Algorytmy i Struktury Danych - ćwiczenia

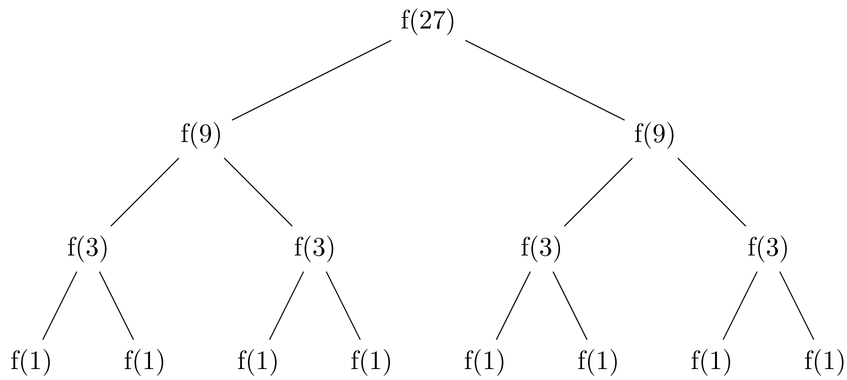
Wojciech Typer

zadanie 1/ lista1
Zadanie sprowadza się do znalezenia najmniejszego n, takiego, że:
$44n^2 < 2^n$
najmniejszym takim n jest $n = 13$
indukcyjnie można pokazać, że dla każdych następnych n
nierówność dalej będzie zachowana:
zał: $44n^2 < 2^n$
krok indukcyjny:
$44(k+1)^2 < 2^{k+1}$
$44(k^2 + 2k + 1) < 2^k * 2$
$44k^2 + 88k + 44 < 2^k * 2$
z założenia mamy, że: $44k^2 < 2^k$
więc musimy pokazać, że: $88k + 44 < 2^k (k \geq 13)$
Ten fragment jest już bardzo łatwo udowodnić indukcyjnie.

zadanie 2/ lista1
znając $f(n) = t$, musimy znaleźć n
przeliczmy jednostki czasu na mikrosekundy:
$1s = 10^6 \mu s$, $30min = 1.8 * 10^9 \mu s$ i $1wiek = 3.1 * 10^15 \mu s$
zatem:
$log_{10}(n) = 10^6 \to n = 10^{60}$,
$log_{10}(n) = 1.8 * 10^9 \to n = 10^1.8 * 10^9$,
$log_{10}(n) = 3.1 * 10^{15} \to n = 10^3.1 * 10^{15}$
$\sqrt{n} = 10^6 \to n = 10^{12}$, $\sqrt{n} = 1.8 * 10^9 \to n = 3.24 * 10^{18}$,
$\sqrt{n} = 3.1 * 10^{15} \to n = 9.61 * 10^{30}$
$2^n = 10^6 \to n = 19, 2^n = 1.8 * 10^9 \to n = 30.7, 2^n = 3.1 * 10^{15} \to n = 51$
$n! = 10^6 \to n = 9, n! = 1.8 * 10^9 \to n = 13, n! = 3.1 * 10^{15} \to n = 18$

zadanie 3/ lista1
1. $e^\pi \to O(1)$
2. $7(log_{10}(n))^7 \to O((log(n))^7)$
3. $\sqrt{2\pi n} \to O\sqrt{n}$
4. $13n + 13 \to O(n)$
5. $44n^2 * log(n) \to O(n^2 * log(n))$
6. $10^n \to O(10^n)$
7. $33^n \to O(33^n)$

zadanie 1/ lista2



Funkcja f(n) dwukrotnie wywołuje samą siebie dla n/3. Drzewo rekurencji dla $n = 3^3$ zostało przedstawione powyżej.
Funkcja dzieli n przez 3 w każdym kroku, aż do osiągnięcia n = 1.
Głębokość drzewa rekurencji wynosi zatem: $log_3(n)$
Na każdym poziomie drzewa liczba wywołań funkcji się podwaja,
i idąc od góry jest to: $2^0 + 2^1 + 2^2 + ... + 2^{log_3(n)}$
co daje: $2^{log_3(n)+1} - 1$
zatem złożoność obliczeniowa wynosi: $O(2^{log_3(n)})$

zadanie 2/ lista2
$h(n) = O(p(n))$, jeśli istnieją takie stałe $c_1, c_2, n_0 \leq 0$, że:
dla każdego n zachodzi:
$c_1 p(n) \leq h(n) \leq c_2 p(n)$
Musimy pokazać, że: $max(f(n), g(n)) = O(f(n) + g(n))$
Ograniczenie z góry:
$max(f(n), g(n)) \leq f(n) + g(n)$
Największa z dwóch liczb f(n) i g(n) nigdy nie przekroczy ich sumy,
czyli mamy ograniczenie górne: $max(f(n), g(n)) = 1 * (f(n) + g(n))$
czyli możemy przyjąć $c_2 = 1$
Ograniczenie z dołu:
$max(f(n), g(n)) \geq 1/2 * (f(n) + g(n))$
Ponieważ jeśli $f(n) \geq g(n)$ to $f(n) \geq \frac{1}{2}(f(n) + g(n))$,
Więc możemy przyjąć $c_1 = \frac{1}{2}$
Zatem istnieją stałe c_1, $c_2 \geq 0$, spełniające definicje O,
więc $max(f(n), g(n)) = O(f(n) + g(n))$
c.n.w

zadanie 3/ lista2
1. $P(i, j) = O(1)$
Złożoność pętli wewnętrznej while: $O(n - 1)$
Złożoność pętli zewnętrznej for to: $\sum_{i=1}^{n} O(n - 1)$

Jest to suma ciągu arytmetycznego $\frac{(n-1)n}{2} = O(n^2)$
Zatem złożoność algorytmu to: $O(n^2)$

2. $R(i,j) = O(j)$
Złożoność pętli wewnętrznej while:
j rośnie wykładniczo: j, 2j, 4j, 8j...
Zatem koszt każdej iteracji to O(j) + O(2j) + ... O(n)
Jest to ciąg geometryczny, którego suma jest proporcjonalna do
największego wyrazu: O(n)
Złożoność pętli zewnętrznej for to: $\sum_{i=1}^{n} O(n)$
Więc tak jak w poprzednim przypadku złożoność tego algorytmu to: $O(n^2)$