Algorytm i złożoność obliczeniowa algorytmu

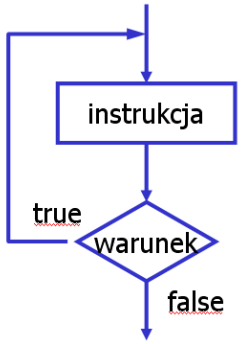
Algorytm - przepis postępowania, którego wykonanie prowadzi do rozwiązania określonego problemu

- określa czynności, jakie należy wykonać
- wyszczególnia **wszystkie** niezbędne czynności oraz **kolejność** ich wykonania
- precyzyjny na tyle, aby posługiwanie się nim polegało na **automatycznym** wykonaniu czynności

Metody zapisu algorytmów (m.in.):

- Pseudokod
  Przypomina język programowania, ale nie jest tak restrykcyjny składniowo; daje się łatwo przełożyć na kod
- Język sieci działań (flow chart)
  Graficzny zapis przy użyciu standardowych symboli (operacja we/wy, operacja, warunek) strzałki pokazują kolejność działań

```
min = t0
FOR n=1, 2, …, N-1
  IF tn < min
  => min = tn
```



Złożoność obliczeniowa to ilość zasobów niezbędna do wykonania algorytmu. Wyróżnia się dwa rodzaje:

- Złożoność czasową (obliczeniową)
- Złożoność pamięciową

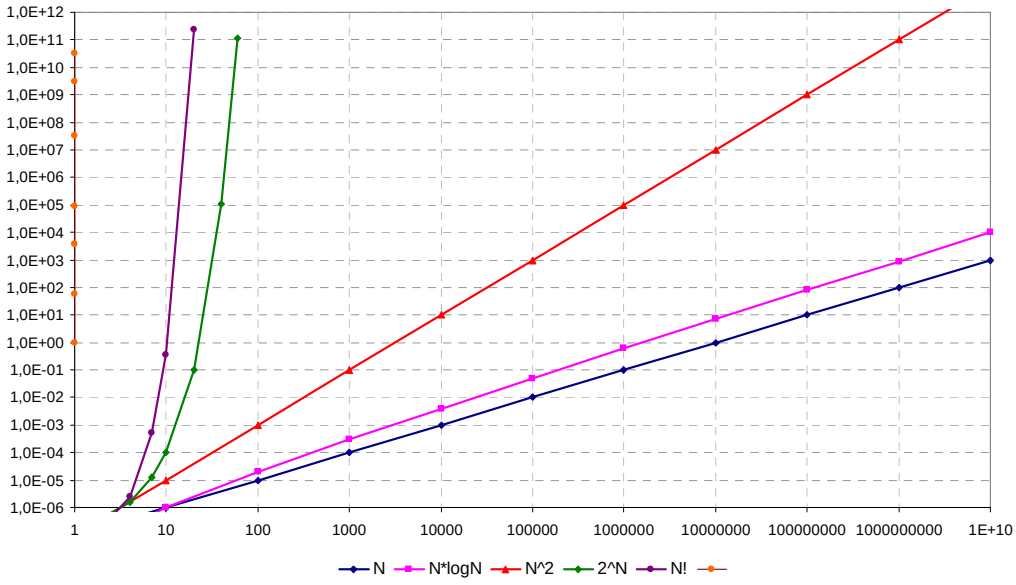
Złożoność zależy od rozmiaru danych wejściowych oraz od algorytmu ich przetwarzania

Istnieje wiele miar złożoności obliczeniowej i wiele notacji. W większości miar najważniejsze jest asymptotyczne tempo wzrostu (tj. jak złożoność rośnie ze wzrostem rozmiaru danych). Do porównywania algorytmów najczęściej używa się notacji O (dużego O), która podaje **asymptotyczne ograniczenie górne**. W notacji O uwzględnia się tylko funkcję dominującą i pomija się współczynniki liczbowe, ponieważ dla odpowiednio dużych zbiorów danych współczynniki stałe mają mniej istotne znaczenie niż rodzaj funkcji dominującej, np.

- f(n) = 100 n2 + 500 => O(n2)
- f(n) = 1000 n => O(n)

Przykładowe złożoności algorytmów, w porządku rosnącym:

- *O(1)* – złożoność stała,
- *O(logn)* – złożoność logarytmiczna,
- *O(n)* – złożoność liniowa,
- *O(nlogn)* – złożoność liniowo-logarytmiczna,
- *O(n2)* – złożoność kwadratowa,
- *O(nk)* – złożoność wielomianowa (*k* jest stałą),
- *O(kn)* – złożoność wykładnicza (*k* jest stałą),
- *O(n!)* – złożoność rzędu silnia



NWD (największy wspólny dzielnik), algorytm Euklidesa

Największy wspólny dzielnik dwóch liczb nie zmienia się, jeżeli od większej liczby odejmujemy mniejszą.

Dla liczb całkowitych *k*, *m* oraz *n* załóżmy, że *k* jest wspólnym czynnikiem liczb *A* oraz *B*
$A = n \cdot k$
$B = m \cdot k$
oraz $A>B$
Różnicę A i B można zapisać jako:
$A - B = (n - m) \cdot k$
więc *k* jest także wspólnym czynnikiem różnicy tych liczb

Szukanie NWD polega na zmniejszaniu A o B (lub odwrotnie, zależnie która jest większa) aż do zrównania się tych liczb, wtedy A=B=NWD. Proces można przyśpieszyć obliczając resztę z dzielenia zamiast różnicy

NWD algorytm Euklidesa - wersja z odejmowaniem

```
WHILE A<>B
  IF A>B
    A <- A-B
  ELSE
    B <- B-A
NWD = A
```

NWD algorytm Euklidesa - wersja z resztą z dzielenia

```
WHILE B<>0
  R = A mod B
  A = B
  B = R
NWD = A (ponieważ B=0)
```

Ciąg Fibonacciego

Ciąg liczb naturalnych, zdefiniowany jako:

$$F_n := \begin{cases} 0 & \text{dla } n = 0; \\ 1 & \text{dla } n = 1; \\ F_{n-1} + F_{n-2} & \text{dla } n > 1. \end{cases}$$

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181

Obliczanie liczby Fibonacciego, z definicji, „w locie”:

```
IF n=0
  Fibonacci = 0
ELSE IF n=1
  Fibonacci = 1
ELSE
  fa = 0
  fb = 1
  FOR i = 2 … N
    fi = fa + fb
    fa = fb
    fb = fi
Fibonacci =fi
```

Zamiast obliczać liczby Fibonacciego "w locie", można zapisywać je w tablicy. Algorytm jest wtedy znacznie prostszy, a dodatkowo może się przydać, jeżeli program kożysta z nich wielokrotnie – wówczas zamiast liczyć je od nowa, można odczytać z tablicy

```
// F[N] – tablica N-elementowa (uwaga! Tu jest pułapka)
F[0] = 0
F[1] = 1
FOR n=2, 3, … (jak daleko???)
  F[n] = …
```

Liczby pierwsze

Liczby które mają dwa naturalne dzielniki: 1 i siebie samą; Mają duże znaczenie w matematyce i informatyce, np. wiele algorytmów kryptograficznych wykorzystuje liczby pierwsze

Najprostszy algorytm sprawdzania czy liczba jest liczbą pierwszą, wynikający wprost z definicji liczby pierwszej, to policzenie naturalnych podzielników – jeżeli są dokładnie 2, to liczba jest pierwsza.Jest to najprostsza, ale też najgorsza (najwolniejsza) metoda

```
Pierwsza (N)
  d = 0
  FOR i = 1, 2, …, N
    IF n mod i = 0
      d = d + 1
  IF d=2
    return TRUE
  ELSE
    return FALSE
```

Sposoby przyśpieszenia sprawdzania liczb pierwszych:

1. **Pominięcie podzielników 1 i N**
   Każda liczba dzieli się przez jeden i siebie, więc po co to sprawdzać?
   Trzeba odpowiednio zmienić pętlę oraz kryterium "pierwszości"
   Zysk minimalny, ale warto zwracać uwagę na takie detale;
2. **Sprawdzanie "do pierwszej wpadki"**
   Tak naprawdę nie jest ważne ILE dokładnie jest podzielników, tylko czy dokładnie 2 czy więcej. Sprawdzanie należy prowadzić w zakresie 2…N i zakończyć po znalezieniu pierwszego podzielnika
3. **Sprawdzanie podzielników do √N**
   Można łatwo wykazać, że jeżeli $p = \sqrt{N}$, to dla każdego podzielnika a, takiego że $a \leq p$, mamy $N/a = b$, gdzie $b \geq p$, Wynika stąd, że wystarczy szukać podzielnika tylko do $\sqrt{N}$ WŁĄCZNIE – jeżeli go nie ma, to powyżej $\sqrt{N}$ też nie będzie. Zmniejszenie złożoności obliczeniowej jest tu duże
4. **Sito Eratostenesa**,
   tj. szukanie podzielnika tylko wśród liczb pierwszych Uwzględniając poprzedni postulat – należy szukać podzielnika tylko wśród liczb pierwszych, mniejszych lub równych √N Implementacja jest trudna, ponieważ najpierw trzeba znaleźć liczby pierwsze ≤ √N (a nie wiadomo dokładnie ile ich jest), zapisać np. w tablicy i dopiero sprawdzać, czy wśród nich jest podzielnik N

Pomiar czasu wykonywania programu w C#

W programach C# można łatwo zmierzyć czas wykonywania obliczeń przy pomocy "stopera" z biblioteki Diagnostics:

```
System.Diagnostics.Stopwatch watch;
watch = System.Diagnostics.Stopwatch.StartNew();
// działania, których czas jest mierzony
watch.Stop();
Console.WriteLine("Czas: {0}", watch.Elapsed);
```

Dokładność pomiaru czasu jest bardzo duża, rzędu ułamków mikrosekundy - wynika z częstotliwości taktowania procesora.

Zamiast powtarzać nazwę biblioteki (*System.Diagnostics*) przy każdym odwołaniu do klasy Stopwatch, można ją dopisać na samym początku programu w klauzuli using:

```
using System.Diagnostics;
```

Metoda statyczna *StartNew()* tworzy za każdym użyciem nową instancję; zamiast tego można utworzyć instancję raz, a następnie resetować i zarazem uruchamiać stoper metodą *Restart()*:
using *System.Diagnostics*;

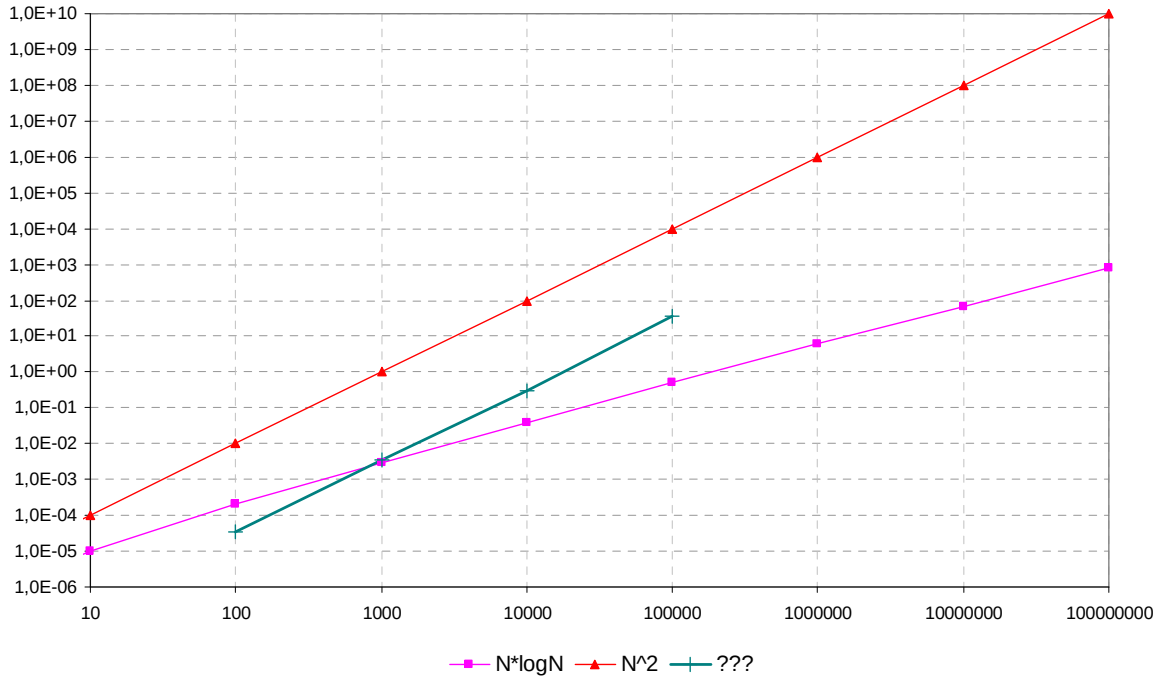
```
Stopwatch watch;
watch = new Stopwatch ();

watch.Restart();
// działania, których czas jest mierzony
watch.Stop();
Console.WriteLine("Czas: {0}", watch.Elapsed);
```

Zadania

Proszę napisać program, który…

1. Znajduje NDW dwóch liczb podanych przez użytkownika za pomocą algorytmu Euklidesa – przy pomocy różnic i reszt z dzielenia
   Program należy też uruchomić w trybie debugowania (krokowym) i obserwować wartości zmiennych w oknie *locals* lub *watches*
2. Wykonuje zadania z punktu 1 i dodatkowo zlicza iteracje („obroty pętli") wykonane podczas wyznaczania NWD. Który algorytm jest lepszy i dlaczego? Jak zmienia się liczba iteracji przy rosnących wartościach A i B? A przy rosnącej różnicy A i B? Rosnącym ilorazie A i B?
3. * Znajduje parę liczb względnie pierwszych (NWD=1), większych od liczby podanej przez użytkownika. Należy opracować dwa warianty algorytmu – częściowo stochastyczny (jedna liczba jest równa liczbie podanej przez użytkownika, druga losowana, do skutku) i całkowicie stochastyczny (obie liczby są losowane, do skutku).
4. * Wykonuje zadania z punktu 3 tak, aby dodatkowo mierzyć czas obliczeń. Który wariant algorytmu jest szybszy?
5. Wyznacza n-ty element ciągu Fibonacciego (n podaje użytkownik)
6. Wyznacza n kolejnych liczb Fibonacciego i zapisuje je do tablicy. Po zakończeniu obliczeń wszystkie wyznaczone liczby należy wyświetlić w oknie konsoli
7. Sprawdza, czy liczna podana przez użytkownika jest liczbą pierwszą; należy zastosować najprostszy algorytm testu pierwszości
8. Wyświetla wszystkie liczby pierwsze zawarte w przedziale wskazanym przez użytkownika (od A do B albo od 1 do A); należy zastosować najprostszy algorytm testu pierwszości
9. Wyznacza liczbę liczb pierwszych w przedziale wskazanym przez użytkownika (od A do B albo od 1 do A); należy zastosować najprostszy algorytm testu pierwszości
10. Wykonuje zadania z punktu 9, ale stosując szybsze wersje algorytmu (należy zachować je wszystkie, będą potrzebne do zadania 12). Należy też zmierzyć i porównać czasy wykonania każdego algorytmu dla tej samej wielkości przedziału liczb.
11. * Znajduje pierwszych N liczb pierwszych większych od liczby A i zapisuje je do tablicy. Liczby N i A podaje użytkownik. Po zakończeniu obliczeń wszystkie wyznaczone liczby należy wyświetlić w oknie konsoli



12. * Wykonuje zadania z punktu 10 w sposób pozwalający eksperymentalnie ocenić złożoność obliczeniową poszczególnych algorytmów. Należy sporządzić w programie Excel wykres funkcji $k \cdot n^2$ oraz $k \cdot n \log n$, tak aby stałą k można było zmieniać. Nanieść na wykres wyniki pomiaru czasu wykonania różnych algorytmów i ocenić złożoność obliczeniową. Wykres powinien mieć obie skale logarytmiczne. Zmieniając stałą k można przesuwać linie $O(n^2)$ i O(nlogn) w górę lub w dół. Złożoność obliczeniowa wynika z **nachylenia** linii, a nie wartości, np. na przykładowym wykresie powyżej złożoność algorytmu *???* jest $O(n^2)$, ponieważ nachylenie linii jest prawie takie samo, jak dla linii N^2. To, że linia *???* leży bliżej linii *N*logN*, jest bez znaczenia.