Właściwości notacji O(·) (tzw. rachunek O(·))

- jeśli $F(N)$ jest funkcją złożoności algorytmu, to spełnienie warunku

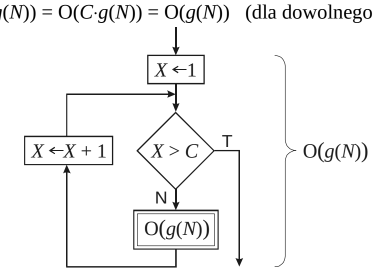
$$\lim_{N\to\infty} \frac{F(N)}{g(N)} = C < \infty$$

jest zapisywane $F(N) = O(g(N))$

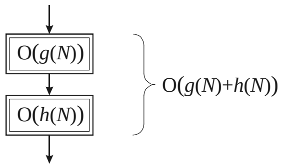
i odczytywane „ złożoność algorytmu jest rzędu nie wyższego niż $g(N)$" lub krócej „złożoność algorytmu jest $O(g(N))$"

- równość w zapisie $F(N) = O(g(N))$ powinna być rozumiana w ten sposób, że funkcja $F(N)$ jest jedną z funkcji, które spełniają powyższy warunek lub precyzyjniej, że funkcja $F(N)$ należy do zbioru wszystkich funkcji spełniających powyższy warunek

- $F(N) = O(F(N))$
- $O(\Theta(g(N))) = O(g(N))$
- $C\cdot O(g(N)) = O(C\cdot g(N)) = O(g(N))$ (dla dowolnego $0 < C < \infty$)



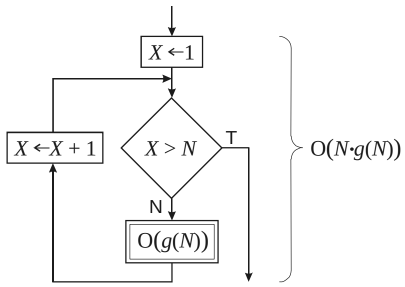
- $O(g(N)) + O(h(N)) = O(g(N) + h(N))$

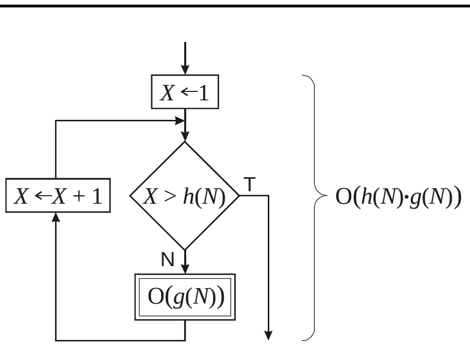


- Jeśli spełniony jest warunek $\lim_{N\to\infty} \frac{g(N)}{h(N)} = 0$, czyli $g(N) \prec h(N)$,

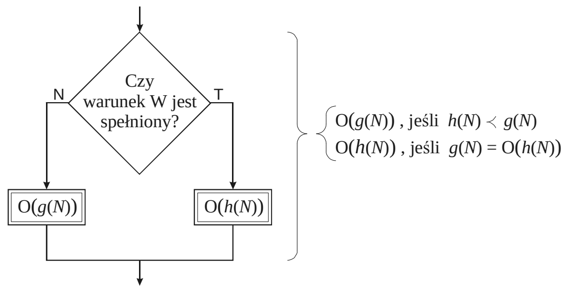
to $O(g(N)) + O(h(N)) = O(g(N) + h(N)) = O(h(N))$

- $O(g(N))\cdot O(h(N)) = O(g(N)\cdot h(N)) = g(N)\cdot O(h(N)) = h(N)\cdot O(g(N))$





- przy analizie złożoności w najgorszym przypadku, jeżeli warunek w wyborze warunkowym zależy od danych wejściowych, to zachodzi:



Złożoność czasowa przykładowych algorytmów

Algorytm sortowania bąbelkowego w pierwotnej wersji
(iteracja zagnieżdżona w iteracji)

1. wykonaj co następuje $N - 1$ razy:
   1.1. ... ,
   1.2. wykonaj co następuje $N - 1$ razy:
      1.2.1. porównaj wskazany element listy z następnym
      1.2.2. ... ,
      1.2.3. ... .

Liczba porównań par elementów (powtórzeń kroku 1.2.1.)
$F(N) = (N - 1)\cdot(N - 1) = N^2 - 2N + 1$

$1 \prec 2N \prec N^2$, czyli złożoność $F(N) = \mathbf{O}(N^2)$

Algorytm sortowania bąbelkowego w poprawionej wersji
(zmniejszająca się liczba powtórzeń w iteracji wewnętrznej)

1. wykonaj co następuje $N - 1$ razy:
   1.1. ... ,
   1.2. wykonaj co następuje $N - K$ razy:
      1.2.1. porównaj wskazany element listy z następnym
      1.2.2. ... ,
      1.2.3. ... .

Liczba porównań par elementów (powtórzeń kroku 1.2.1.)
$F(N) = (N - 1) + (N - 2) + (N - 3) + ... + 2 + 1 = 0{,}5\cdot N^2 - 0{,}5\cdot N$

$N \prec N^2$, czyli złożoność $F(N) = \mathbf{O}(N^2)$

Algorytm sumowania *N* liczb (pojedyncza iteracja)
Liczba dodawań dwóch wartości $F(N) = \mathbf{O}(N)$

Algorytm „brute force" znajdowania największej przekątnej w wielokącie wypukłym
(przegląd tablicy dwuwymiarowej)
Liczba porównań par wartości w celu znalezienia odcinka o maksymalnej długości
$F(N) = (N - 1) + (N - 2) + (N - 3) + ... + 2 + 1 = 0{,}5\cdot N^2 - 0{,}5\cdot N$
czyli złożoność $F(N) = \mathbf{O}(N^2)$

Najlepszy algorytm znajdowania największej przekątnej w wielokącie wypukłym (obracanie pary prostych równoległych wokół wielokąta – pojedyncza iteracja)
Liczba możliwych obrotów $F(N) = \mathbf{O}(N)$

Rekurencyjny algorytm dla problemu wież Hanoi

Procedura **przenieś** (N) **krążków** ...
1. jeśli $N = 1$, ...
2. w przeciwnym razie (tj. jeśli $N > 1$) wykonaj co następuje:
   2.1. Wywołaj procedurę **przenieś** ($N - 1$) **krążków** ...
   2.2. Wypisz ruch „X → Y",
   2.1. Wywołaj procedurę **przenieś** ($N - 1$) **krążków** ...
3. wróć ...

Oznaczmy nieznaną funkcję złożoności przez $T(N)$ i ułóżmy równanie, które $T(N)$ musi spełniać (tzw. **równanie rekurencyjne**):

$T(1) = 1$

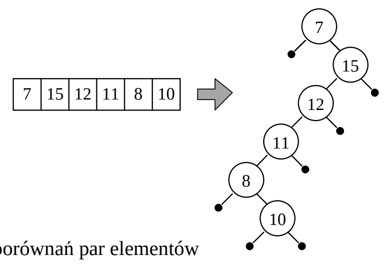
$T(N) = 2\cdot T(N - 1) + 1$

$T(N)$ – liczba przeniesień pojedynczego krążka w zadaniu z N krążkami

Równanie spełnia funkcja $F(N) = 2^N - 1 = \mathbf{O}(2^N)$

Algorytm sortowania drzewiastego
(dwa etapy: budowa drzewa BST i przegląd drzewa)

Etap budowy drzewa BST ma pesymistyczną złożoność $\mathbf{O}(N^2)$:

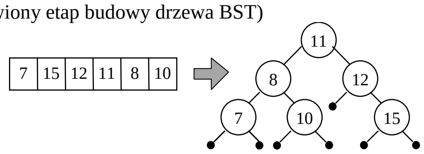


Liczba porównań par elementów
$F(N) = 1 + 2 + ... + (N - 2) + (N - 1) = 0{,}5\cdot N^2 - 0{,}5\cdot N = \mathbf{O}(N^2)$

Etap lewostronnego obejścia drzewa BST ma złożoność $\mathbf{O}(N)$

Zatem cały algorytm ma pesymistyczną złożoność
$O(N^2) + O(N) = \mathbf{O}(N^2)$

Algorytm sortowania drzewiastego z samoorganizacją drzewa
(poprawiony etap budowy drzewa BST)



Etap budowy drzewa BST ma pesymistyczną złożoność $\mathbf{O}(N\cdot\lg N)$

Zatem cały algorytm ma złożoność $O(N\cdot\lg N) + O(N) = \mathbf{O}(N\cdot\lg N)$

Porównanie rzędów złożoności:

Długość listy	Sortowanie bąbelkowe	Sort. drzewiaste
N	$\boldsymbol{N^2}$	$\boldsymbol{N \cdot \lg N}$
10	100	33
100	10 000	664
1 000	1 000 000	9 965
1 000 000	1 000 000 000 000	19 931 568
1 000 000 000	1 000 000 000 000 000 000	29 897 352 853

Algorytm sortowania przez scalanie (rekurencja)

Procedura ***sortuj*** (L)
1. jeśli lista L zawiera tylko jeden element, to jest posortowana,
2. w przeciwnym przypadku wykonaj co następuje:
   - 2.1. podziel listę L na dwie połowy L_1 i L_2 ,
   - 2.2. wywołaj ***sortuj*** (L_1) ,
   - 2.3. wywołaj ***sortuj*** (L_2) ,
   - 2.4. scal posortowane listy L_1 i L_2 w jedną posortowaną listę ,
3. wróć do poziomu wywołania.

Ułóżmy równanie rekurencyjne dla nieznanej funkcji złożoności:

$T(1) = 0$

$T(N) = 2 \cdot T(N / 2) + N$

$T(N)$ – liczba porównań par elementów

← w najgorszym przypadku

Równanie spełnia funkcja $F(N) = N \cdot \lg N = \mathbf{O}(N \cdot \lg N)$

Średnia złożoność v. pesymistyczna złożoność

Analiza średniego przypadku v. analiza najgorszego przypadku

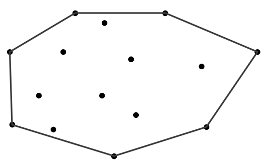
W analizie średniego przypadku istotną rolę odgrywają założenia o rozkładzie prawdopodobieństwa w zbiorze dopuszczalnych danych wejściowych.

Algorytm	*Śr. złożoność*	*Pes. złożoność*
sumowanie n liczb	$\mathbf{O}(N)$	$\mathbf{O}(N)$
sortowanie bąbelkowe	$\mathbf{O}(N^2)$	$\mathbf{O}(N^2)$
sortowanie drzewiaste z samoorganizacją drzewa	$\mathbf{O}(N \cdot \lg N)$	$\mathbf{O}(N \cdot \lg N)$
sortowanie przez scalanie	$\mathbf{O}(N \cdot \lg N)$	$\mathbf{O}(N \cdot \lg N)$
Quicksort	$\mathbf{O}(N \cdot \lg N)$	$\mathbf{O}(N^2)$

- Sortowanie przez scalanie jest jednym z najlepszych algorytmów sortowania pod względem pesymistycznej złożoności, choć ma nie najlepszą złożoność pamięciową O(N)
- Średnia złożoności algorytmu Quicksort jest najlepsza ze wszystkich algorytmów sortowania i wynosi **1,4·*N* ·lg *N*** i dlatego jest często stosowany w praktyce
- Algorytm Quicksort oparty jest na metodzie „dziel i zwyciężaj” i można go implementować w kilku wariantach różniących się sposobami wyboru miejsca podziału listy i głębokością rekurencji **http://en.wikipedia.org/wiki/Quicksort**

Przykład analizy złożoności algorytmu

Problem algorytmiczny wyznaczania powłoki wypukłej dla zadanego zbioru N punktów na płaszczyźnie:



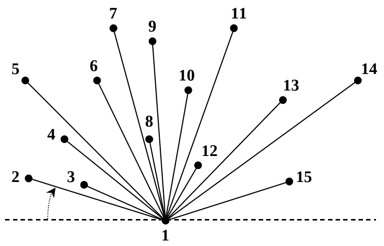
Algorytm „brute force” ma złożoność $\mathbf{O}(N^3)$:

dla każdego z N punktów trzeba dobierać kolejno po jednym z pozostałych N–1 punktów i sprawdzać czy dla prostej, która przechodzi przez taką parę reszta punktów (N–2) leży tylko po jednej jej stronie; $F(N) = N \cdot (N-1) \cdot (N-2)$

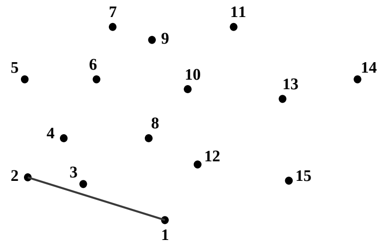
Algorytm „przyrostowy”o niższej złożoności (?)

1. znajdź punkt P_1 o najmniejszej współrzędnej Y,
2. posortuj pozostałe punkty rosnąco według kątów tworzonych przez odcinki $\overline{P_1 P_J}$ z linią poziomą przechodzącą przez P_1 - powstanie lista $P_2, \ldots, P_N$
3. dołącz do powłoki punkty P_1 i P_2 ,
4. **dla** J **od** 3 **do** N **wykonaj co następuje**:
   - 4.1. dołącz do powłoki punkt P_J ,
   - 4.2. cofając się wstecz po odcinkach aktualnej powłoki, usuwaj z niej te punkty P_K , dla których prosta przechodząca przez P_K i P_{K-1} przecina odcinek $\overline{P_1 P_J}$, aż do napotkania pierwszego punktu nie dającego się usunąć.

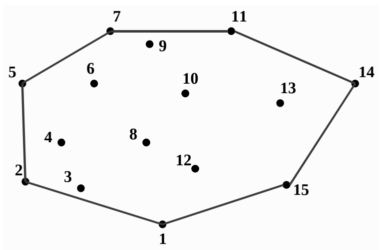
Po 2. kroku algorytmu:



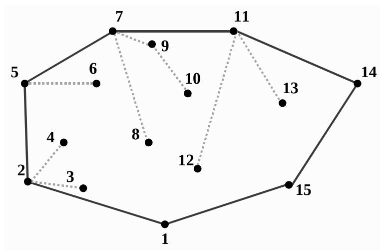
Po 3. kroku algorytmu:



Kolejne iteracje w 4. kroku algorytmu:



Liczba korekt dokonanych podczas iteracji w 4. kroku:



Podsumowanie złożoności algorytmu krok po kroku:

Krok 1.	**O(*N*)**	(wybór pierwszego punktu)
Krok 2.	**O(*N*·lg*N*)**	(sortowanie)
Krok 3.	**O(1)**	(dołączenie 1 odcinka)
Krok 4.	**O(*N*)**	(dołączanie z usuwaniem)
Razem	**O(*N*+*N*·lg*N*+1+*N*) = O(*N*·lg*N*)**	