Rekurencje

Jeśli algorytm zawiera wywołanie samego siebie, jego czas działania może być określony rekurencją.

Przykład: sortowanie przez scalanie:

$T(n) = \Theta(1)$ (dla $n = 1$)
$T(n) = 2\,T(n/2) + \Theta(n)$ (dla $n > 1$)

Rozwiązanie: $T(n) = \Theta(n \log_2 n)$

Metody rozwiązywania rekurencji:

– metoda podstawienia
– metoda iteracyjna
– metoda rekurencji uniwersalnej

Szczegóły techniczne

– zakładamy często, że argumentami funkcji są liczby całkowite

Przykład: ściśle powinniśmy zapisać dla sortowania przez scalanie:

$T(n) = \Theta(1)$ (dla $n = 1$)
$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$ (dla $n > 1$)

– pomijamy szczegółowe warunki brzegowe

Przykład: piszemy, że $T(n) = 2\,T(n/2) + \Theta(n)$ bez dokładnego podawania wartości dla małych n.

Metoda podstawienia

– odgadujemy postać rozwiązania
– wykazujemy słuszność rozwiązania przez indukcję

Przykład: obliczyć górne ograniczenie funkcji zadanej następującą zależnością rekurencyjną: $T(n) = 2T(\lfloor n/2 \rfloor) + n$.

1. Odgadujemy rozwiązanie: $T(n) = O(n \log n)$.

2. Musimy udowodnić, że $T(n) \leq c\, n \log n$ dla pewnej stałej c i $n > n_0$.

Przyjmujemy założenie, że ograniczenie zachodzi dla $\lfloor n/2 \rfloor$, tzn. że zachodzi $T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \log \lfloor n/2 \rfloor$.

Podstawiamy do równania rekurencyjnego:

$$
\begin{aligned}
T(n) &\leq 2\,c\lfloor n/2\rfloor \log\lfloor n/2\rfloor + n \\
&\leq c\,n \log\lfloor n/2\rfloor + n \\
&= c\,n \log n - c\,n \log 2 + n \\
&= c\,n \log n - c\,n + n \quad \text{(niech } c \geq 1\text{)} \\
&\leq c\,n \log n
\end{aligned}
$$

Sprawdzamy dla $n = 2$: $T(2) = 4 \leq c\ 2 \log 2$

$n = 3$: $T(3) = 5 \leq c\ 3 \log 3$

Dla $c \geq 2$ warunki powyższe są spełnione.

(warunek dla n = 1: $T(1) \leq c\ 1 \log 1 = 0$ – nierówność nie jest spełniona, ale nam wystarczy wykazać że jest spełniona dla wszystkich $n > n_0$).

Odgadywanie rozwiązania:

– podstawiamy postać rozwiązania podobną do rozwiązania już znanej rekurencji.

Przykład: $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$.

Dla dużych n, różnica między $T(\lfloor n/2 \rfloor + 17)$ a $T(\lfloor n/2 \rfloor)$ nie jest duża: obydwie rekurencje dzielą się prawie równo na połowy. Zgadujemy więc, że rozwiązanie wynosi $T(n) = O(n \log n)$, co można udowodnić korzystając np. z metody podstawiania.

Inny sposób: szacujemy zgrubnie górne i dolne oszacowanie (np. $T(n) = \Omega(n)$ oraz $T(n) = O(n^2)$), stopniowo poprawiając dolne i górne oszacowanie, aż dojdziemy do otrzymania rozwiązania dokładnego.

Metoda iteracyjna

Idea: rozwijanie (iterowanie) rekurencji i wyrażenie jej jako sumy składników zależnych tylko od n oraz warunków brzegowych.

Przykład: $T(n) = 3T(\lfloor n/4 \rfloor) + n$.

Iterujemy:

$$
\begin{aligned}
T(n) &= n + 3T(\lfloor n/4 \rfloor) = \\
&= n + 3(\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor)) = \\
&= n + 3(\lfloor n/4 \rfloor + 3(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor))) = \\
&= n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27T(\lfloor n/64 \rfloor) =
\end{aligned}
$$

i-ty składnik ciągu: $3^i \lfloor n/4^i \rfloor$. Iterowanie kończymy, gdy $n = 1$ lub $\lfloor n/4^i \rfloor = 1$ (równoważnie: gdy i przekracza $\log_4 n$).

Stwierdzamy, że suma odpowiada malejącemu szeregowi geometrycznemu:

$$T(n) \leq n + 3n/4 + 9n/16 + 27n/64 + \ldots + 3^{\log_4 n}\Theta(1) \leq$$
$$\leq n \sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i + n^{\log_4 3}\,\Theta(1) = 4n + o(n) = O(n)$$

Skorzystaliśmy z własności:

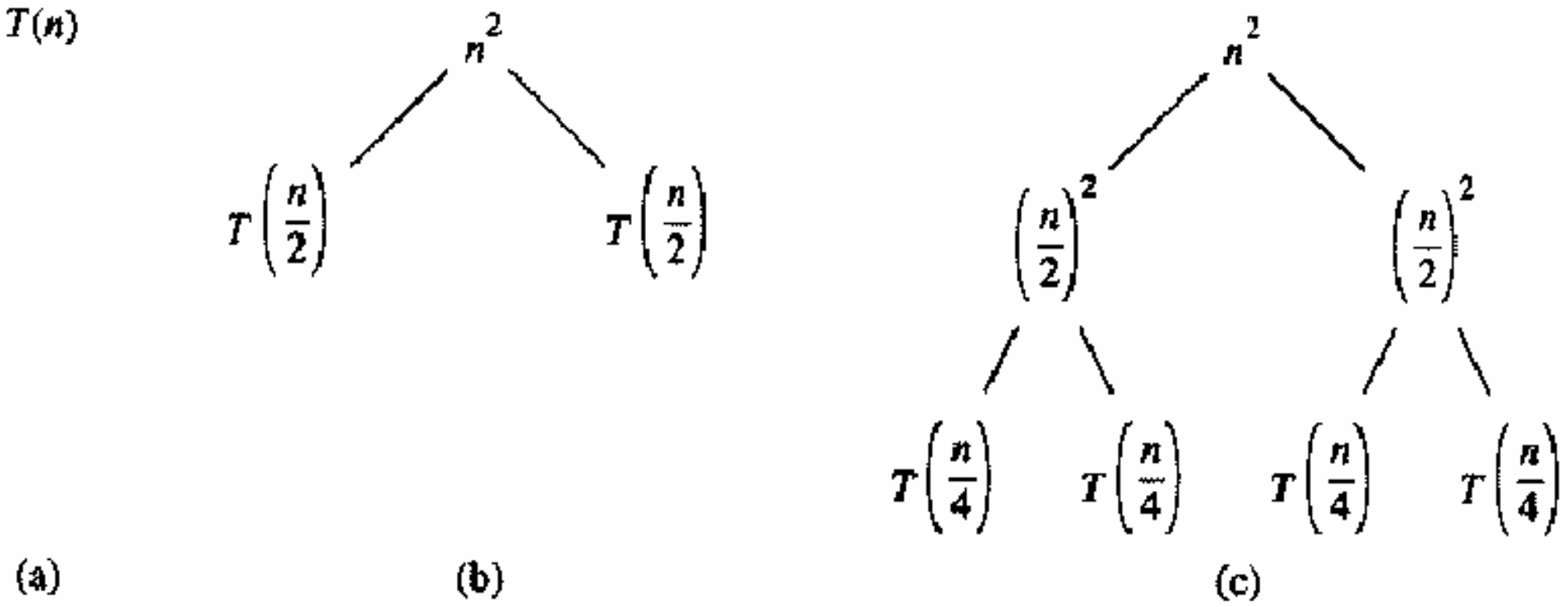
a) $\lfloor \lfloor n/a \rfloor / b \rfloor = \lfloor n/ab \rfloor$

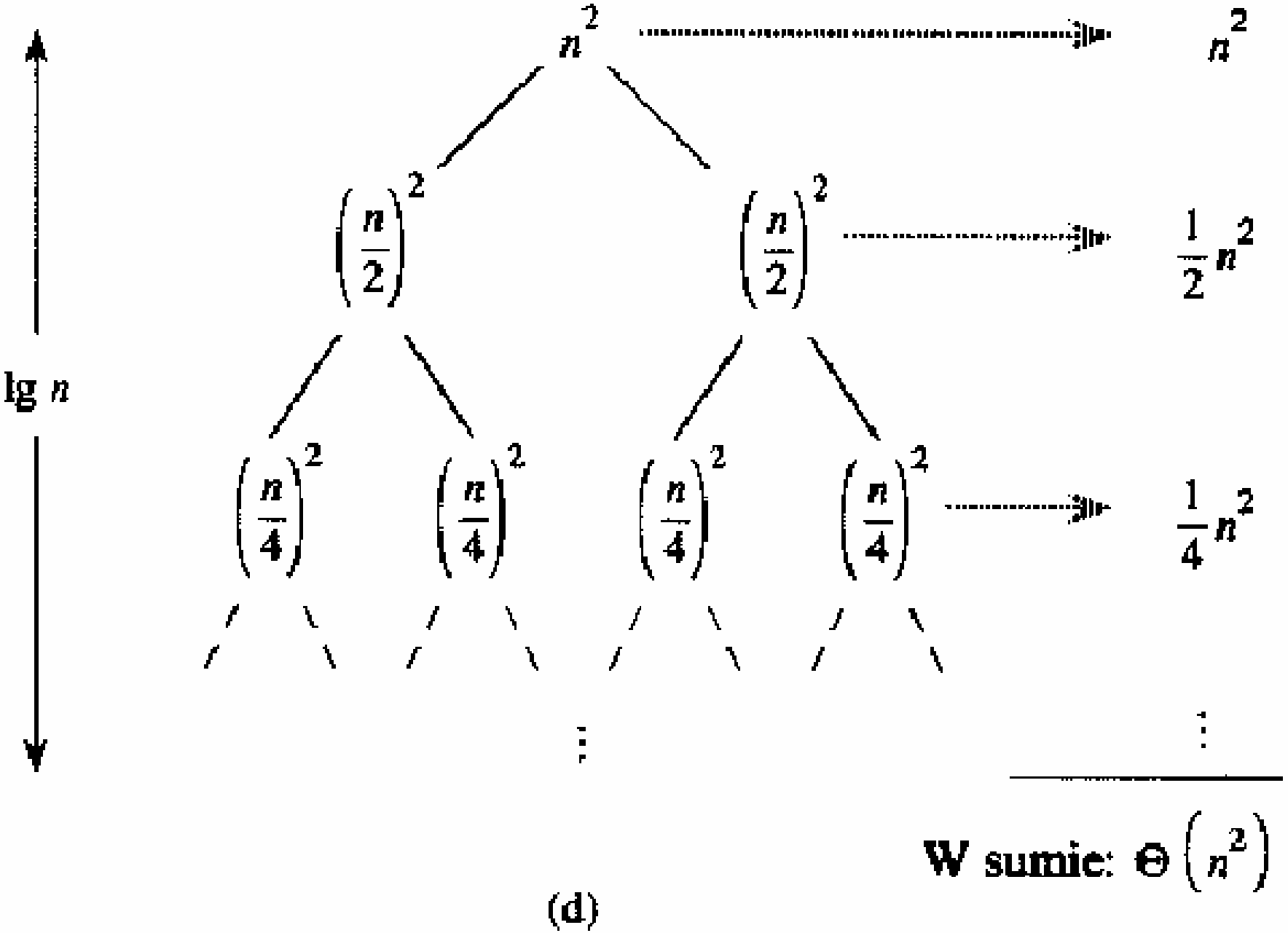
b) $3^{\log_4 n} = n^{\log_4 3}$

c) $\log_4 3 < 1$

Drzewa rekursji – ilustracja rozwijania rekurencji

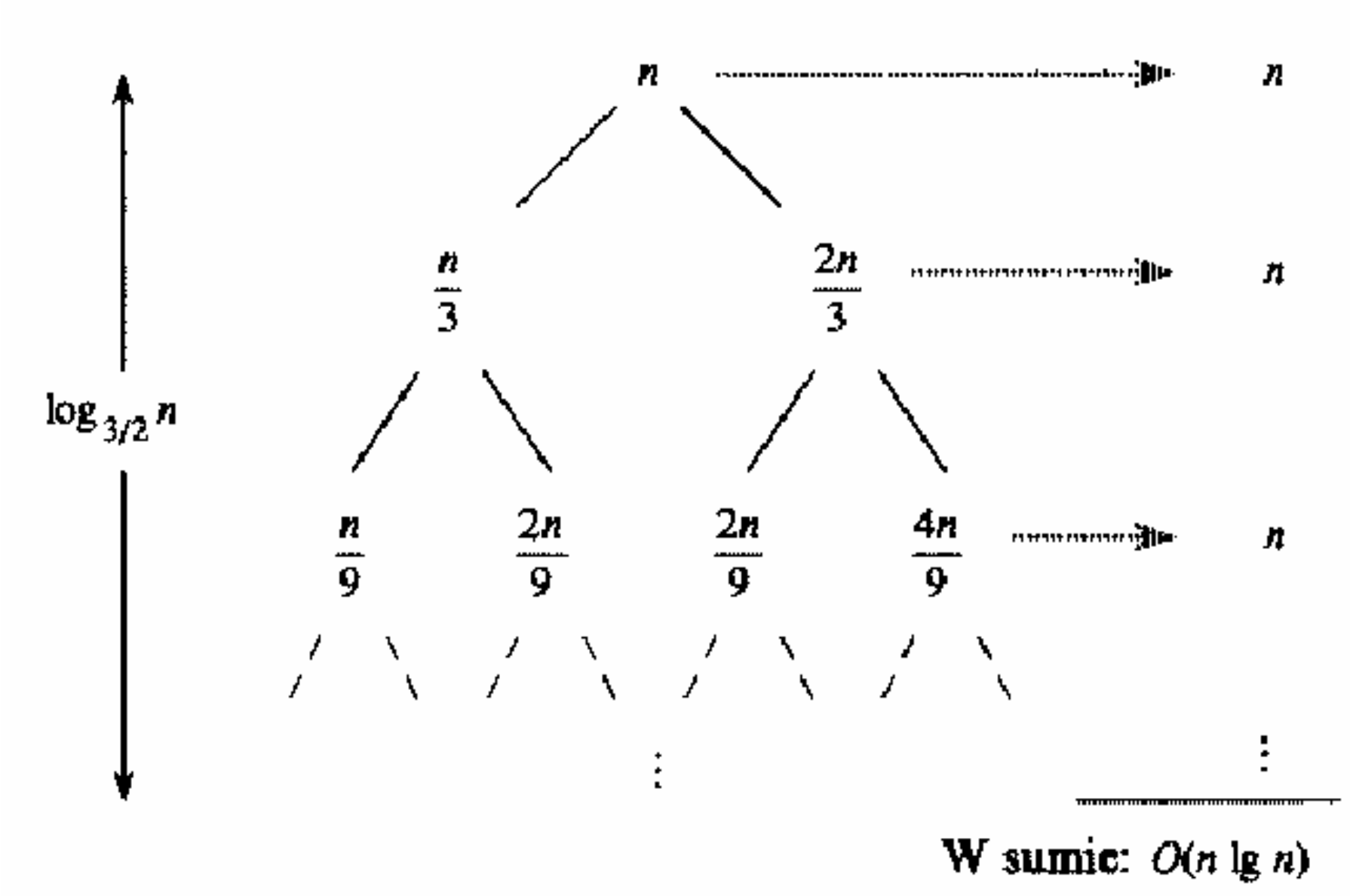
$T(n) = 2T(n/2) + n^2$




n^2
n^2
$\left(\frac{n}{2}\right)^2$
$\left(\frac{n}{2}\right)^2$
$\frac{1}{2}n^2$
lg n
$\left(\frac{n}{4}\right)^2$
$\left(\frac{n}{4}\right)^2$
$\left(\frac{n}{4}\right)^2$
$\left(\frac{n}{4}\right)^2$
$\frac{1}{4}n^2$
W sumie: $\Theta\left(n^2\right)$

(d)

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



Metoda rekurencji uniwersalnej

Metoda pozwala rozwiązać równanie rekurencyjne postaci

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

gdzie $a \geq 1$ i $b \geq 1$ są stałymi, a $f(n)$ jest funkcją asymptotycznie dodatnią.

(dzielimy problem rozmiaru n na a podproblemów, każdy rozmiaru n/b, gdzie a i b są dodatnimi stałymi. Koszt dzielenia problemu oraz łączenia rezultatów częściowych jest opisany funkcją $f(n)$).

Twierdzenie o rekurencji uniwersalnej

Niech $a \geq 1$ i $b \geq 1$ będą stałymi, $f(n)$ pewną funkcją i niech $T(n)$ będzie zdefiniowane dla nieujemnych liczb całkowitych przez rekurencję

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

gdzie n/b oznacza $\lfloor n/b \rfloor$ lub $\lceil n/b \rceil$. Wtedy funkcja $T(n)$ może być ograniczona asymptotycznie w następujący sposób:

1. Jeśli $f(n) = O(n^{\log_b a - \varepsilon})$ dla pewnej stałej $\varepsilon > 0$, to $T(n) = \Theta(n^{\log_b a})$.
2. Jeśli $f(n) = \Theta(n^{\log_b a})$, to $T(n) = \Theta(n^{\log_b a} \log n)$.
3. Jeśli $f(n) = \Omega(n^{\log_b a + \varepsilon})$ dla pewnej stałej $\varepsilon > 0$ i jeśli $a\, f(n/b) \leq c\, f(n)$ dla pewnej stałej $c > 1$ i wszystkich dostatecznie duzych n, to $T(n) = \Theta(f(n))$.

Uwagi:
Warunek 1. mówi, że funkcja $n^{\log_b a}$ musi być wielomianowo większa od $f(n)$.
Warunek 3. mówi, że funkcja $n^{\log_b a}$ musi być wielomianowo mniejsza od $f(n)$.
Warunek 3. mówi, że funkcje $n^{\log_b a}$ i $f(n)$ muszą być tego samego rzędu, przy dodatkowym warunku regularności.

Przykład: $T(n) = 9T(n/3) + n$
$a = 9$, $b = 3$, $f(n) = n$. Liczymy $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$.
Ponieważ $f(n) = O(n^{\log_3 9-\varepsilon})$ gdzie np. $\varepsilon = 1$, możemy zastosować przypadek 1 z twierdzenia.
Zatem $T(n) = \Theta(n^2)$.

Przykład: $T(n) = T(2n/3) + 1$
$a = 1$, $b = 3/2$, $f(n) = 1$. Liczymy $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$.
Ponieważ $f(n) = \Theta(n^{\log_b a}) = 1$, to stosujemy przypadek 2.
Zatem $T(n) = \Theta(\log n)$.

Przykład: $T(n) = 3T(n/4) + n \log n$
$a = 3$, $b = 4$, $f(n) = n \log n$. Liczymy $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$. Ponieważ $f(n) = \Omega(n^{\log_4 3+\varepsilon})$ gdzie $\varepsilon \approx 0.2$, to możemy zastosować przypadek 3, o ile zachodzi warunek regularności dla dostatecznie dużych n:
$a\, f(n/b) = 3\, n/4 \log n/4 \leq 3/4\, n \log n = c\, f(n)$ dla $c = 3/4$.
Zatem $T(n) = \Theta(n \log n)$.

Generowanie podciągów

Problem

Na wejściu znajdują się dwie liczby całkowite *m* i *n*, gdzie *m* <= *n*. Wynikiem jest posortowana lista *m* losowych liczb całkowitych z przedziału 1 ... *n*, wśród których żadna nie powtarza się dwukrotnie.

Algorytm nr 1 – zwykłe wybieranie

Generujemy *m* liczb pseudolosowych z przedziału 1... *n*. Po każdym losowaniu sprawdzamy, czy liczba się nie powtórzyła; w takim przypadku powtarzamy losowanie. Wylosowane liczby sortujemy wybraną metodą.
Zalety: prosty, intuicyjny
Wady: nieefektywny (szczególnie dla $m \approx n$), konieczność sortowania wyników

Algorytm nr 2 – test losowy

Algorytm analizuje kolejno liczby całkowite 1, 2, ..., *n* i na podstawie odpowiedniego testu losowego decyduje, czy wybrać, czy też odrzucić każdą z nich. Oglądając liczby po kolei mamy pewność, że w wyniku otrzymamy ciąg posortowany.
Rozważmy przykład, w którym $m = 2$ i $n = 5$. Powinniśmy wybrać liczbę 1 z prawdopodobieństwem 2/5; program powinien realizować to zadanie za pomocą instrukcji w rodzaju `if RandReal(0,1) < 2/5 then ...`
Niestety, nie możemy wybrać liczby 2 z tym samym prawdopodobieństwem: w wyniku takiego postępowania nie mielibyśmy pewności, że otrzymamy dokładnie 2 liczby spośród 5. Tak więc na następnej decyzji zaważy poprzedni wybór i decyzję na temat liczby 2 podejmiemy z prawdopodobieństwem 1/4, jeśli jedynka została wybrana i 2/4, jeśli wybrana nie została. Ogólnie, aby wylosować *w* liczb spośród *p* pozostałych, będziemy następną liczbę wybierać z prawdopodobieństwem *w/p*.

Zalety: oparty na solidnych podstawach matematycznych, wynik od razu posortowany
Wady: nieefektywny dla $m \ll n$

Algorytm nr 3 – przemieszanie

Umieszczamy liczby z przedziału 1 ... *n* w *n*-elementowej tablicy *T*. Następnie mieszamy (czyli zamieniamy) pierwszych *m* elementów tablicy *T* z elementami 1 ... *n* tej samej tablicy (oczywiście, w szczególnym przypadku taka zamiana może nie nastąpić, gdy chcąc przemieszać *i*-tą liczbę wylosujemy właśnie liczbę *i*). Wynikiem (po posortowaniu) jest tablica złożona z pierwszych *m* przemieszanych elementów z tablicy *T*.

Zalety: szybki (szczególnie dla $m \ll n$)
Wady: wymaga użycia sortowania oraz pomocniczej tablicy