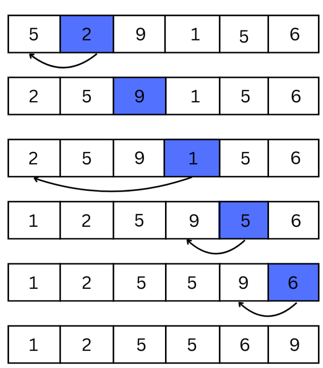
Algorytmy i Struktury Danych

Wojciech Typer

Algorithm 1 Insertion Sort

procedure InsertionSort(A, n)
    for $i = 1$ to $n - 1$ **do**
        $key = A[i]$
        $j = i - 1$
        while $j \geq 0$ and $A[j] > key$ **do**
            $A[j + 1] = A[j]$
            $j = j - 1$
        end while
        $A[j + 1] = key$
    end for
end procedure

Złożoność czasowa: $O(n^2)$
Best case: w najlepszym przypadku złożoność czasowa będzie wynosić $O(n)$
Złożoność pamięciowa: $O(1)$



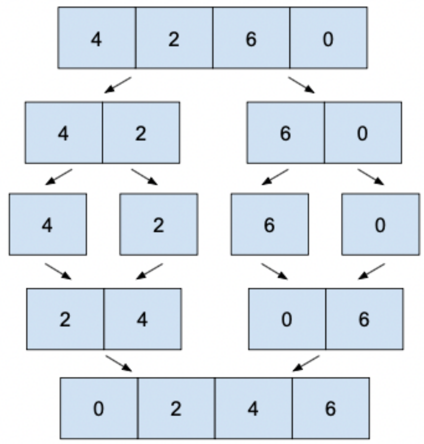
Algorithm 2 Merge Sort

```
procedure MergeSort(A, 1, n)
    if |A[1..n]| == 1 then
        return A[1..n]
    else
        B = MergeSort(A, 1, ⌊n/2⌋)
        C = MergeSort(A, ⌊n/2⌋, n)
        return Merge(B, C)
    end if
end procedure
```

Algorithm 3 Merge

```
procedure Merge(X[1..k], Y[1..n])
    if X = ∅ then
        return Y
    else if Y = ∅ then
        return X
    else if X[1] ≤ Y[1] then
        return [X[1]] × Merge(X[2..k], Y[1..n])
    else
        return [Y[1]] × Merge(X[1..k], Y[2..n])
    end if
end procedure
```

Złożoność czesowa Merge Sort: $O(n \log n)$
Złożoność pamięciowa Merge Sort: $O(n)$



Istnieje również iteracyjna wersja algorytmu Merge, sort, która została przedstawiona poniżej w postaci pseudokodu.

Algorithm 4 IterativeMergeSort

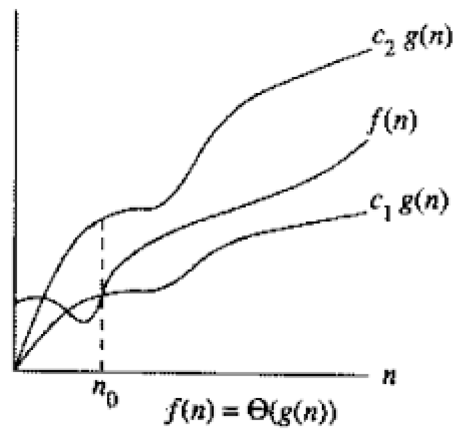
---

procedure ITERATIVEMERGESORT(A[1..n])
  for $size = 1$ **to** $n - 1$ **by** $size \times 2$ **do**
    for $left = 0$ **to** $n - 1$ **by** $2 \times size$ **do**
      $mid \leftarrow \min(left + size - 1, n - 1)$
      $right \leftarrow \min(left + 2 \times size - 1, n - 1)$
      MERGE(A, left, mid, right)
    end for
  end for
end procedure

Złożoność czasowa Iterative Merge Sort: $O(n \log n)$ - dzieje się tak, ponieważ size jest podwajany o 2 w każdej iteracji, więc potrzebujemy około $\log_2 n$ iteracji, a w każdej z nich wykonujemy $O(n)$ operacji.

Złożoność pamięciowa Iterative Merge Sort: $O(n)$

Notacja asymptotyczna O:f(n) = O(g(n)) $\rightarrow (\exists c > 0)(\exists n_0 \in N) : \forall n \geq n_0 \rightarrow 0 \leq f(n) \leq c \cdot g(n)$



$f(n) = O(g(n)) \to lim_{n \to \infty} \frac{|f(n)|}{|g(n)|} < \infty$

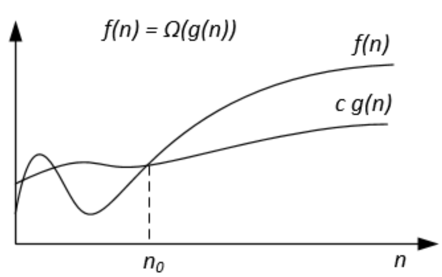
Notacja asymptotyczna - własności

a) $f(n) = n^3 + O(n^2) \to (\exists h(n) = O(n^2))(f(n) = n^3 + h(n))$

b) $n^2 + O(n) = O(n^2) \to (\forall f(n) = O(n))(\exists h(n) = O(n^2))(n^2 + f(n) - h(n))$

Notacja Ω

$f(n) = \Omega(g(n)) \to (\exists c > 0)(\exists n_0 \in N)(\forall n \geq n_0)(c * g(n) \leq |f(n))$



Notacja Ω - własności

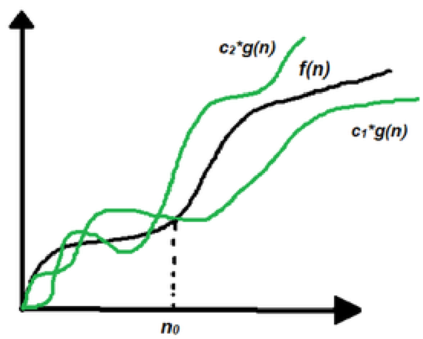
a) $n^3 = \Omega(2n^2)$

b) $n = \Omega(log(n))$

c) $2n^2 = \Omega(n^2)$

Notacja Θ

$f(n) = \Theta(g(n)) \rightarrow (\exists c_1, c_2 > 0)(\exists n_0 \in N)(\forall n \geq n_0)(c_1 g(n) \leq |f(n) \leq c_2 g(n))$



$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

Notacja o- małe

$f(n) = o(g(n)) \rightarrow (\forall c > 0)(\exists n_0 \in N)(\forall n \geq n_0)(|f(n)| < c * |g(n)|)$

Notacja o- małe - przykłady

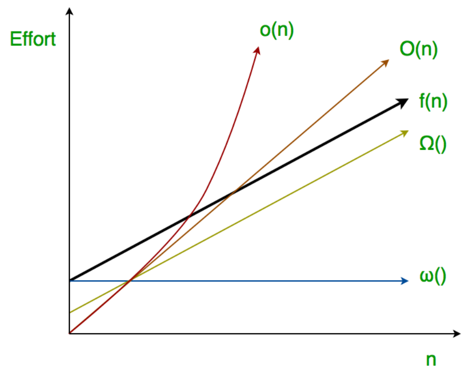
a) $117nlog(n) = o(n^2)$

b) $n^2 = o(n^3)$

Notacja ω

$f(n) = \omega(g(n)) \rightarrow (\forall c > 0)(\exists n_0 \in N)(\forall n \geq n_0)(|f(n)| > c * |g(n)|)$

$lim_{n \rightarrow \infty} \frac{|f(n)|}{|g(n)|} = \infty$



Rekurencje

Metoda podstawiania (metoda dowodzenia indukcyjnego)

1. Zgadnij odpowiedź (bez stałych)
2. Sprawdź przez indukcję, czy dobrze zgadliśmy
3. Znajdź stałe

Przykład 1:

$T(n) = 4T(\frac{n}{2}) + n$
Pierwszy strzał: $T(n) = O(n^3)$
Cel: pokazać, że $(\exists c > 0)T(n) \leq c * n^3$
Krok początkowy: $T(1) = \Theta(1) = c * 1^3 = c$
Krok indukcyjny: zał. że, $(\forall_(k < n))(T(k) \leq c * k^3) =$
Dowód: $T(n) = 4T(\frac{n}{2}) + n \leq 4c * (\frac{n}{2})^3 + n = \frac{1}{2}cn^3 + n =$
$= cn^3 - \frac{1}{2}cn^3 + n = cn^3 - (\frac{1}{2}cn^3 - n) \leq cn^3$
Pokazaliśmy, że $T(n) = O(n^2)$

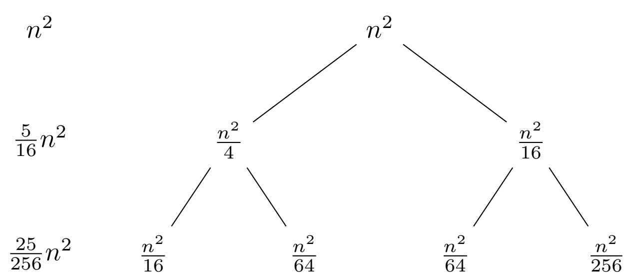
Spróbujmy wzmocnić zał. indukcyjne: $T(n) \leq c_1n^2 - c_2n$
$T(n) \leq 4T(\frac{n}{2}) + n \leq 4(c_1(\frac{n}{2})^2 - c_2\frac{n}{2}) + n =$
$= c_1n^2 - 2c_2n + n = c_1n^2 - (2c_2 - 1)n \leq c_1n^2 - c_2n$
Musimy dobrać takie c_1ic_2, aby $2c_1 \geq c_2$
Wówczas otrzymamy $T(1) = O(1) \leq c_11^2 - c_21$

Przykład 2:

$T(n) = 2T(\sqrt{n}) + log(n)$
Załóżmy, że n jest potęgą dwójki $n = 2^m \to m = log(n)$
$T(2^m) = 2T(2^{m/2}) + m$
oznaczmy $T(2^m) = S(m)$
$T(2^m) = 2T(2^{m/2}) + m \to 2S(m/2) + m$
$S(m) = O(mlog(m))$
$T(n) = O(log(n)log(log(n)))$ (formalnie powinniśmy to udowodnić)

Drzewo rekursji

Przykład : $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + n^2$



Trzeba pamiętać, że drzewo rekursji samo w sobie nie jest formalnym rozwiązaniem problemu. Nie można go urzywać do dowodzenia złożoności algorytmów. Jest to jedynie intuicyjne podejście do problemu. Formmalnie T(n) należałoby policzyć jako sumę wszystkich wierzchołków w drzewie rekursji:

$$T(n) = \sum_{k=0}^{\infty} \left(\frac{5}{16}\right)^k \cdot n^2 = n^2 \sum_{k=0}^{\infty} \left(\frac{5}{16}\right)^k = n^2 \frac{1}{1 - \frac{5}{16}} = n^2 \frac{16}{11} = \frac{16}{11} n^2$$

Widzimy zatem, że $T(n) = O(n^2)$

Master Theorem

Niech $a \geq 1, b > 1, f(n), d \in N$ oraz $f(n)$ będzie funkcją nieujemną. Rozważmy rekurencję:

$$T(n) = aT(\frac{a}{b}) + \Theta(n^d)$$

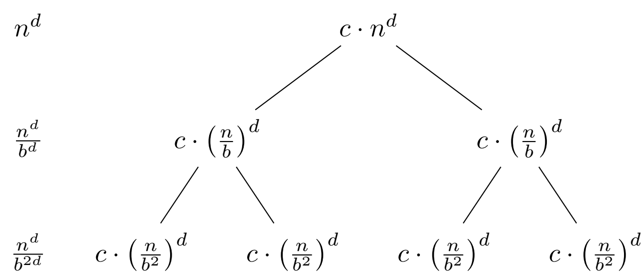
Wówczas:

- $\Theta(n^d)$, jeśli $d > log_b a$

- $\Theta(n^d log(n))$, jeśli $d = log_b a$
- $\Theta(n^{log_b a})$, jeśli $d < log_b a$

Do przedstawienia problemu użyjemy drzewa rekursji. Rozważmy rekurencję:

$$T(n) = aT(\frac{n}{b}) + \Theta(n^d)$$



1. suma kosztoów w k–tym kroku

$$a^k c(\frac{n}{b^k})^d = c(\frac{a}{b^d})^k n^d$$

gdzie $c(\frac{n}{b^k})^d$ to koszt jednego podproblemu w k–tym kroku

2. obliczenie wysokości drzewa:

$$\frac{n}{b^h} = 1 \to h = \log_b n$$

3. Obliczenie $T(n)$

$$\begin{aligned} T(n) &= \Theta\left(\sum_{k=0}^{\log_b n} c\frac{a}{b^k} n^d\right) \\ &= \Theta\left(c \cdot n^d \sum_{k=0}^{\log_b n} \left(\frac{a}{b^d}\right)^k\right) \\ &= \Theta\left(c \cdot n^d \frac{1 - \left(\frac{a}{b^d}\right)^{\log_b n + 1}}{1 - \frac{a}{b^d}}\right) \\ &\implies T(n) = \Theta(n^d) \end{aligned}$$

4. rozważmy 3 przypadki:
   (a) $d > \log_b a$
   $$T(n) = \Theta(n^d)$$
   (b) $d = \log_b a$
   $$T(n) = \Theta(n^d \log n)$$
   (c) $d < \log_b a$
   $$T(n) = \Theta(n^{\log_b a})$$

Przykłady

- $T(n) = 4T(\frac{n}{2}) + 11n$
  Wtedy kożystając z **Master Theorem** mamy:
  $$a = 4, b = 2, d = 1$$
  Jak i również
  $$\log_b a = \log_2 4 = 2 > 1 = d \implies T(n) = \Theta(n^2)$$
- $T(n) = 4T(\frac{n}{3}) + 3n^2$
  Wtedy
  $$a = 4, b = 3, d = 2$$
  Jak i również
  $$\log_b a = \log_3 4 < 2 = d \implies T(n) = \Theta(n^2)$$
- $T(n) = 27T(\frac{n}{3}) + \frac{n^2}{3}$
  Wtedy
  $$a = 27, b = 3, d = 2$$
  Jak i również
  $$\log_b a = \log_3 27 = 3 > 2 = d \implies T(n) = \Theta(n^3 \log n)$$

Metoda dziel i zwyciężaj (D&C)

Na czym ona polega?

1. Podział problemu na mniejsze podproblemy
2. Rozwiazanie rekurencyjnie mniejsze podpoblemy
3. połącz rozwiązania podproblemów w celu rozwiązania problemu wejściowego

Algorytm – Binary Search

- **Input**: posortowania tablica `A[1..n]` oraz element `x`
- **Output**: indeks `i` taki, że `A[i] = x` lub `0` jeśli `x` nie występuje w `A`
- przebieg algorytmu:

Algorithm 5 Binary Search

```
procedure BinarySearch(A, x)
    l = 1
    r = |A|
    while l ≤ r do
        m = ⌊(l+r)/2⌋
        if A[m] = x then
            return m
        else if A[m] < x then
            l = m + 1
        else
            r = m - 1
        end if
    end while
    return 0
end procedure
```

- **Asypmtotyka** Algorytm spełnia następująca rekurencje:

$$T(n) = T(\frac{n}{2}) + \Theta(1)$$

Rozwiązując za pomocą **Master Theorem** otrzymujemy:

$$T(n) = \Theta(\log n)$$