаме право да опростяваме изрази под знака на логаритъма. Например от $2n \asymp n$ следва, че $\log 2n \asymp \log n$. Причината е, че логаритмуването смалява разликите между функциите: то заменя степенуването с умножение, а умножението — със събиране.

Обратно, антилогаритмуването усилва разликите между функциите, затова нямаме право да опростяваме степенен показател по порядък. Нямаме право също да опростяваме по порядък аргумента на функция, за която не знаем дали усилва, или смалява разликите.

Свойство 10. $\forall a > 1\ \forall t > 0\ \forall \varepsilon > 0: \log_a^t n \prec n^\varepsilon$.

Доказателство: Пресмятаме границата $\lim\limits_{n\to\infty} \frac{\log_a n}{n^{\frac{\varepsilon}{t}}} = \lim\limits_{x\to\infty} \frac{\log_a x}{x^{\frac{\varepsilon}{t}}}$.

От правилото на Лопитал имаме:

$$\lim_{x\to\infty} \frac{\log_a x}{x^{\frac{\varepsilon}{t}}} = \lim_{x\to\infty} \frac{1}{x \ln a \frac{\varepsilon}{t} x^{\frac{\varepsilon}{t}-1}} = \lim_{x\to\infty} \frac{t}{\varepsilon \ln a\, x^{\frac{\varepsilon}{t}}} = 0.$$

Степенуваме:

$$\lim_{x\to\infty} \left(\frac{\log_a n}{n^{\frac{\varepsilon}{t}}}\right)^t = 0^t = 0 \Rightarrow \log_a^t n \prec n^\varepsilon.$$

Пример 1. Нека $p(x) = a_0 x^k + a_1 x^{k-1} + \ \dots\ + a_k$ е полином от степен k и $a_0 > 0$. Тогава $p(n) \asymp n^k$.

Доказателство:

$$\lim_{n\to\infty} \frac{p(n)}{n^k} = \lim_{n\to\infty} \left(a_0 + \frac{a_1}{n} + \ \dots\ + \frac{a_k}{n^k}\right) = a_0 > 0 \Rightarrow p(n) \asymp n^k.$$

Пример 2. За фиксирано $k \in \mathbb{N}$ е в сила асимптотичното равенство $\binom{n}{k} \asymp n^k$.

Доказателство:

$$\lim_{n\to\infty} \binom{n}{k}.\frac{1}{n^k} = \lim_{n\to\infty} \frac{n.(n-1)\dots(n-k+1)}{k!.n^k} = \lim_{n\to\infty} 1.\left(1-\frac{1}{n}\right)\dots\left(1-\frac{k-1}{n}\right).\frac{1}{k!} = \frac{1}{k!} \Rightarrow \binom{n}{k} \asymp n^k.$$

Пример 3. $(n+1)^n \asymp n^n$.

Доказателство:

$$\lim_{n\to\infty} \frac{(n+1)^n}{n^n} = \lim_{n\to\infty} \left(1+\frac{1}{n}\right)^n = e \Rightarrow (n+1)^n \asymp n^n.$$

Пример 4. Съществуват асимптотично несравними функции.

Доказателство: Нека $g(n) = n$, $f(n) = \begin{cases} 1\ , & n \text{ е четно} \\ n^2, & n \text{ е нечетно.} \end{cases}$

Да допуснем, че $f = O(g)$. Тогава $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0: 0 \leq f(n) \leq c.g(n)$, т.е. $0 \leq n^2 \leq c.n$ за всички достатъчно големи нечетни n, което не е вярно за $n > c$.

Да допуснем, че $g = O(f)$. Тогава $\exists c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0: 0 \leq g(n) \leq c.f(n)$, т.е. $0 \leq n \leq c$ за всички достатъчно големи четни n, което не е вярно за $n > c$.

Щом f и g са несравними чрез релацията O, те не са сравними и чрез другите релации.

Пример 5. Възможно е $f = O(g)$, без да е вярно нито едно от сравненията $f = o(g)$ и $f = \Theta(g)$.

Доказателство: Нека $g(n) = n$, $f(n) = \begin{cases} n, n \text{ е четно} \\ \frac{1}{n}, n \text{ е нечетно.} \end{cases}$

В сила е сравнението $f = O(g)$, тъй като $\forall n \geq 1: 0 \leq f(n) \leq g(n)$.

Да допуснем, че $f = o(g)$. Тогава $\forall c > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0: 0 \leq f(n) < c.g(n)$, което не е вярно за четно n и $c \leq 1$ (получава се $0 \leq n < c.n$).

Да допуснем, че $f = \Theta(g)$. Тогава $\exists c_1 > 0\ \exists c_2 > 0\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0: 0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)$ и при нечетно $n > \frac{1}{\sqrt{c_1}}$ се стига до противоречие: $0 \leq c_1.n \leq \frac{1}{n}$.

Формула на Стирлинг: $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$.

Следствие от формулата на Стирлинг: $n! \asymp \frac{n^n\sqrt{n}}{e^n}$.

Пример 6. $\lg(n!) \asymp n\lg n$.

Доказателство: От формулата на Стирлинг следва, че

$$\lg n! \approx \lg\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right) = \lg\sqrt{2\pi n} + n\lg n - n\lg e \asymp n\lg n.$$

Пример 7. $\binom{2n}{n} \asymp \frac{4^n}{\sqrt{n}}$.

Доказателство: Отново използваме формулата на Стирлинг:

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{\sqrt{4\pi n}\left(\frac{2n}{e}\right)^{2n}}{2\pi n\left(\frac{n}{e}\right)^{2n}} = \sqrt{\frac{1}{\pi n}}4^n \asymp \frac{4^n}{\sqrt{n}}.$$

Задача 1. Докажете, че $1 \prec \lg\lg n \prec \lg n \prec n \prec n\lg n \prec n^2 \prec n^3 \prec 2^n \prec n! \prec n^n$.

Решение:

1. $\lim\limits_{n\to\infty}\frac{1}{\lg\lg n} = 0 \Rightarrow 1 \prec \lg\lg n$.

2. $\lim\limits_{n\to\infty}\frac{\lg\lg n}{\lg n} = \lim\limits_{x\to\infty}\frac{\lg\lg x}{\lg x}$.

   Тази граница пресмятаме с помощта на правилото на Лопитал:

   $$\lim_{x\to\infty}\frac{\lg\lg x}{\lg x} = \lim_{x\to\infty}\frac{1}{\lg x}.\frac{(\lg x)'}{(\lg x)'} = 0 \Rightarrow \lg\lg n \prec \lg n.$$

3. $\lim\limits_{n\to\infty}\frac{\lg n}{n} = \lim\limits_{x\to\infty}\frac{\lg x}{x}$.

   Отново чрез правилото на Лопитал намираме

   $$\lim_{x\to\infty}\frac{\lg x}{x} = \lim_{x\to\infty}\frac{1}{x} = 0 \Rightarrow \lg n \prec n.$$

4. $\lim\limits_{n\to\infty}\frac{n}{n\lg n} = 0 \Rightarrow n \prec n\lg n$.

5. $\lim\limits_{n\to\infty}\frac{n\lg n}{n^2} = 0 \Rightarrow n\lg n \prec n^2$.

6. $\lim\limits_{n\to\infty}\frac{n^2}{n^3} = 0 \Rightarrow n^2 \prec n^3$.

7. Тъй като $\lg(n^3) = 3\lg n \prec n\lg 2 = \lg(2^n)$, то $n^3 \prec 2^n$. Тук използвахме свойство 9.

8. Тъй като $\lg(2^n) = n\lg 2 \prec n\lg n \asymp \lg(n!)$, то $2^n \prec n!$ (отново използвахме свойство 9).

9. $\forall n \in \mathbb{N}: 0 < n! \leq n^{n-1}$. Затова $\forall n \in \mathbb{N}: 0 < \frac{n!}{n^n} \leq \frac{1}{n}$. Следва, че $\lim\limits_{n\to\infty}\frac{n!}{n^n} = 0 \Rightarrow n! \prec n^n$.

Задача 2. Докажете, че $\sqrt[n]{n} \asymp 1$.

Доказателство: Използваме следната лема:

$$\lim_{x\to\infty} f(x) = a > 0 \Leftrightarrow \lim_{x\to\infty} \ln f(x) = \ln a$$

Пресмятаме

$$\lim_{n\to\infty} \ln \sqrt[n]{n} = \lim_{n\to\infty} \frac{\ln n}{n} = 0 \Rightarrow \lim_{n\to\infty} \frac{\sqrt[n]{n}}{1} = e^0 = 1 \Rightarrow \sqrt[n]{n} \asymp 1$$

Ако трябва да подредим много функции, е добре първо да налучкаме тяхното подреждане и едва тогава да сравняваме функциите по порядък. Така сравняваме само съседните функции вместо всяка двойка.

Самото налучкване става с помощта на следното правило.

Асимптотичен ред на функциите от неотрицателен порядък:

Функции от неотрицателен порядък се наричат асимптотично неотрицателните функции, които са ограничени отдолу от някаква положителна константа, поне от някое място нататък. Тези функции се подреждат от по-бавно към по-бързо растящи по следния начин:

1. ограничените отгоре функции (вкл. константите);
2. логаритмите и техните степени (т. нар. полилогаритми);
3. степенните функции;
4. показателните функции;
5. факториелите;
6. функциите от вида n^n;
7. суперекспонентите;

и тъй нататък.

Няма най-бързо растяща функция. За всеки две функции съществуват безброй други с междинен порядък.

Порядъците са повече от реалните (положителни) числа. Действително, за всяко реално (положително) число c съществува функция от порядък c: функцията n^c. Обратното не е вярно: свойство 10 показва, че логаритъмът расте по-бавно от всяка степенна функция с положителен показател, но по-бързо от функция с нулев показател. В този смисъл се казва, че логаритъмът е функция от безкрайно малък положителен порядък.

Задача 3. Подредете в асимптотично нарастващ ред функциите

$$\sqrt{2}^{\lg n},\quad n^3,\quad n!,\quad (\lg n)!,\quad \lg^2 n,\quad \lg n!,\quad 2^{2^n},\quad n^{\frac{1}{\lg n}},\quad \ln\ln n,\quad \left(\frac{3}{2}\right)^n,$$

$$n.2^n,\quad 4^{\lg n},\quad (n+1)!,\quad \sqrt{\lg n},\quad 2^{\sqrt{2.\lg n}},\quad n^{\lg\lg n},\quad \ln n,\quad 2^{\lg n},\quad (\lg n)^{\lg n}.$$

Решение: Правилно е следното подреждане:

$$n^{\frac{1}{\lg n}} \prec \ln\ln n \prec \sqrt{\lg n} \prec \ln n \prec \lg^2 n \prec 2^{\sqrt{2.\lg n}} \prec \sqrt{2}^{\lg n} \prec 2^{\lg n} \prec \lg n! \prec 4^{\lg n} \prec n^3 \prec (\lg n)! \prec$$

$$\prec (\lg n)^{\lg n} \asymp n^{\lg\lg n} \prec \left(\frac{3}{2}\right)^n \prec n.2^n \prec n! \prec (n+1)! \prec 2^{2^n}.$$

Доказателство: Сравняваме всеки две съседни функции.

1) $n^{\frac{1}{\lg n}} = n^{\log_n 2} = 2 \prec \ln\ln n$.

2) $\ln\ln n \prec \sqrt{\lg n}$, тъй като $\sqrt{\lg n} \asymp \sqrt{\ln n}$ и $\ln\ln n \prec \sqrt{\ln n}$, подобно на $\ln n \prec \sqrt{n}$.

3) $\sqrt{\lg n} \asymp \sqrt{\ln n} \prec \ln n$, тъй като $1 \prec \sqrt{\ln n}$.

4) $\ln n \prec \lg^2 n$, тъй като $\ln n \prec \ln^2 n \asymp \lg^2 n$.

5) Неравенството $\lg^2 n \prec 2^{\sqrt{2.\lg n}}$ се доказва чрез логаритмуване:

тъй като $\lg(\lg^2 n) = 2\lg\lg n \prec \sqrt{2.\lg n}\lg 2 = \lg\left(2^{\sqrt{2.\lg n}}\right)$, то $\lg^2 n \prec 2^{\sqrt{2.\lg n}}$.

6) Неравенството $2^{\sqrt{2.\lg n}} \prec \sqrt{2}^{\lg n}$ се доказва чрез логаритмуване:

тъй като $\lg\left(2^{\sqrt{2.\lg n}}\right) = \sqrt{2.\lg n}\lg 2 \prec \frac{1}{2}\lg n = \lg(\sqrt{n}) = \lg\sqrt{2}^{\lg n}$, то $2^{\sqrt{2.\lg n}} \prec \sqrt{2}^{\lg n}$.

7) $\sqrt{2}^{\lg n} = \sqrt{n} \prec n = 2^{\lg n}$.

8) $2^{\lg n} = n \prec n\lg n \asymp \lg n!$.

9) $\lg n! \asymp n\lg n \prec n^2 = 4^{\lg n}$.

10) $4^{\lg n} = n^2 \prec n^3$.

11) Неравенството $n^3 \prec (\lg n)!$ се доказва чрез логаритмуване:

тъй като $\lg n^3 = 3\lg n \prec \lg n\lg\lg n \asymp \lg(\lg n)!$, то $n^3 \prec (\lg n)!$.

12) $(\lg n)! \prec (\lg n)^{\lg n}$, подобно на $n! \prec n^n$.

13) Равенството $(\lg n)^{\lg n} = n^{\lg\lg n}$ следва от свойството на логаритъма: $a^{\log_b c} = c^{\log_b a}$.

14) Неравенството $n^{\lg\lg n} \prec \left(\frac{3}{2}\right)^n$ се доказва чрез логаритмуване:

тъй като $\lg\left(n^{\lg\lg n}\right) = \lg\lg n.\lg n \prec \lg^2 n \prec n\lg\frac{3}{2} = \lg\left(\frac{3}{2}\right)^n$, то $n^{\lg\lg n} \prec \left(\frac{3}{2}\right)^n$.

15) $\left(\frac{3}{2}\right)^n \prec 2^n \prec n.2^n$.

16) Неравенството $n.2^n \prec n!$ се доказва чрез логаритмуване:

тъй като $\lg(n.2^n) = \lg n + n\lg 2 \prec n\lg n \asymp \lg(n!)$, то $n.2^n \prec n!$.

17) Неравенството $n! \prec (n+1)!$ се доказва чрез граничен преход:

$$\lim_{n\to\infty}\frac{n!}{(n+1)!} = \lim_{n\to\infty}\frac{1}{n+1} = 0 \Rightarrow n! \prec (n+1)!.$$

18) Неравенството $(n+1)! \prec 2^{2^n}$ се доказва чрез логаритмуване:

тъй като $\lg(n+1)! \asymp (n+1).\lg(n+1) \prec 2^n\lg 2 = \lg\left(2^{2^n}\right)$, то $(n+1)! \prec 2^{2^n}$.

Коректност на алгоритми

Коректност на итеративни алгоритми доказваме най-често с помощта на ***инварианта на цикъл.*** Методът се състои от следните стъпки:

1. Формулираме използваема инварианта на цикъла. Инвариантата представлява твърдение относно променливите на алгоритъма, което е в сила при всяка проверка за край на цикъла. Инвариантата е използваема, когато от нея може да се изведе коректността на алгоритъма.
2. Доказваме инвариантата чрез една разновидност на метода на математическата индукция. Базата съответства на първото изпълнение на проверката за край (влизането в цикъла). Индуктивната стъпка (тук наричана поддръжка) установява запазването на истинността на инвариантата след всяко изпълнение на тялото на цикъла.
3. Завършек:
   а) Доказваме, че алгоритъмът рано или късно ще приключи работа (тоест че не се зацикля).
   б) Прилагаме доказаната инварианта към последната проверка за край на цикъла.
   Ако сме избрали използваема инварианта, ще получим тъкмо твърдението за коректност на алгоритъма.

Пример 1. Докажете, че следният алгоритъм връща 2^n.

```
Alg1(int n)
{
   int s = 1;
   for (int k = 0; k < n; k++)
      s = s * 2;
   return s;
}
```

Инварианта на цикъла: При всяка проверка на условието за край на цикъла `k < n` е в сила равенството $s = 2^k$.

Доказателство на инвариантата:

База: При първата проверка: $k = 0$, $s = 2^0 = 1$, следователно твърдението е вярно.

Поддръжка: Нека инвариантата е в сила при някоя проверка, която не е последна: $s = 2^k$. Да означим с прим стойностите, които променливите ще имат след още едно изпълнение на тялото на цикъла.

От наличието на командите `s = s * 2` и `k++` следва, че $s' = 2s$, $k' = k+1$. Ето защо $2^{k'} = 2^{k+1} = 2 . 2^k = 2s = s'$, тоест $s' = 2^{k'}$. Следователно инвариантата ще бъде изпълнена и при следващата проверка на условието за край на цикъла.

С това инвариантата е доказана.

Завършек: Тялото на цикъла се изпълнява точно n пъти. При последната проверка за край са в сила следните равенства:

$s = 2^k$ (от току-що доказаната инварианта);
$k = n$ (от условието за край на цикъла).

Следователно в този миг е изпълнено равенството $s = 2^n$. После се изпълнява командата `return s`, която връща текущата стойност на s, тоест 2^n.

Пример 2. Докажете, че следният алгоритъм връща сбора на елементите на масива $a[n]$:

```
Alg2(int a[n])
{
   int s = 0;
   for (int i = 0; i < n; i++)
      s = s + a[i];
   return s;
}
```

Инварианта на цикъла: При всяка проверка на условието за край на цикъла `i < n` е изпълнено равенството $s = a[0] + \ldots + a[i-1]$.

Пример 3. Коректност на сортирането чрез пряк избор.

```
Selection_sort(int a[n])
{
   for (int i = 0; i < n - 1; i++)
   {
      int m = i;
      for (int j = i + 1; j < n; j++)
         if (a[j] < a[m])
            m = j;
      if (m > i)
         swap(a[i], a[j]);
   }
}
```

Инварианта на външния цикъл: При всяка проверка на условието за край на цикъла `i < n - 1` елементите $a[0], \ldots, a[i-1]$ са сортирани и в тях се съдържат най-малките i стойности на масива.

Инварианта на вътрешния цикъл: При всяка проверка на условието за край на цикъла `j < n` елементите $a[0], \ldots, a[i-1]$ са сортирани, в тях се съдържат най-малките i стойности на масива и е изпълнено равенството $a[m] = \min\{a[i], \ldots, a[j-1]\}$.

Забележка: Доказателството на всяка от двете инварианти зависи от истинността на другата. Поддръжката на външната инварианта разчита на правилната работа на вътрешния цикъл (което се доказва с помощта на вътрешната инварианта); а тъй като вътрешната инварианта повтаря външната инварианта (и добавя още едно твърдение), то и базата, и поддръжката на вътрешната инварианта зависят от правилността на външната. Повторението не може да се избегне, защото трябва да изразим някак си това, че вътрешният цикъл не разваля постигнатото от предишните изпълнения на тялото на външния цикъл.

Никоя от двете инварианти не може да се докаже преди другата: получава се логически кръг. Това може да се избегне, като всички етапи от доказателството на вътрешната инварианта — както базата, така и поддръжката — бъдат вложени в поддръжката на външната инварианта. По този начин структурата на разсъждението точно отговаря на структурата на алгоритъма: на вложените цикли съответстват вложени доказателства.

Коректност на рекурсивен алгоритъм доказваме с ***математическа индукция*** по подходящ входен параметър или израз, образуван от входните параметри. Често се налага да използваме т. нар. ***силна индукция***. Ако избраният входен параметър не е от примитивен тип (тоест ако е масив, списък, граф и т.н.), доказателството се извършва с помощта на ***структурна индукция*** по подходящо избрана числова характеристика на структурата (дължина на масива или списъка, брой върхове или брой ребра на графа, дълбочина на дървото и т.н.).

И за рекурсивен алгоритъм трябва да се доказва (отделно от индукцията), че алгоритъмът рано или късно ще завърши (тоест че рекурсията не е бездънна).

Пример 4. Да се докаже, че следният алгоритъм за бързо степенуване връща x^n.

```
int exp_by_sqr(int x, unsigned int n)
{
   if (n == 0)
      return 1;
   int y = exp_by_sqr(x * x, n / 2); // Целочислено деление!
   if (n is odd)
      y = x * y;
   return y;
}
```

Доказателство: със силна индукция по n. Това е необходимо, защото индуктивната стъпка е от $n / 2$ към n, а не от $n - 1$ към n.

Алгоритъмът ще завърши някога, защото при всяко увеличаване на дълбочината на рекурсията стойността на n намалява поне двойно, а не съществува безкрайна строго намаляваща редица от цели положителни числа. Рано или късно n ще стане 0, т.е. рекурсията ще стигне до дъното.

Доказахме, че алгоритъмът ще върне някаква стойност. Сега ще докажем чрез силна индукция, че алгоритъмът ще върне именно x^n.

База: $n = 0$. Алгоритъмът връща 1, което е правилната стойност, тъй като всяко цяло число, повдигнато на нулева степен, дава единица (приемаме, че $0^0 = 1$).

Индуктивна стъпка: Нека твърдението е вярно за всички стойности на втория входен параметър, по-малки от някаква стойност $n > 0$. Ще докажем, че твърдението е вярно и при тази стойност n. Ако n е четно, алгоритъмът връща стойността на y, без да я умножава по x, т.е. $\left(x^2\right)^{n/2} = x^n$. Ако n е нечетно, алгоритъмът връща стойността на y след едно допълнително умножение по x, тоест $x\left(x^2\right)^{(n-1)/2} = x \,.\, x^{n-1} = x^n$.

Пример 5. Имаме кутия с 53 сини и 42 жълти пионки. Разполагаме с неограничен запас от жълти пионки извън кутията. Разглеждаме следната игра:

```
Докато в кутията има повече от една пионка:
   Изваждаме две случайно избрани пионки.
   Ако двете извадени пионки са с еднакъв цвят,
      добавяме в кутията една жълта пионка,
   иначе
      връщаме синята пионка в кутията.
```

Какъв ще бъде цветът на последната пионка в кутията?

Решение: Отначало в кутията има 95 пионки. На всеки ход вадим две пионки и добавяме една, т.е. броят на пионките в кутията намалява с единица. Ето защо играта завършва след 94 хода.

Инварианта на цикъла: При всяка проверка на условието за край на играта е в сила следното твърдение: в кутията има нечетен брой сини пионки.

Инвариантата се доказва с индукция по броя на ходовете. От това, че инвариантата е в сила и при последната проверка, следва, че последната пионка в кутията ще бъде синя.

Пример 6. Алгоритъм на Кадан за търсене на подмасив с максимален сбор.
"Подмасив" означава редица от последователни елементи, взети в същия ред.
Допуска се редицата да е празна; тогава сборът от нейните елементи е нула.

```
Kadane(int a[n])
{
   int c = 0, m = 0;
   for (int i = 0; i < n; i++)
   {
      if (a[i] + c > 0)
         c = a[i] + c;
      else
         c = 0;
      if (m < c)
         m = c;
   }
   return m;
}
```

Инварианта на цикъла: При всяка проверка на условието за край на цикъла `i < n` променливата c съдържа най-големия сбор сред подмасивите с последен елемент $a[i-1]$, а m съдържа най-големия сбор сред подмасивите с последен елемент $a[0], \ldots, a[i-1]$.

Забележка: Този алгоритъм намира и празни подмасиви (ако всички елементи са отрицателни). Алгоритъмът може лесно да се промени така, че да търси само непразни подмасиви:

```
Kadane2(int a[n])
{
   int c = a[0], m = a[0];
   for (int i = 1; i < n; i++)
   {
      if (c > 0)
         c = a[i] + c;
      else
         c = a[i];
      if (m < c)
         m = c;
   }
   return m;
}
```

Коректността на новата версия на алгоритъма се доказва с инвариантa, подобна на предишната. Трябва само да добавим към инвариантата уточнение, че става дума за *непразни* подмасиви.

Анализ на времевата сложност чрез сумиране

Някои важни суми:

$$\sum_{i=1}^{n} 1 = n, \qquad \sum_{i=a}^{b} 1 = b - a + 1, \qquad \sum_{i=1, i=i+k}^{n} 1 = \left\lceil \frac{n}{k} \right\rceil \approx \frac{n}{k},$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2), \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3),$$

$$\sum_{i=1}^{n} i^{\alpha} = \begin{cases} \Theta(n^{\alpha+1}) & , \alpha > -1 \\ \Theta(\log n) & , \alpha = -1 \\ \Theta(1) & , \alpha < -1. \end{cases}$$

Тези формули се извеждат по различни начини. Първите три се проверяват непосредствено. Четвъртата и петата се доказват по индукция. Шестата следва от интегралния метод за сумиране.

В следващите примери се търси времевата сложност на даден програмен фрагмент.

Пример 1.

```
for (int i = 1; i <= n; i++)
   print("a");
```

Времето за изпълнение на цикъла се изразява чрез сумата $\sum_{i=1}^{n} c = c \sum_{i=1}^{n} 1 = \Theta(n)$.

Тялото на цикъла се изпълнява n пъти, като всеки път се изразходва едно и също време $c > 0$.

В следващите примери ще заместим константата с 1, защото в задачите от този тип тя не влияе върху асимптотичното поведение на сумите.

Пример 2.

```
for (int i = 1; i <= n; i++)
   for (int j = 1; j <= n; j++)
      print("a");
```

Времето за изпълнение на този код се изразява с двойната сума $\sum_{i=1}^{n} \sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n^2 = \Theta(n^2)$.

Пример 3.

```
for (int i = 1; i <= n; i++)
   for (int j = 1; j <= i; j++)
      print("a");
```

Времето за изпълнение на този код се изразява с двойната сума $\sum_{i=1}^{n} \sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \Theta(n^2)$.

Пример 4.

```
for (int i = 1; i <= n; i++)
   for (int j = 1; j <= n; j += i)
      print("a");
```

Времето за изпълнение на този код е равно на $\sum_{i=1}^{n} \sum_{j=1, j=j+i}^{n} 1 \approx \sum_{i=1}^{n} \frac{n}{i} = n \sum_{i=1}^{n} \frac{1}{i} = \Theta(n \log n)$.

Пример 5.

```
for (int i = 1; i <= n; i++)
   for (int j = 1; j <= n; j++)
      if (i == j)
         for (int k = 1; k <= n; k++)
            print("a");
```

Първите два вложени цикъла определят време за изпълнение n^2. Третият цикъл се изпълнява само в случаите, когато $i = j$, а те са точно n на брой: $i = j = 1, i = j = 2$ и т.н. От своя страна, всяко изпълнение на третия цикъл изисква време n, така че за него имаме общо време n^2.

Цялото време на изпълнение е сборът от времето на външните два цикъла и времето на третия, вътрешния цикъл, тоест $n^2 + n^2 = 2n^2 = \Theta(n^2)$.

Пример 6.

```
for (int i = 1; i <= n; i++)
   for (int j = 1; j <= n; j += i)
      for (int k = 1; k <= n; k += i)
         print("a");
```

Времето за изпълнение на този код е равно на

$$\sum_{i=1}^{n}\sum_{j=1,j=j+i}^{n}\sum_{k=1,k=k+i}^{n}1 \approx \sum_{i=1}^{n}\sum_{j=1,j=j+i}^{n}\frac{n}{i} = \sum_{i=1}^{n}\left(\frac{n}{i}.\sum_{j=1,j=j+i}^{n}1\right) \approx \sum_{i=1}^{n}\frac{n^2}{i^2} = n^2.\sum_{i=1}^{n}\frac{1}{i^2} = \Theta(n^2).$$

Пример 7. Времева сложност на процедурата *BuildHeap*, построяваща двоична пирамида по даден масив с n елемента.

```
BuildHeap(числов масив с n елемента)
Постави елементите на масива в двоично дърво.
Обходи дървото от листата към корена и за всеки възел:
   провери дали числото във възела е по-голямо от някой пряк наследник;
   ако да — спускай числото до изчезване на дефекта.
```

След отстраняване на всички дефекти дървото представлява двоична пирамида.

Започваме със следното наблюдение: за $|x| < 1$ важи формулата

$$\frac{x}{(1-x)^2} = x\left(\frac{1}{1-x}\right)' = x.\left(\sum_{n=0}^{\infty}x^n\right)' = \sum_{n=1}^{\infty}nx^n.$$

За пълна двоична пирамида с височина h и $n = 2^{h+1} - 1$ елемента времето на $BuildHeap$ е

$$S(n) = \sum_{i=1}^{h}2^{h-i}.i = 2^h\sum_{i=1}^{h}\frac{i}{2^i} \le 2^h\sum_{i=1}^{\infty}\frac{i}{2^i} = 2^h.2 = O(2^h) = O(n).$$

За непълна пирамида времето остава $O(n)$, защото алгоритъмът извършва по-малко работа, отколкото за най-малката пълна двоична пирамида с брой елементи $s \ge n$. Тъй като $s \le 2n$, то търсената работа е $O(s)$, което не надвишава $O(2n) = O(n)$.

Долната граница $\Omega(n)$ се дължи на това, че алгоритъмът чете всеки елемент поне веднъж.

Окончателно, времевата сложност на алгоритъма е $\Theta(n)$.

Рекурентни уравнения

Във всички задачи се търси решението на даденото рекурентно уравнение.

Пример 1. $T(n) = 4T(n-2) + n.2^n + 4.3^n$.

Решение: Това е нехомогенно ***линейно-рекурентно уравнение.*** За него има специален метод. От хомогенната част правим ***характеристично уравнение,*** заменяйки индексите със степенни показатели: $x^2 - 4 = 0$. Уравнението има два прости корена: $x = \pm 2$. Затова мултимножеството от характеристичните корени е $\{2\,;\,-2\}_m$. Всеки характеристичен корен се взима толкова пъти, колкото е кратността на корена.

От нехомогенната част получаваме две мултимножества $\{2\,;\,2\}_m$ от $n.2^n$ и $\{3\}_m$ от 4.3^n. Всяко от тези мултимножества се състои от една от основите на показателните функции, взета толкова пъти, колкото е степента на съответния полином, увеличена с единица.

Обединяваме всички получени мултимножества в едно мултимножество:

$$\{2\,;\,2\,;\,2\,;\,-2\,;\,3\}_m.$$

Следователно общото решение има вида:

$$T(n) = c_1 3^n + c_2 2^n + c_3 n 2^n + c_4 n^2 2^n + c_5 (-2)^n = \Theta(3^n).$$

При определяне на порядъка приемаме за удобство, че $c_1 > 0$. Всеки характеристичен корен поражда толкова събираеми, колкото пъти участва в сборното мултимножество. Събираемите на един и същи характеристичен корен се различават по показателя на степенната функция, който започва от нула и нараства с единица за всяко следващо събираемо на същия корен.

Пример 2. $T(n) = 2T(n-1) - T(n-2)$.

Решение: Това линейно-рекурентно уравнение е хомогенно: то няма свободен член. Съставяме характеристично уравнение и го решаваме:

$$x^2 - 2x + 1 = 0, x_{1,2} = 1 \Rightarrow \{1\,;\,1\}_m \text{ е мултимножеството от корените.}$$

Общото решение на рекурентното уравнение има следния вид: $T(n) = c_1 1^n + c_2 n.1^n = \Theta(n)$.

Пример 3. $T(n) = T(n-1) + 1$.

Решение: Рекурентното уравнение може да се реши с помощта на характеристично уравнение, но тук има по-лесен начин: ***развиваме*** уравнението, т.е. прилагаме го към дясната му страна.

$$T(n) = T(n-1) + 1 = T(n-2) + 1 + 1 = \cdots = T(0) + \underbrace{1 + 1 + \cdots + 1}_{n \text{ пъти}} = T(0) + n = \Theta(n).$$

Пример 4. $T(n) = T(n-1) + \frac{1}{n}$.

Решение: Методът на характеристичното уравнение е неприложим, тъй като свободният член няма необходимия вид. Той трябва да бъде квазиполином — сбор от произведения на степенна и показателна функция, като показателите на степенните функции трябва да са неотрицателни цели числа. Затова прилагаме друг метод: развиване на рекурентното уравнение.

$$T(n) = T(0) + \frac{1}{n} + \frac{1}{n-1} + \frac{1}{n-2} + \ \ldots\ + \frac{1}{1} = T(0) + \sum_{i=1}^{n} \frac{1}{i} = \Theta(\ln n).$$

Пример 5. $T(n) = 2T(n-1) + \frac{1}{n}$.

Решение: Развиваме уравнението:

$$T(n) = 2^n T(0) + \frac{1}{n} + \frac{2}{n-1} + \frac{4}{n-2} + \ \ldots\ + \frac{2^{n-1}}{1} = 2^n . T(0) + 2^n \sum_{i=1}^{n} \frac{1}{i . 2^i} = \Theta(2^n).$$

При асимптотичната оценка използвахме неравенствата:

$$\sum_{i=1}^{n} \frac{1}{i . 2^i} \le \sum_{i=1}^{n} \frac{1}{2^i} \le \sum_{i=1}^{\infty} \frac{1}{2^i} = 1 \ \Rightarrow \ 2^n \sum_{i=1}^{n} \frac{1}{i . 2^i} \le 2^n.$$

Пример 6. $T(n) = \frac{n}{n+1} T(n-1) + 1$.

Решение: Развиваме уравнението:

$$T(n) = \frac{1}{n+1} T(0) + 1 + \frac{n}{n+1} + \frac{n-1}{n+1} + \ \ldots\ + \frac{2}{n+1} = \frac{1}{n+1} . T(0) + \frac{1}{n+1} \sum_{i=2}^{n+1} i =$$

$$= \frac{1}{n+1} . T(0) + \frac{1}{n+1} \left(\frac{(n+1)(n+2)}{2} - 1 \right) = \Theta(n).$$

Втори начин: Умножаваме даденото рекурентно уравнение по $n+1$:

$$(n+1)\, T(n) = n\, T(n-1) + n + 1.$$

Полагаме $P(n) = (n+1)\, T(n)$. Тогава $n\, T(n-1) = P(n-1)$ и уравнението приема вида:

$$P(n) = P(n-1) + n + 1.$$

Това вече е линейно-рекурентно уравнение. То може да се реши с характеристично уравнение или чрез развиване. Решението му е $P(n) = \Theta\left(n^2\right)$. От полагането намираме функцията $T(n)$:

$$T(n) = \frac{P(n)}{n+1} = \frac{1}{n+1} \Theta\left(n^2\right) = \Theta(n).$$

Пример 7. Средната сложност на бързото сортиране удовлетворява рекурентното уравнение

$$Q(n) = \frac{1}{n} . \sum_{i=0}^{n-1} [Q(i) + Q(n-1-i)] + c . n = \frac{2}{n} . \sum_{i=0}^{n-1} Q(i) + c . n.$$

Преобразуваме уравнението:

$$n . Q(n) = 2 . \sum_{i=0}^{n-1} Q(i) + c . n^2.$$

Уравнението е линейно, но е с ***променлива дължина на историята.*** Заместваме n с $n-1$:

$$(n-1) . Q(n-1) = 2 . \sum_{i=0}^{n-2} Q(i) + c . (n-1)^2.$$

Изваждаме последните две уравнения, след което изразяваме $Q(n)$:

$$Q(n) = \frac{n+1}{n} Q(n-1) + 2c - \frac{c}{n}.$$

Развиваме това уравнение:

$$Q(n) = (n+1) Q(0) + 2c(n+1) . \sum_{i=2}^{n+1} \frac{1}{i} - c(n+1) . \sum_{i=2}^{n+1} \frac{1}{i . (i-1)}.$$

Пресмятаме сумите:

$$Q(n) = (n+1) Q(0) + 2c(n+1) \sum_{i=2}^{n+1} \frac{1}{i} - cn = \Theta(n \log n).$$

Пример 8. $T(n) = 2T(\sqrt{n}) + 1$.

Решение: Полагаме $n = 2^{2^m}$, тоест $m = \lg\lg n$, и $S(m) = T(n)$. Следователно

$$S(m) = T(2^{2^m}) = 2T(2^{2^{m-1}}) + 1, \text{ тоест } S(m) = 2S(m-1) + 1.$$

Това рекурентно уравнение се решава чрез характеристично уравнение или чрез развиване. Решението му е $\Theta(2^m) = \Theta(2^{\lg\lg n}) = \Theta(\lg n)$.

Какво става, ако числото n не е от вида 2^{2^m} за никое цяло число m?

Тогава $n = 2^{2^x}$ за някое дробно x. Следователно $x \in [m, m+1]$ за някое цяло m. Тъй като функцията T е растяща, то за достатъчно големи m, т.е. за големи x, съществуват положителни числа c_1 и c_2, такива че

$$\frac{1}{2}.c_1.2^x \leq \frac{1}{2}c_1.2^{m+1} = c_1.2^m \leq T(2^{2^m}) \leq T(2^{2^x}) \leq T(2^{2^{m+1}}) \leq c_2.2^{m+1} = 2.c_2.2^m \leq 2.c_2.2^x.$$

Следователно за всички достатъчно големи n е в сила двойното неравенство

$$\frac{1}{2}.c_1.\lg n \leq T(n) \leq 2.c_2.\lg n,$$

откъдето следва, че $T(n) = \Theta(\lg n)$.

Пример 9. $T(n) = T(\sqrt{n}) + 1$.

Решение: Полагаме $n = 2^{2^m}$, тоест $m = \lg\lg n$, и $S(m) = T(n)$. Следователно

$$S(m) = T(2^{2^m}) = T(2^{2^{m-1}}) + 1, \text{ тоест } S(m) = S(m-1) + 1.$$

Това рекурентно уравнение се решава чрез характеристично уравнение или чрез развиване. Решението му е $\Theta(m) = \Theta(\lg\lg n)$.

Пример 10. Да се намери времевата сложност на следния рекурсивен алгоритъм:

```
Alg(int n)
{
   int s = 0;
   for (int i = 1; i < n; i++)
      s = s + 2 * Alg(i) + 1;
   return s;
}
```

Решение: Сложността на алгоритъма удовлетворява следното рекурентно уравнение:

$$T(n) = \sum_{i=1}^{n-1}\Big(T(i) + c\Big), \text{ тоест } T(n) = (n-1)c + \sum_{i=1}^{n-1} T(i).$$

където c е времето за извършване на всички операции от едно изпълнение на тялото на цикъла с изключение на рекурсивното извикване, $c > 0$. В уравнението заместваме n със $n-1$:

$$T(n-1) = (n-2)c + \sum_{i=1}^{n-2} T(i).$$

От първото уравнение изваждаме второто:

$$T(n) - T(n-1) = T(n-1) + c, \text{ тоест } T(n) = 2.T(n-1) + c.$$

Решението на това линейно-рекурентно уравнение е $T(n) = \Theta(2^n)$.

Пример 11. $T(n) = n\,T(n-1)$.

Решение: Това рекурентно уравнение е линейно, но не е линейно-рекурентно, защото съдържа ***променлив коефициент.*** Затова то не може да бъде решено чрез характеристично уравнение. Методът на характеристичното уравнение е приложим само когато следните изисквания са изпълнени едновременно:
— рекурентното уравнение е линейно;
— коефициентите му са постоянни;
— рекурентното уравнение има постоянна дължина на историята;
— свободният му член (ако има такъв) е квазиполином.

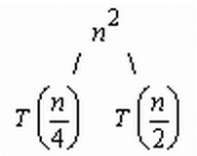
В тази задача е нарушено изискването за постоянни коефициенти, затова методът на характеристичното уравнение е неприложим. Вместо това развиваме рекурентното уравнение:

$$T(n) = nT(n-1) = n(n-1)T(n-2) = n(n-1)(n-2)T(n-3) = \cdots = n!\,T(0) = \Theta(n!).$$

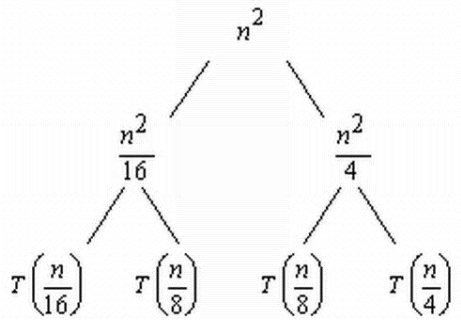
По принцип ***методът на развиването*** е приложим към всяко рекурентно уравнение, обаче не всякога е уместно да бъде прилаган. Най-добре е да се прилага към уравнения, чиято лява страна не съдържа нищо освен неизвестната функция $T(n)$, а в дясната страна неизвестната функция се среща само веднъж. В противен случай развиването на уравнението води до сложни изрази, които се опростяват трудно. За опростяването може да помогне проследяването на разклоненията на изразите с помощта на т. нар. ***дърво на рекурсията.***

Пример 12. $T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2$.

Решение: Дясната страна изглежда така.

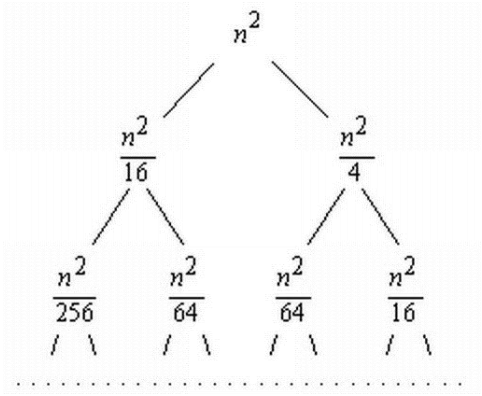


Ако развием $T\left(\frac{n}{4}\right)$ и $T\left(\frac{n}{2}\right)$, ще получим следващото равнище от дървото на рекурсията.



Неизвестната функция $T(n)$ е сборът от стойностите на всички върхове на дървото.

Продължавайки развиването, получаваме следното дърво:



Дървото на рекурсията е двоично, но не е балансирано: лявата част е по-плитка от дясната, тъй като при деление на 4 аргументът на функцията по-бързо стига до стойност единица, отколкото при деление на 2. Затова последните няколко равнища на дървото са непълни. В листата на дървото стоят събираеми от порядъка на $T(1)$.

Да означим с h височината на дървото, тоест дължината на най-дългия му клон. Това е клонът, който се спуска все надясно. По този клон аргументът на функцията, започвайки от n, стига до стойност 1 чрез последователни деления на 2. Ако се условим, че дробните частни се закръглят надолу, то броят на необходимите деления е $h = \lfloor \log_2 n \rfloor$.

Събираемите, които не са листа, сумираме слой по слой. От пълните слоеве получаваме

$$n^2 + \frac{5n^2}{16} + \frac{25n^2}{256} + \cdots$$

Тази сума е крайна, защото пълните слоеве са краен брой — поне $\lfloor \log_{3/2} n \rfloor + 1$ (по левия клон) и не повече от $h+1 = \lfloor \log_2 n \rfloor + 1$ (по десния клон на дървото). За да получим една горна граница на сбора на възлите, които не са листа, ще вземем сумата на безкрайния ред. Очевидно е, че събираемите образуват геометрична прогресия с частно $\frac{5}{16}$, затова безкрайната сума е $\frac{16n^2}{11}$.

И така, сборът на възлите, които не са листа, не надвишава числото $\frac{16n^2}{11}$ (горна граница). От друга страна, този сбор съдържа поне събираемото n^2 и всички събираеми са положителни, следователно търсеният сбор е най-малко n^2 (долна граница). Въпреки че не го пресметнахме точно, получените оценки отгоре и отдолу са достатъчни, за да направим извода, че сборът на възлите, които не са листа, е $\Theta\left(n^2\right)$.

Сборът от листата е равен на $T(1)$ по броя на листата, който не надхвърля $2^h \le n$. Затова този сбор е $O(n)$.

Окончателно, $T(n)$ е сумата на двата сбора, тоест

$$T(n) = O(n) + \Theta\left(n^2\right) = \Theta\left(n^2\right).$$

Рекурентни уравнения
от тип "разделяй и владей"

Този тип уравнения се решават чрез развиване. За да не бъдат извършвани едни и същи стъпки многократно, уравненията от този вид са били изследвани в обща форма. Получените резултати са обобщени в т. нар. **мастър-теорема:**

Нека $a \geq 1$, $b > 1$, $f(n)$ и $T(n)$ са асимптотично положителни функции и $T(n) = a.T\left(\frac{n}{b}\right) + f(n)$. Нека $k = \log_b a$. Тогава:

Случай 1: Ако $f(n) = O(n^{k-\varepsilon})$ за някое $\varepsilon > 0$, то $T(n) = \Theta(n^k)$.

Случай 2: Ако $f(n) = \Theta(n^k)$, то $T(n) = \Theta(n^k \log n)$.

Случай 3: Ако $f(n) = \Omega(n^{k+\varepsilon})$ за някое $\varepsilon > 0$
и $\exists c \in (0,1)\ \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : a.f\left(\frac{n}{b}\right) \leq c.f(n)$ (условие за регулярност на f),
то $T(n) = \Theta\big(f(n)\big)$.

Доказателството се извършва чрез развиване на рекурентното уравнение.

В следващите примери се иска да се реши даденото рекурентно уравнение.

Пример 1. $T(n) = 4T\left(\frac{n}{2}\right) + n$.

Решение: $k = 2$, можем да вземем $\varepsilon = 0{,}1$. От първия случай на мастър-теоремата: $T(n) = \Theta(n^2)$.

Пример 2. $T(n) = 4T\left(\frac{n}{\sqrt{2}}\right) + n^3$.

Решение: $k = 4$, можем да вземем $\varepsilon = 0{,}1$. От първия случай на мастър-теоремата: $T(n) = \Theta(n^4)$.

Пример 3. $T(n) = 2T\left(\frac{n}{2}\right) + n$.

Решение: $k = 1$, намираме се във втория случай на мастър-теоремата, затова $T(n) = \Theta(n \lg n)$.

Пример 4. $T(n) = T\left(\frac{n}{2}\right) + 1$.

Решение: $k = 0$, намираме се във втория случай на мастър-теоремата, затова $T(n) = \Theta(\lg n)$.

Пример 5. $T(n) = 2T\left(\frac{n}{8}\right) + n$.

Решение: $k = 1/3$, намираме се в третия случай на мастър-теоремата. Свободният член е регулярна функция, защото $\forall n \geq 1 : 2.\frac{n}{8} \leq \frac{1}{4}.n$, тоест можем да вземем $c = \frac{1}{4}$, $n_0 = 1$.
От заключението на третия случай на мастър-теоремата намираме $T(n) = \Theta(n)$.

Пример 6. $T(n) = 3T\left(\frac{n}{9}\right) + n\lg n.$

Решение: $k = 1/2$, намираме се в третия случай на мастър-теоремата. Свободният член е регулярна функция, защото $\forall n \geq 1: 3.\frac{n}{9}.\lg\frac{n}{9} \leq \frac{1}{3}n\lg n$, тоест можем да вземем $c = \frac{1}{3}$, $n_0 = 1$. От заключението на третия случай на мастър-теоремата намираме $T(n) = \Theta(n\lg n)$.

Пример 7. $T(n) = 8T\left(\frac{n}{4}\right) + n\lg n.$

Решение: $k = 3/2$. От първия случай на мастър-теоремата намираме $T(n) = \Theta\left(n^{3/2}\right)$.

Пример 8. $T(n) = 2\sqrt{2}T\left(\frac{n}{\sqrt{2}}\right) + n^3.$

Решение: $k = 3$. От втория случай на мастър-теоремата намираме $T(n) = \Theta(n^3\lg n)$.

Пример 9. $T(n) = 4T\left(\frac{n}{2}\right) + n^2\sqrt{n}.$

Решение: $k = 2$, намираме се в третия случай на мастър-теоремата. Свободният член е регулярна функция, защото $\forall n \geq 1: 4.\frac{n^2\sqrt{n}}{4\sqrt{2}} \leq \frac{1}{\sqrt{2}}n^2\sqrt{n}$, тоест можем да вземем $c = \frac{1}{\sqrt{2}}$, $n_0 = 1$. От третия случай на мастър-теоремата намираме $T(n) = \Theta\left(n^2\sqrt{n}\right)$.

Трите случая на мастър-теоремата не изчерпват всички възможности за свободния член. За някои от неразгледаните случаи съществуват други теореми.

Теорема: Нека $a \geq 1, b > 1, k = \log_b a, t \geq 0$, $T(n)$ е асимптотично положителна функция и $T(n) = a.T\left(\frac{n}{b}\right) + \Theta\left(n^k\lg^t n\right)$. Тогава $T(n) = \Theta\left(n^k\lg^{t+1} n\right)$.

Пример 10. $T(n) = 2T\left(\frac{n}{4}\right) + 2\sqrt{n}\lg^3 n.$

Решение: $k = 1/2$, $t = 3$, следователно $T(n) = \Theta\left(\sqrt{n}\lg^4 n\right)$.

Пример 11. $T(n) = T\left(\frac{n}{2}\right) + \lg n.$

Решение: $k = 0$, $t = 1$, следователно $T(n) = \Theta(\lg^2 n)$.

Пример 12. $T(n) = T\left(\frac{n}{2}\right) + \frac{1}{\lg n}.$

Решение: $k = 0$, $t = -1 < 0$, следователно теоремата не е приложима. Вместо това полагаме $n = 2^m$, $S(m) = T(2^m) = T(2^{m-1}) + \frac{1}{m} = S(m-1) + \frac{1}{m}$. Развиваме полученото уравнение за $S(m)$:

$$T(n) = S(m) = S(0) + 1 + \frac{1}{2} + \ldots + \frac{1}{m} = S(0) + \sum_{i=1}^{m}\frac{1}{i} \asymp \ln m \asymp \lg m = \lg\lg n.$$