Wykład 13
Zasada dziel i zwyciężaj

Zasada dziel i zwyciężaj

- schemat metody
- analiza złożoności
- przykłady algorytmów i ich analiza

Zasada dziel i zwyciężaj

Schemat metody

Metoda rozwiązywania problemów „dziel i zwyciężaj" (*divide and conquer*) polega na podzieleniu zadania na mniejsze podzadania, a po ich rozwiązaniu, przeprowadzeniu syntezy rozwiązania całego zadania z rozwiązań mniejszych podzadań:

```
1:  function RozwiązanieZłożonegoProblemu(P)
2:      if problem P ma proste rozwiązanie then
3:          return  Rozwiązanie(P)
4:      else
5:          podziel problem P na k podproblemów P_1, P_2, ..., P_k
6:          for i ← 1, k do
7:              R_i ← RozwiązanieZłożonegoProblemu(P_i)
8:          end for
9:          return Synteza(R_1, R_2, ..., R_k)
10:     end if
11: end function
```


Zasada dziel i zwyciężaj

Analiza złożoności

Theorem (O rekurencji uniwersalnej)

Załóżmy, że a i b są liczbami dodatnimi oraz c dodatnią liczbą naturalną takimi, że $T(n)$ jest funkcją, która dla n będącego potęgą liczby c spełnia równanie:

$$T(n) = \begin{cases} b & gdy\ n \leq c, \\ a \cdot T(\frac{n}{c}) + b \cdot n & gdy\ n > c. \end{cases}$$

Wtedy ze względu na relację między wartościami a i c mamy:

1. *Jeśli $a < c$, to $T = O(n)$.*
2. *Jeśli $a = c$, to $T = O(n \log n)$.*
3. *Jeśli $a > c$, to $T = O(n^{\log_c a})$.*

Zasada dziel i zwyciężaj

Analiza złożoności

- Funkcja $T(n)$, z twierdzenia o rekurencji uniwersalnej, może wyrażać liczbę operacji wykonywanych przez funkcję RozwiązanieZłożonegoProblemu(P), dla problemu P rozmiaru n.
- Składnik $b \cdot n$ odpowiada liczbie operacji wykonywanych przez funkcję *Synteza*().
- Stała a odpowiada liczbie podproblemów na jaką dzielimy problem P, natomiast stała c wskazuje ilu krotnie mniejszych rozmiarów są to podproblemy, w stosunku do rozmiaru problemu P.

Example

Jeśli dzielimy problem P na trzy podproblemy o połowę mniejszych rozmiarów, to $a = 3$ i $c = 2$.

Zasada dziel i zwyciężaj

Przykłady analizy: minimalny i maksymalny element

Wyszukiwanie najmniejszej wartości *min* w tablicy $A[1..n]$ można wykonać za pomocą $n - 1$ porównań:

```
1: min ← A[1];
2: for i ← 2, n do
3:     if A[i] < min then
4:         min ← A[i]
5:     end if
6: end for
```

Dla znalezienie wartości największej można wykonać kolejnych $n - 1$ porównań.
Wówczas wykonanych zostanie $2 \cdot n - 2$ porównań.

Zasada dziel i zwyciężaj

Przykłady analizy: minimalny i maksymalny element

```
1: procedure MinMax(A, L, P, out min, out max)
2:     if L = P then
3:         min ← A[L]; max ← A[L]
4:     else
5:         if L + 1 = P then
6:             niech min i max będą odpowiednio mniejszą i większą spośród
   wartości A[L] i A[P];
7:         else
8:             S ← (L + P) div 2;
9:             MinMax(A, L, S, min_1, max_1);
10:            MinMax(A, S + 1, P, min_2, max_2);
11:            min ← minimum(min_1, min_2);
12:            max ← maximum(max_1, max_2);
13:        end if
14:    end if
15: end procedure
```


Zasada dziel i zwyciężaj

Przykłady analizy: minimalny i maksymalny element

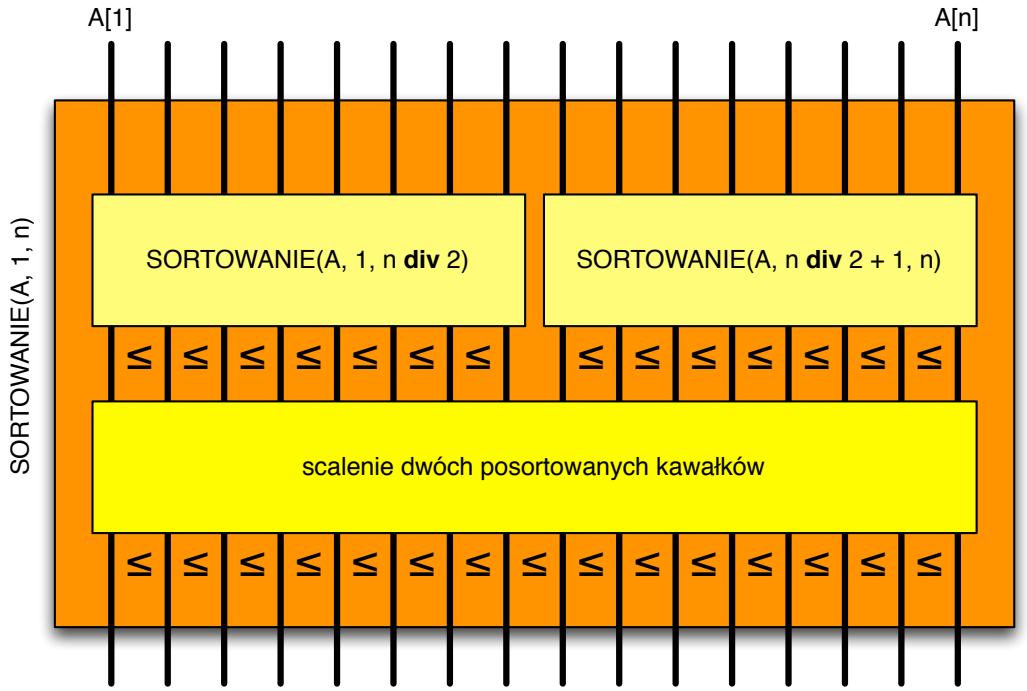
Załóżmy, że mamy znaleźć minimalne i maksymalne wartości z tablicy rozmiaru $n = 2^k$. Niech $T(n)$ oznacza liczbę potrzebnych do tego celu porównań.
Jeśli $k = 1$, to wystarczy jedno porównanie, zatem $T(2) = 1$.
Załóżmy teraz, że mamy daną tablicę A długości $n = 2^k$. Za pomocą $T(n/2)$ porównań możemy wyznaczyć minimum min_1 i maksimum max_1 z $A[1..n/2]$ oraz za pomocą tej samej liczby porównań możemy wyznaczyć odpowiednie wartości min_2 i max_2 z tablicy $A[n/2 + 1..n]$.
Jeśli $min_1 < min_2$ to minimalną wartości jest min_1 a w przeciwnym przypadku jest nią min_2. Podobnie, jeśli $max_1 > max_2$, to maksymalną wartością jest max_1 a w przeciwnym wypadku jest nią max_2. Zatem

$$T(n) = 2 \cdot T(n/2) + 2.$$

Łatwo sprawdzić metodą indukcji matematycznej, że dla n będących potęgami dwójki mamy $T(n) = \frac{3}{2}n - 2$, co jest mniejsze od $2n - 2$.

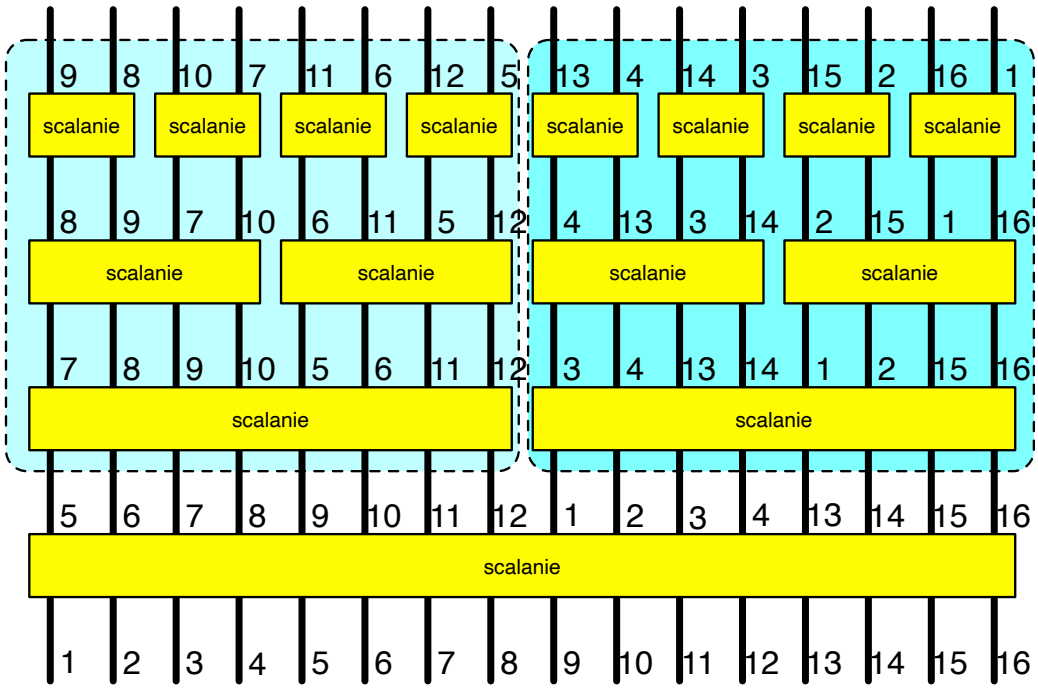
Zasada dziel i zwyciężaj

Przykłady analizy: sortowanie przez scalanie



Zasada dziel i zwyciężaj

Przykłady analizy: sortowanie przez scalanie



Zasada dziel i zwyciężaj

Przykłady analizy: sortowanie przez scalanie

Omówić źródła programu **scalsort.c** z repozytorium **c-clang-wip**.

Zasada dziel i zwyciężaj

Przykłady analizy: sortowanie przez scalanie

Theorem

Tablicę rozmiaru n można posortować za pomocą $O(n \log n)$ porównań i podstawień.

Dowód.

- Niech $C(n)$ oznacza liczbę porównań i podstawień podczas sortowania n elementów.
- Załóżmy, że n jest potęgą liczby 2.
- Wówczas $C(1) = 0$, natomiast $C(n) = 2 \cdot C(n/2) + O(n)$.
- Zatem, z twierdzenia o rekurencji uniwersalnej, $C(n) = O(n \log n)$.

□

Zasada dziel i zwyciężaj

Przykłady analizy: operacje na dużych liczbach

- Nieujemne liczby całkowite można przechowywać w tablicy n-elementowej, w której $A[0]$ jest najmniej znaczącą cyfrą dziesiętną, natomiast $A[n-1]$ jest najbardziej znaczącą cyfrą dziesiętną.
- Wartość zapisana w takiej tablicy jest równa:

$$\sum_{i=0}^{n-1} 10^i \cdot A[i].$$

Zasada dziel i zwyciężaj

Przykłady analizy: operacje na dużych liczbach

Wynik dodawania dwóch n-cyfrowych liczb może mieć $n+1$ cyfr i dlatego poniższa funkcja zwraca ostatnie przeniesienie, na które nie ma miejsca w tablicy C, jako swoją wartość:

```
1: function Dodaj(A, B, out C, n)
2:     przeniesienie ← 0;
3:     for i ← 0, n − 1 do
4:         suma ← A[i] + B[i] + przeniesienie;
5:         C[i] ← suma mod 10;
6:         przeniesienie ← suma div 10
7:     end for
8:     Dodaj ← przeniesienie
9: end function
```

Funkcja zawiera jedną pętlę wykonywaną n razy, zatem jej czasowa złożoność obliczeniowa jest $O(n)$.

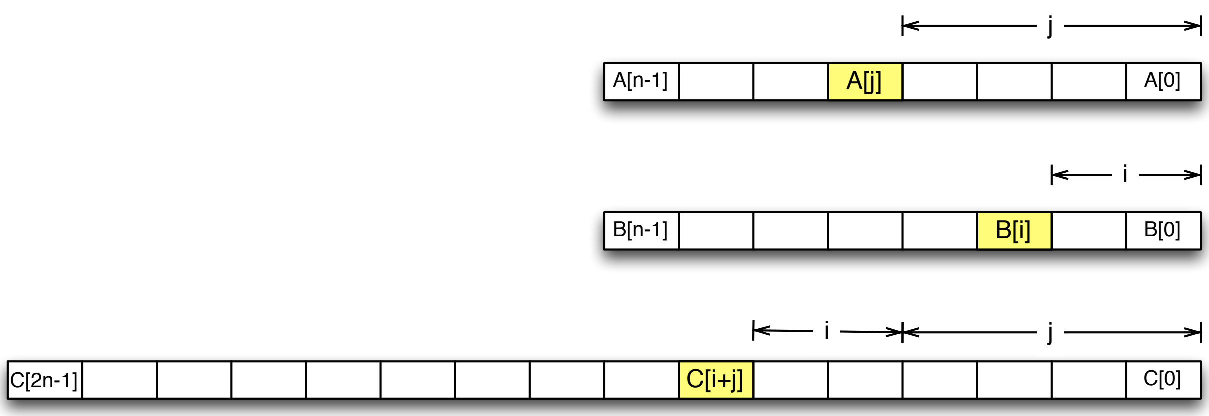
Zasada dziel i zwyciężaj

Przykłady analizy: operacje na dużych liczbach

```
         76147654
x            2611
-----------------
         76147654  =  1*76147654*1
        761476540  =  1*76147654*10
      45688592400  =  6*76147654*100
+    152295308000  =  2*76147654*1000
-----------------
     198821524594
```


Zasada dziel i zwyciężaj

Przykłady analizy: operacje na dużych liczbach



suma ← C[i+j] + A[j]*B[i] + przeniesienie
C[i+j] ← suma **mod** 10
przeniesienie ← suma **div** 10

Zasada dziel i zwyciężaj

Przykłady analizy: operacje na dużych liczbach

Wynik mnożenie dwóch liczb n-cyfrowych może mieć $2 \cdot n$ cyfr:

```
1:  procedure Pomnóż(A, B, out C, n)
2:      for i ← 0, 2 * n − 1 do
3:          C[i] ← 0
4:      end for
5:      for i ← 0, n − 1 do
6:          przeniesienie ← 0;
7:          for j ← 0, n − 1 do
8:              suma ← C[i + j] + B[i] * A[j] + przeniesienie;
9:              C[i + j] ← suma mod 10;
10:             przeniesienie ← suma div 10;
11:         end for
12:     end for
13: end procedure
```


Zasada dziel i zwyciężaj

Przykłady analizy: algorytm szybkiego mnożenia Karatsuby

- Omówimy teraz szybszą metodę, wymyśloną przez Karatsubę w roku 1962.
- Załóżmy, że chcemy pomnożyć dwie długie liczby x i y całkowite zapisane w układzie o podstawie 10.
- Załóżmy, że obie liczby mają parzystą liczbę $n = 2m$ cyfr (jeśli nie, to dodajmy zera z ich lewej strony). Możemy je zapisać w postaci

$$x = 10^m \cdot x_1 + x_2, y = 10^m \cdot y_1 + y_2.$$

gdzie wszystkie liczby x_1, x_2, y_1, y_2 są już m-cyfrowe.

- Wtedy

$$x \cdot y = 10^{2m} x_1 y_1 + 10^m (x_1 y_2 + x_2 y_1) + x_2 y_2,$$

więc powinniśmy umieć szybko wyznaczyć liczby $x_1 y_1$, $x_1 y_2 + x_2 y_1$ i $x_2 y_2$.

Zasada dziel i zwyciężaj

Przykłady analizy: algorytm szybkiego mnożenia Karatsuby

- Karatsuba zauważył, że cztery iloczyny x_1y_1, $x_1y_2 + x_2y_1$ i x_2y_2 można wyznaczyć wykonując trzy mnożenia.
- Niech $A = x_1y_1$, $B = x_2y_2$ oraz $C = (x_1 + x_2)(y_1 + y_2)$.
- Liczbę $x_1y_2 + x_2y_1$ można wyliczyć z liczb A, B i C za pomocą jednego dodawania i jednego odejmowania:

$$C - (A + B) = x_1y_2 + x_2y_1.$$

Zasada dziel i zwyciężaj

Przykłady analizy: algorytm szybkiego mnożenia Karatsuby

- Aby wyznaczyć produkty liczb m-cyfrowych można zastosować rekurencyjnie ten sam trik.
- Niech T(n) oznacza czas potrzebny do pomnożenia dwóch n-cyfrowych liczb metodą Karatsuby. Wtedy

$$T(n) = 3 \cdot T\left(\frac{n}{2}\right) + b \cdot n.$$

dla pewnej stałej b.

Theorem

Algorytm Karatsuby działa w czasie $O(n^{\log_2 3})$.

Zasada dziel i zwyciężaj

Przykłady analizy: algorytm szybkiego mnożenia Karatsuby

Dowód.

Na mocy twierdzenia o rekursji uniwersalnej, jeśli funkcja T spełnia równanie $T(n) = 3 \cdot T(\frac{n}{2}) + b \cdot n$, to $T(n) = O(n^{\log_2 3})$ dla liczb n będących potęgą liczby c. □

Zauważ, że $\log_2 3 = \frac{\log_{10} 3}{\log_{10} 2} \simeq 1.585$, zatem $n^{\log_2 3} = o(n^2)$. Algorytm Karatsuby jest więc znacznie szybszy od prostego „ręcznego" algorytmu działającego w czasie $O(n^2)$.
Znane są jeszcze szybsze metody mnożenia dużych liczb naturalnych. Algorytm Schönhagena - Strassena z roku 1972 działa w czasie $O(n \cdot \log n \cdot \log \log n)$. Wykorzystuje on stosunkowo zaawansowane narzędzia matematyczne, a mianowicie transformacje Fouriera.