1. Złożoność obliczeniowa

Złożoność obliczeniowa to ilość zasobów komputerowych koniecznych do wykonania programu realizującego algorytm. Przedstawiamy ją jako funkcję pewnego parametru, określającego rozmiar rozwiązywanego zadania.

Ponieważ w przypadku szacowania złożoności obliczeniowej mówimy o czasie i pamięci, to wyróżniamy złożoność pamięciową i czasową.

Złożoność pamięciowa (zawsze jako funkcja rozmiaru danych!) to ilość pamięci wykorzystanej w celu realizacji algorytmu, wyrażana w liczbie bajtów lub liczbie zmiennych typów elementarnych.

Złożoność czasowa (zawsze jako funkcja rozmiaru danych!): jest to czas wykonania algorytmu wyrażany w standardowych jednostkach czasu, liczbie cykli procesora lub w liczbie wszystkich operacji. Dokładna wartość złożoności czasowej jest niemożliwa do uzyskania na etapie analizy algorytmu, dlatego zazwyczaj ograniczamy się do oszacowania przybliżonej wartości złożoności czasowej. Złożoności czasowej nie wyrażamy w standardowych jednostkach czasu, gdyż w zależności od tego, jaką maszyną dysponujemy, algorytm będzie wykonywał się na różnych maszynach w różnym czasie. Dlatego złożoność obliczeniową czasową wyrażamy w ilości operacji dominujących.

Dla algorytmów, gdzie czas wykonania zależy od egzemplarza danych, wyróżnia się złożoności: pesymistyczną, optymistyczną i oczekiwaną (średnią).
Dla przykładu weźmy algorytm sprawdzający, czy w zbiorze danych wejściowych jest liczba ujemna. W przypadku optymistycznym pierwsza ze sprawdzonych wartości okaże się ujemna, więc złożoność obliczeniowa optymistyczna $T_o(n) = 1$. W przypadku pesymistycznym, kiedy w zbiorze nie ma liczb ujemnych, bądź jest jedna i algorytm znajdzie ją jako ostatnią sprawdzaną wartość, złożoność pesymistyczna $T_p(n)=n$. Złożoność oczekiwana średnia $T_s(n)$ wyniesie natomiast n/2 ponieważ liczba ujemna będzie znajdowana wcześniej bądź później, ale średnio po sprawdzeniu połowy zbioru.

Zatem podsumowując:

- Złożoność optymistyczna określa zużycie zasobów dla najkorzystniejszego zestawu danych.
- Złożoność średnia określa zużycie zasobów dla typowych (tzw. losowych) danych.
- Złożoność pesymistyczna określa zużycie zasobów dla najbardziej niekorzystnego zestawu danych.

1.1. Przykłady złożoności obliczeniowej

Złożoności wielomianowe:
(n) - liniowa np. T(n) = 230n
(n2) - kwadratowa np. T(n) = $12n^2 + 135n - 23$
(n3) - sześcienna np. T(n) = $n^3 + 20n^2 - 19n + 1$

Złożoności ograniczone przez wielomian:
(log n) - logarytmiczna np. T(n) = 3log (n+1) - 2
(nlog n) - quasiliniowa np. T(n) = 3nlog (n+1) – 2

Złożoności niewielomianowe:
NP-zupełna
(a^n) - wykładnicza np. u(n) = $e^n + n^{13} - n^2$
silnie NP-zupełna
NP-trudna

1.2. Klasy złożoności obliczeniowej

Przy analizie algorytmów korzysta się z tzw. klas złożoności obliczeniowej, które określają rząd funkcji T(n). Jednym ze sposobów określania rzędu tej funkcji jest popularna notacja omikron (zwana także notacją dużego O) o następującej definicji:

Notacja O („duże o”)
Mówimy, że T(n) = O(f(n)) (funkcja złożoności obliczeniowej T(n) jest rzędu funkcji f(n)) jeśli potrafimy znaleźć takie $n_0 \in N$ oraz takie $c \in R$, iż dla każdego $n \geq n_0$ prawdziwa jest nierówność:
$T(n) \leq c \cdot f(n)$
Przykład: $2n^2 = O(n^3)$ ($c=1$, $n_0=2$)

Notacja Θ
Mówimy, że T(n) = Θ (f(n)) jeśli istnieją stałe dodatnie c_1, c_2 i n_0 takie $n_0 \in N$, iż dla każdego $n \geq n_0$ prawdziwa jest nierówność:
$c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$

Notacja Ω
Mówimy, że T(n) = Θ (f(n)) jeśli istnieją stałe dodatnie c i $n_0 \in N$ takie, że dla każdego $n \geq n_0$ prawdziwa jest nierówność:
$c \cdot T(n) \leq f(n)$

2. Szacowanie złożoności obliczeniowej

Szacowanie złożoności obliczeniowej algorytmu najlepiej zilustruje przykład. Przeanalizujemy algorytm obliczający sumę *n* liczb.

Krok	Operacja	Czas wykonania
Krok 1:	Wczytaj n	$1 \times t_1$
Krok 2:	$suma := 0$	$1 \times t_2$
Krok 3:	$i := 1$	$1 \times t_2$
Krok 4:	Jeżeli $i > n$ to idź do K7	$(n+1) \times t_3$
Krok 5	$i := i + 1$	$n \times t_2$

Krok 6:	$suma \coloneqq suma + i$	$n \times t_2$
Krok 7:	Pisz $suma$	$1 \times t_4$
Krok 8:	Koniec	$1 \times t_5$

$T(n) = t_1 + t_2 + t_2 + (n+1)t_3 + nt_2 + nt_2 + t_4 + t_5$
$T(n) = t_1 + 2t_2 + nt_3 + t_3 + 2nt_2 + t_4 + t_5$
$T(n) = n(t_3 + 2t_2) + t_1 + 2t_2 + t_3 + t_4 + t_5$

Podstawiając $t_6 = t_3 + 2t_2$ oraz $t_7 = t_1 + 2t_2 + t_3 + t_4 + t_5$

$T(n) = nt_6 + t_7$

A zatem otrzymaliśmy złożoność obliczeniową liniową.

Innym sposobem określenie złożoności czasowej jest wyznaczenie w algorytmie operacji dominującej i zliczenie liczby jej wykonań. Pozostałe operacje traktujemy jako nieistotne - tzn. ich czas wykonania jest pomijalnie mały w porównaniu z czasem wykonania wszystkich operacji dominujących. W naszym algorytmie taką operacją dominującą może na przykład jeden obieg pętli sumującej liczby naturalne.

Zadania: Oszacuj złożoność obliczeniową następujących algorytmów:

Zad. 1
Przeszukiwanie elementów tablicy a w celu znalezienia danego elementu x

Krok	**Operacja**
Krok 1:	Wczytaj n
Krok 2:	$i \coloneqq 1$
Krok 3:	Jeżeli ($a[i] = x$ lub $i = n$) to idź do K5
Krok 4:	$i \coloneqq i + 1$
Krok 5	Koniec

Zad. 2
Przeszukiwanie elementów posortowanej tablicy a w celu znalezienia danego elementu x (metoda bisekcji lub przeszukiwania połówkowego)

Start
$i: = 1$
$j \coloneqq n$
dopóki ($x! = a[i]\ oraz\ x! = a[j]$)
    $k \coloneqq (i + j) div 2$
    Jeżeli $x > a[k]$
        $i \coloneqq k + 1$
    w p. p. $j \coloneqq k - 1$
Koniec

Zad. 3
Sortowanie bąbelkowe $n-$ elementowej tablicy a.

Start
dopóki $(n > 1)$
$for(i \coloneqq 0\ ;\ i < n-1; i++)$
Jeżeli $a[i] > a[i+1]$
zamień $a[i] oraz\ a[i+1]$
$n = n-1$
Koniec

Zad. 4
Sortowanie przez wybieranie.

Start
$for(i \coloneqq 0\ ;\ i < n-1; i++)$
$for(j \coloneqq i+1\ ;\ j < n; j++)$
Jeżeli $a[j] > a[i]$
zamień $a[j] oraz\ a[i]$
Koniec