ZŁOŻONOŚĆ ALGORYTMÓW

- **Złożoność pamięciowa algorytmu**

  wynika z liczby i rozmiaru struktur danych wykorzystywanych w algorytmie;
- **Złożoność czasowa algorytmu**

  wynika z liczby operacji elementarnych wykonywanych w trakcie przebiegu algorytmu.

ZŁOŻONOŚĆ CZASOWA ALGORYTMU
<u>**zależność**</u> pomiędzy rozmiarem danych wejściowych a liczbą wybranych operacji elementarnych wykonywanych w trakcie przebiegu algorytmu
(zależność podawana jako <u>funkcja</u>, której argumentem jest rozmiar danych, a wartością liczba operacji).

Przykłady funkcji złożoności czasowej

1. Algorytm sortowania bąbelkowego:
   dla listy o długości *N* jego złożoność może wyrazić funkcja
   F(*N*) = liczbie porównań par sąsiednich elementów (?)
2. Algorytm rozwiązywania problemu wież Hanoi:
   dla *N* krążków jego złożoność może wyrazić funkcja
   F(*N*) = liczba przeniesień pojedynczego krążka z kołka na kołek (?)
3. Algorytm wyznaczania najdłuższej przekątnej wielokąta wypukłego: dla wielokąta o *N* wierzchołkach jego złożoność może wyrazić funkcja
   F(*N*) = liczba porównań długości dwóch przekątnych (?)
4. Algorytm zachłanny wyznaczania „najkrótszej sieci kolejowej”:
   dla *N* „miast” i *M* możliwych do zbudowania odcinków jego złożoność może wyrazić funkcja
   F(*N*, *M*) = liczba porównań długości dwóch odcinków (?)

W praktyce złożoność czasowa algorytmu decyduje często o jego przydatności

Dzieje się tak ponieważ:

- trzeba rozwiązywać algorytmicznie coraz większe zadania:
  - w komputerowych systemach wspomagania decyzji,
  - przy komputerowych symulacjach i prognozach złożonych zjawisk.
- rozwijane są komputerowe systemy czasu rzeczywistego:
  - sterujące automatycznie złożonymi układami (transport, produkcja),
  - wyszukujące i przetwarzające duże ilości informacji.

Chcemy zmniejszyć czas wykonania algorytmu poprzez zmniejszenie liczby wykonanych operacji elementarnych

1. Przykład zmniejszenia liczby operacji w algorytmie

Algorytm normalizacji wartości przechowywanych w tablicy jednowymiarowej względem wartości maksymalnej

Dane wejściowe zapisane w tablicy *T*(*K*) dla *K* = 1, 2, ..., *N*

Alg. 1
1. wyznacz w zmiennej *MAX* największą z wartości;
2. **dla** *K* **od** 1 **do** *N* **wykonuj co następuje:**
   2.1. *T*(*K*) ← *T*(*K*) · 100 / *MAX*

$\boldsymbol{F_1(N) = 2 \cdot N}$

Alg. 2
1. wyznacz w zmiennej *MAX* największą z wartości
2. *ILORAZ* ← 100 / *MAX* ;
3. **dla** *K* **od** 1 **do** *N* **wykonuj co następuje:**
   3.1. *T*(*K*) ← *T*(*K*) · *ILORAZ*

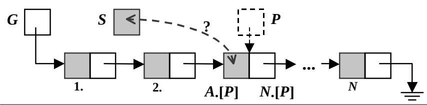
$\boldsymbol{F_2(N) = N + 1}$

Wybieramy operację elementarną, którą jest wyznaczenie wartości iloczynu lub ilorazu dwóch zmiennych

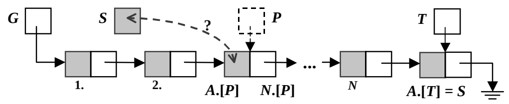
2. Przykład zmniejszenia liczby operacji w algorytmie

Algorytm (liniowego) wyszukiwania elementu z podanej listy

Dane wejściowe to *N* elementów zapisanych w polach kluczowych listy wskaźnikowej (pola rekordu: *A* – kluczowe, *N* – wskaźnikowe; *G* – wskaźnik pierwszego rekordu) i szukany element podany w *S*.



Alg. 1
1. *P* ← *G* ; *F* ← NIL ;
2. **dopóki** *P* ≠ NIL **i** *F* = NIL **wykonuj co następuje:**
   2.1. **jeżeli** *S* = *A*.[*P*] , **to** *F* ← *P* ;
   2.2. *P* ← *N*.[*P*]
3. **jeżeli** *F* ≠ NIL, **to** odczytaj wartość wskaźnika *F*.



Alg. 2
1. wstaw rekord na koniec listy; *A*.[*T*] ← *S* ;
2. *P* ← *G* ; *F* ← NIL ;
3. **dopóki** *F* = NIL **wykonuj co następuje:**
   3.1. **jeżeli** *S* = *A*.[*P*] , **to** *F* ← *P* ;
   3.2. *P* ← *N*.[*P*]
4. **jeżeli** *F* ≠ *T*, **to** odczytaj wartość wskaźnika *F*.

$\boldsymbol{F_2(N) = N + 2}$

Wybieramy operację elementarną, którą jest porównanie zawartości wskaźników *P* lub *F* z innym wskaźnikiem lub adresem NIL

Badamy najgorszy przypadek

Alg. 1 $\boldsymbol{F_1(N) = 2 \cdot N + 1}$

<u>Poprawianie czasowej złożoności</u> algorytmu jest czymś więcej, niż tylko zmniejszaniem liczby operacji wykonywanych w trakcie jego działania!

Jak możemy stwierdzić, że złożoność została zmniejszona?

Wykonując <u>asymptotyczną analizę złożoności</u> stwierdzamy jak szybko w miarę wzrostu rozmiaru danych wejściowych rośnie funkcja złożoności podająca liczbę operacji wykonywanych w algorytmie (badamy tzw. rząd złożoności algorytmu).

Jeżeli chcemy stwierdzić, który z dwóch algorytmów będących rozwiązaniami tego samego problemu algorytmicznego ma istotnie niższą złożoność, to musimy porównać ich funkcje złożoności i określić czy różnią się ich rzędy złożoności.

Jeżeli porównujemy dwa algorytmy wykonujące to samo zadanie za pomocą jednej pętli ograniczonej, w której liczba powtórzeń (iteracji) jest wprost proporcjonalna do rozmiaru danych wejściowych *N*, to liczby operacji elementarnych (czasy wykonania) tych algorytmów w funkcji rozmiaru danych opisane są odpowiednio przez:

$$\boldsymbol{F}_1(N) = \mathrm{K}_1 + \mathrm{L}_1 \cdot N$$

$$\boldsymbol{F}_2(N) = \mathrm{K}_2 + \mathrm{L}_2 \cdot N$$

Do porównania złożoności tych algorytmów możemy wykorzystać iloraz:

$$\boxed{s(N) = \frac{F_1(N)}{F_2(N)}}$$

Wtedy spełnienie warunków: $\boxed{s(N) = \frac{F_1(N)}{F_2(N)}}$

$s(N) = 1$ oznaczałoby jednakową złożoność algorytmów,

$s(N) < 1$ oznaczałoby, że 1. algorytm ma niższą (lepszą) złożoność.

Ale zauważmy, że $s(N) = 1$ zachodzi tylko wtedy,
kiedy $\mathrm{K}_1 = \mathrm{K}_2$ i $\mathrm{L}_1 = \mathrm{L}_2$,
co oznacza, że oba algorytmy są praktycznie identyczne.

A co mamy powiedzieć,
kiedy $s(N) \geq 1$ dla $N \leq N_0$, ale $s(N) < 1$ dla $N > N_0$.

Który algorytm jest lepszy?

W <u>asymptotycznej analizie złożoności</u> przyjęto, że porównując algorytmy badamy wartość ilorazu $s(N)$ dla bardzo dużych wartości *N*, czyli formalnie jego granicę dla $N \to \infty$.

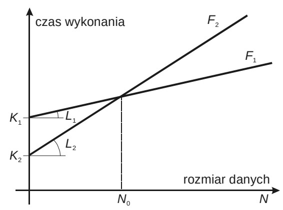
Zatem o wyniku porównania złożoności dwóch algorytmów decyduje wartość:

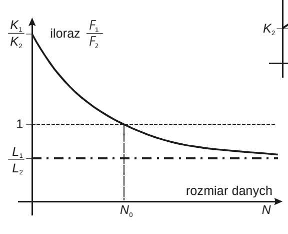
$$\boxed{\lim_{N\to\infty} s(N) = \lim_{N\to\infty} \frac{F_1(N)}{F_2(N)}}$$

W przypadku porównywania dwóch funkcji liniowych będzie to wartość:

$$\boxed{\lim_{N\to\infty} s(N) = \frac{L_1}{L_2}}$$

Przykład asymptotycznego porównania dwóch liniowych funkcji złożoności





Algorytm o funkcji złożoności F_1 ma niższą złożoność.

Ale czy jest to złożoność niższa o rząd?

Asymptotyczna analiza złożoności algorytmów

Dwa algorytmy opisane funkcjami $F_1(N)$ i $F_2(N)$ mają **złożoność tego samego rzędu**, jeśli

$$\lim_{N\to\infty} \frac{F_1(N)}{F_2(N)} = C \quad \text{i zachodzi} \quad \boxed{\mathbf{0 < C < \infty}}$$

Sytuację, w której dwie funkcje złożoności są tego samego rzędu zapisujemy

$\boxed{F_1(N) = \Theta(F_2(N))}$ (notacja theta)

Jeśli zachodzi $F_1(N) = \Theta(F_2(N))$, to także $F_2(N) = \Theta(F_1(N))$

Algorytm opisany funkcją $F_1(N)$ ma **złożoność nie wyższego rzędu** niż algorytm opisany funkcją $F_2(N)$, jeśli

$$\lim_{N\to\infty} \frac{F_1(N)}{F_2(N)} = C \quad \text{i zachodzi} \quad \boxed{C < \infty}$$

Sytuację, w której pierwsza funkcja złożoności jest nie wyższego rzędu niż druga zapisujemy

$\boxed{F_1(N) = O(F_2(N))}$ (notacja O duże)

Jeśli zachodzi jednocześnie $F_1(N) = O(F_2(N))$ i $F_2(N) = O(F_1(N))$, to $F_1(N) = \Theta(F_2(N))$.

Jeśli zachodzi $F_1(N) = \Theta(F_2(N))$, to także $F_1(N) = O(F_2(N))$.

Algorytm opisany funkcją $F_1(N)$ ma **złożoność niższego rzędu** niż algorytm opisany funkcją $F_2(N)$, jeśli

$$\lim_{N\to\infty} \frac{F_1(N)}{F_2(N)} = 0 \quad \text{, czyli} \quad \boxed{C = 0}$$

Sytuację, w której pierwsza funkcja złożoności jest niższego rzędu niż druga zapisujemy

$\boxed{F_1(N) \prec F_2(N)}$

Badanie złożoności algorytmów w sposób asymptotyczny prowadzi do wyróżnienia typowych rzędów złożoności:
- liniowa,
- kwadratowa,
- logarytmiczna, ...

Algorytm opisany funkcją $F(N)$ ma **złożoność liniową,** jeśli

$$\lim_{N\to\infty} \frac{F(N)}{N} = C \quad \text{i zachodzi} \quad \boxed{0 < C < \infty}$$

Złożoność liniowa (rzędu N) oznaczana jest symbolem $\mathbf{\Theta(N)}$

Algorytm ma złożoność **co najwyżej liniową**, jeśli $C < \infty$; oznaczenie $\mathbf{O(N)}$

Algorytm opisany funkcją $F(N)$ ma **złożoność kwadratową,** jeśli

$$\lim_{N\to\infty} \frac{F(N)}{N^2} = C \quad \text{i zachodzi} \quad \boxed{0 < C < \infty}$$

Złożoność kwadratową (rzędu N^2) oznaczana jest symbolem $\mathbf{\Theta(N^2)}$

Algorytm ma złożoność **co najwyżej kwadratową**, jeśli $C < \infty$; oznaczenie $\mathbf{O(N^2)}$

$\boxed{N \prec N^2}$

Przypadek, w którym zachodzi warunek
$\forall\ N : 0 \le F(N) \le C < \infty$ oznaczamy $F(N) = O(1)$

Dopiero obniżenie rzędu złożoności algorytmu jest istotnym ulepszeniem rozwiązania problemu algorytmicznego!

Jeżeli algorytm może wykonywać różną liczbę operacji elementarnych dla danych wejściowych o tym samym rozmiarze, w zależności od konkretnego ich zestawu, to możemy badać jego złożoność **w najgorszym przypadku**
(czyli skupiając się na takich przypadkach dopuszczalnych danych wejściowych, dla których liczba operacji jest największa)

- ***analiza złożoności najgorszego przypadku*** lub ***pesymistyczna***
- analiza złożoności średniego przypadku (trudniejsza)

Porównanie złożoności przykładowych algorytmów normalizacji tablicy

Złożoność czasowa algorytmu 1. wynosi $F_1(N) = 2 \cdot N$, czyli $F_1(N) = \Theta(N)$

Złożoność czasowa algorytmu 2. wynosi $F_2(N) = N + 1$, czyli $F_2(N) = \Theta(N)$

$\boxed{F_1(N) = \Theta(F_2(N))}$

Porównanie złożoności przykładowych algorytmów wyszukiwania elementu z listy (złożoność pesymistyczna)

Złożoność czasowa algorytmu 1. wynosi $F_1(N) = 2 \cdot N + 1$, czyli $F_1(N) = \Theta(N)$

Złożoność czasowa algorytmu 2. wynosi $F_2(N) = N + 2$, czyli $F_2(N) = \Theta(N)$

$\boxed{F_1(N) = \Theta(F_2(N))}$

Czy można zaproponować algorytm dla wyszukiwania elementu z listy o pesymistycznej złożoności niższej niż liniowa?

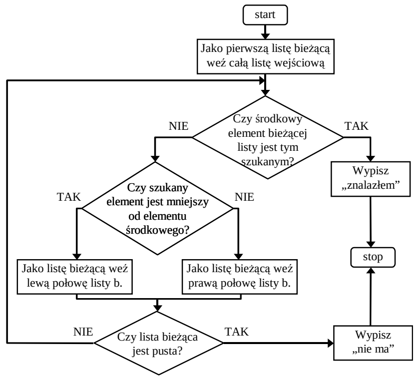
Szukamy zatem algorytmu, dla którego pesymistyczna funkcja złożoności spełniałaby warunek $F(N) \prec N$.

Jeżeli założymy, że lista $Y_1, Y_2, ..., Y_N$ jest <u>uporządkowana</u>, tzn. dla każdego $i < j$ zachodzi $Y_i \le Y_j$, to takim algorytmem jest wyszukiwanie **binarne** (przez połowienie)

Jeśli dane wejściowe są zapisane w polach kluczowych rekordów z listy wskaźnikowej o podanej w przykładzie strukturze, to warunek uporządkowania ma postać:

dla każdego bieżącego wskaźnika P zachodzi $A.[P] \le A.[N.[P]]$

Algorytm wyszukiwania binarnego (z listy uporządkowanej)



Jeśli, jako zliczaną operację elementarną, wybierzemy porównanie elementu szukanego ze środkowym elementem listy, to analiza złożoności będzie polegała po pierwsze na znalezieniu odpowiedzi na pytanie:
ile razy w najgorszym przypadku jest powtarzana pętla w algorytmie?

- Najgorszy przypadek jest wtedy, kiedy na liście nie ma szukanego elementu;
- Po każdej iteracji długość bieżącej listy maleje o połowę i iteracje są przerywane, gdy jej długość osiągnie wartość 0.

Zmiany w długości listy bieżącej:

start N
po 1 iteracji $N/2$
po 2 iteracji $N/4$
po 3 iteracji $N/8$
...
po K iteracji $N/2^K$
...
po przedostatniej iteracji $1 = N/2^M$

$\lg N \prec N$

bo $\lim\limits_{N\to\infty} \frac{\lg N}{N} = 0$

Zatem $M = \log_2 N$

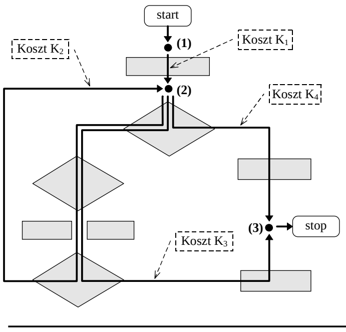
Pesymistyczna liczba iteracji $F(N) = 1 + \lg N = \Theta(\lg N) = \mathbf{O}(\lg N)$

Algorytm wyszukiwania binarnego ma złożoność logarytmiczną

Skala poprawy złożoności (o rząd):

N	$1+\lfloor \lg N \rfloor$
10	4
100	7
1 000	10
10 000	14
1 000 000	20
1 000 000 000	30
1 000 000 000 000	40
1 000 000 000 000 000	50
1 000 000 000 000 000 000	60

Przy asymptotycznej analizie złożoności o złożoności algorytmu decyduje, rosnąca w zależności od rozmiaru danych wejściowych, liczba wykonywanych iteracji (powtórzeń pętli)



Całkowity koszt czasowy w najgorszym przypadku:

$\boldsymbol{F(N)} =$

$= K_1 + \max(K_3, K_4) + K_2 \cdot \lg N$

$= \mathbf{O}(\lg N)$