

# AiSD

Rafał Włodarczyk

INA 4, 2025

## Contents

<b>1</b>	<b>Lecture I - Sortowanie</b>	<b>1</b>
1.1	Worst-case analysis . . . . .	2
1.2	Average-case analysis . . . . .	2
1.3	Analiza losowego sortowania . . . . .	2
1.4	Insertion Sort $(A, n)$ . . . . .	2
1.4.1	Worst-case analysis - Insertion Sort $(A, n)$ . . . . .	3
1.4.2	Average-case analysis - Insertion Sort $(A, n)$ . . . . .	3
1.5	Przykład złożoności . . . . .	3
<b>2</b>	<b>Lecture II - Merge Sort</b>	<b>4</b>
2.1	Merge sort $(A, 1, n)$ . . . . .	4
<b>3</b>	<b>Lecture III - Narzędzia do analizy algorytmów</b>	<b>6</b>
3.1	Notacja asymptotyczna . . . . .	6
3.2	Notacja Big- $O$ . . . . .	6
3.3	Notacja Big- $\Omega$ . . . . .	7
3.4	Notacja Big- $\Theta$ . . . . .	7
3.5	Notacja small- $o$ . . . . .	8
3.6	Notacja small- $\omega$ . . . . .	8
3.7	Metody rozwiązywania rekurencji . . . . .	8
3.8	Rozwiązywanie rekurencji . . . . .	9
3.9	Metoda podstawiania - Metoda dowodu indukcyjnego . . . . .	9
<b>4</b>	<b>Lecture IV - Metoda drzewa rekursji</b>	<b>10</b>
4.1	Metoda drzewa rekursji . . . . .	10
4.2	Metoda iteracyjna . . . . .	12
4.3	Master Theorem . . . . .	12
4.4	Divide and Conquer . . . . .	14
4.5	Wyszukiwanie elementów w portowanej tablicy . . . . .	15
4.6	Binary search . . . . .	15

## 1 Lecture I - Sortowanie

Definiujemy problem:

1. Input:  $A = (a_1, \dots, a_n), |A| = n$
2. Output: Permutacja tablicy wyjściowej  $(a'_1, a'_2, \dots, a'_n)$ , takie że:  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

### 1.1 Worst-case analysis

$$T(n) = \max_{\text{wszystkie wejścia}} \{\# \text{operacji po wszystkich } |n| \text{-wejściach}\} \quad (1)$$

### 1.2 Average-case analysis

Zakładamy pewien rozkład prawdopodobieństwa na danych wejściowych. Z reguły myślimy o rozkładzie jednostajnym. Niech  $T$  - zmienna losowa liczby operacji wykonanych przez badany algorytm.

$$\mathbf{E}(T) - \text{wartość oczekiwana } T \quad (2)$$

Później możemy badać wariancję, oraz koncentrację.

### 1.3 Analiza losowego sortowania

Dla poprzedniego algorytmu zobaczmy, że:  $n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$  [czyli  $f(n) \sim g(n) \equiv \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$ ].  
To jest tragiczna złożoność.

### 1.4 Insertion Sort $(A, n)$

$(A, n) = ((a_1, a_2, \dots, a_n), n)$

```
for j = 2...n
{
    key = A[j]
    i=j-1
    while(i>0 && A[i]>key) {
        A[i+1] = A[i]
        i = i - 1
    }
    A[i+1] = key
}
```

Przykład:  $A = (8, 2, 4, 9, 3, 6), n = 6$

- $8_i, 2_j, 4, 9, 3, 6$   $j = 2, i = 1, key = 2$  while
- $2, 8_j, 4, 9, 3, 6$
- $2, 8_i, 4_j, 9, 3, 6$   $j = 3, i = 2, key = 4$  while
- $2, 4, 8, 9, 3, 6$
- $2, 4, 8_i, 9_j, 3, 6$   $j = 4, i = 3, key = 9$  no while

- 2, 4, 8, 9<sub>i</sub>, 3<sub>j</sub>, 6     $j = 5, i = 4, key = 3$  while
- 2, 3, 4, 8, 9, 6
- 2, 3, 4, 8, 9<sub>i</sub>, 6<sub>j</sub>     $j = 6, i = 5, key = 6$  while
- 2, 3, 4, 6, 8, 9

```
| <= x | > x | x | ... |
| <= x | x | > x | ... |
```

Porównujemy element ze wszystkim co jest przed nim - wszystko przed  $j$ -tym elementem będzie posortowane. Insertion sort nie swapuje par elementów w tablicy, a przenosi tam gdzie jest jego miejsce.

#### 1.4.1 Worst-case analysis - Insertion Sort ( $A, n$ )

Odwrotnie posortowana tablica powoduje najwięcej przesunięć. Ponieważ ustaliliśmy że liczba operacji w while zależy od  $j$ , wtedy:

$$T(n) = \sum_{j=2}^n O(j-1) = \sum_{j=1}^{n-1} O(j) = O\left(\sum_{j=1}^{n-1} j\right) = \quad (3)$$

$$= O\left(\frac{1+n-1}{2} \cdot (n-1)\right) = O\left(\frac{(n-1) \cdot (n)}{2}\right) = O\left(\frac{n^2}{2}\right) = O(n^2) \quad (4)$$

c

#### 1.4.2 Average-case analysis - Insertion Sort ( $A, n$ )

Policzmy dla uproszczenia, że na wejściu mamy  $n$ -elementowe permutacje, z których każda jest jednakowo prawdopodobna  $p = \frac{1}{n!}$ . Spróbujmy wyznaczyć  $\mathbf{E}$ , korzystając z inwersji permutacji. Wartość oczekiwana liczby inwersji w losowej permutacji wynosi:

$$\mathbf{E} \sim \frac{n^2}{4} \quad (5)$$

Pominęliśmy stałe wynikające z innych operacji niż porównywanie. W average-case będziemy około połowę szybciej niż w worst-case.

*Pseudokod bez przykładu jest słaby.*

### 1.5 Przykład złożoności

Patrzymy na wiodący czynnik.

$$13n^2 + 91n \log n + 4n + 13^{10} = O(n^2) \quad (6)$$

$$= 13n^2 + O(n \log n) \quad (7)$$

Chcielibyśmy gdzie to konieczne, zapisać *lower order terms*.

*Pytanie o dzielenie liczb* - istnieją algorytmy, które ze względu na arytmetyczne właściwości liczb sprawiają, że mniejsze liczby mogą dzielić się dłużej niż większe. Podczas tego kursu nie omawiamy złożoności dla takich algorytmów.

## 2 Lecture II - Merge Sort

### 2.1 Merge sort $(A, 1, n)$

Niech złożoność  $T(n)$  - złożoność algorytmu.

Funkcja merge sort

```

O(1)          | if |A[1...n]| == 1 return A[1...n]
               | else
T(floor(n/2)) |   B = MERGE_SORT(A,1,floor(n/2))
T(ceil(n/2))  |   C = MERGE_SORT(A,floor(n/2)+1, n)
O(n)          |   return MERGE(B,C)

```

Funkcja merge

```

MERGE(X[1...k], Y[1...l])
if k = 0 return Y[1...l]
if l = 0 return X[1...k]
if X[1] <= Y[1]
    return X[1] o MERGE(X[2...k], Y[1...l])
else
    return Y[1] o MERGE(X[1...k], Y[2...l])

```

```

MERGE(A,B)
2 1 ----> [1] + MERGE(A,B (bez 1))
7 9
13 10
19 11
20 14

2 9 ----> [1,2] + MERGE(A (bez 2),B)
7 10
13 11
19 14
20 .

... ----> [1,2,7,9,10,11,13,14]
19 .
20 .

... ----> [1,2,7,9,10,11,13,14,19,20]

```

```

[10], [2], [5], [3], [7], [13], [1], [6]
[2, 10], [3,5], [7,13], [1,6]
[2,3,5,10], [1,6,7,13]
[1,2,3,5,6,7,10,13]

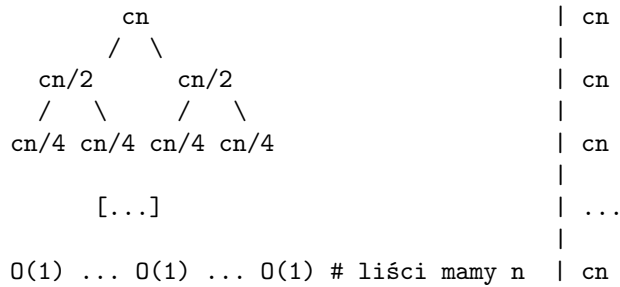
```

Złożoność obliczeniowa merge-a wynosi  $O(k + l)$  - w najgorszym przypadku bierzemy najpierw z jednej strony, potem z drugiej i na zmianę.

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n) \quad (8)$$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n) \quad (9)$$

Rozpiszmy tzw drzewo rekursji:



Musimy dodać wszystkie koszty, które pojawiły się w drzewie. Dodajmy piętra, a następnie zsumujmy. Żeby znać wysokość drzewa interesuje nas dla jakiego  $h$  znajdzie  $\frac{n}{2^h} = 1$

$$\frac{n}{2^h} = 1 \implies 2^h = n \implies h = \log_2 n \quad (10)$$

Zatem złożoność:

$$\sum_{i=1}^{\log n} cn = cn \log n \sim O(n \log n) \quad (11)$$

### 3 Lecture III - Narzędzia do analizy algorytmów

*Dzisiejszy wykład prowadzi GODfryd*

#### 3.1 Notacja asymptotyczna

- Big- $O$  ( $O$ -duże)  $f : \mathbb{N} \rightarrow \mathbb{R}$
- Big- $\Omega$  ( $\Omega$ -duże)  $f : \mathbb{N} \rightarrow \mathbb{R}$
- Big- $\Theta$  ( $\Theta$ -duże)  $f : \mathbb{N} \rightarrow \mathbb{R}$
- Small- $o$  ( $o$ -małe)  $f : \mathbb{N} \rightarrow \mathbb{R}$

#### 3.2 Notacja Big- $O$

**Definition. Notacja Big- $O$ .** Funkcja  $f(n) \in O(g(n))$ , gdy:

$$f(n) = O(g(n)) \equiv (\exists c > 0) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) (|f(n)| \leq c \cdot |g(n)|)$$

Przykład:  $2n^2 = O(n^3)$ , dla  $n_0 = 2, c = 1$  definicja jest spełniona.

*Pomijamy tutaj stałe - interesuje nas rząd wielkości*

$$O(g(n)) = \{f \in \mathbb{N}^{\mathbb{R}} : f \text{ spełnia definicję}\}$$

$O(g(n))$  jest klasą funkcji, ale jako informatycy możemy zapisywać  $f = O(g)$ , zamiast  $f \in O(g)$ . Notacja nie ma symetrii, to znaczy  $f = O(g) \nrightarrow g = O(f)$

**Fact. Definicja Big-O za pomocą granicy.** Możemy zapisać alternatywnie:

$$f(n) = O(g(n)) \equiv \limsup_{n \rightarrow \infty} \left| \frac{f(n)}{g(n)} \right| \leq \infty$$

Uwaga. Jeśli  $\lim_{n \rightarrow \infty} \left| \frac{f(n)}{g(n)} \right| < \infty$  (istnieje), to:

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

Przykłady:  $\begin{cases} f(n) = n^2 \\ g(n) = (-1)^n n^2 \end{cases}$

Granica nie istnieje, ale  $\limsup = 1$

$$\left\{ \frac{f(n)}{g(n)} \right\} = \begin{cases} 1, & 2 \mid n \\ \frac{1}{n}, & 2 \nmid n \end{cases}$$

Granica nie istnieje.

**Fact. Dokładność zapisu Big-O.** Pomijamy składniki niższego rzędu jako mniej istotne, ale podkreślamy że istnieją:

$$f(n) = n^3 + O(n^2) \equiv (\exists h(n) = O(n^2)) (f(n) = n^3 + h(n)) \quad (12)$$

Rozważmy następnie stwierdzenie:

$$n^2 + O(n) = O(n^2) \equiv (\forall f(n) = O(n)) (\exists h(n) = O(n^2)) (n^2 + f(n) = h(n)) \quad (13)$$

Rozumiemy to następująco - dodając dowolną funkcję z klasy funkcji liniowych do  $n^2$  otrzymamy funkcję z klasy funkcji kwadratowych.

### 3.3 Notacja Big-Ω

**Definition. Notacja Big-Ω.** Funkcja  $f(n) \in \Omega(g(n))$ , gdy:

$$f(n) = \Omega(g(n)) \equiv (\exists c > 0) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) (|f(n)| \geq c \cdot |g(n)|) \quad (14)$$

biorąc  $c' = \frac{1}{c} > 0$  mamy:  $(|g(n)| \leq c' \cdot |f(n)|)$ , czyli  $g(n) = O(f(n))$ .

Przykład:

$$2n^2 = O(n^3) \quad (15)$$

$$n^3 = \Omega(2n^2) \quad (16)$$

$$n = \Omega(\log n) \quad (17)$$

*Każda funkcja jest Omega od siebie samej.*

### 3.4 Notacja Big-Θ

**Definition. Notacja Big-Θ.** Funkcja  $f(n) \in \Theta(g(n))$ , gdy:

$$f(n) = \Theta(g(n)) \equiv (\exists c_1, c_2 > 0) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) (c_1 \cdot |g(n)| \leq |f(n)| \leq c_2 \cdot |g(n)|) \quad (18)$$

Przykład:

$$n^2 = \Theta(2n^2) \quad (19)$$

$$n^3 = \Theta(n^3) \quad (20)$$

$$n^4 + 3n^2 + \log n = \Theta(n^4) \quad (21)$$

**Fact. Dokładność zapisu Theta.**

$$f(n) = \Theta(g(n)) \equiv f(n) = O(g(n)) \wedge f(n) = \Omega(g(n)) \quad (22)$$

$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n)) \quad (23)$$

Rozważmy przypadek patologiczny

$$f(n) = n^{1+\sin \frac{\pi \cdot n}{2}} \quad g(n) = n \quad (24)$$

$$f \neq O(g), g \neq O(f) \quad (25)$$

### 3.5 Notacja small- $o$

**Definition. Notacja small- $o$ .** Funkcja  $f(n) \in o(g(n))$ , gdy:

$$f(n) = o(g(n)) \equiv (\forall c > 0) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) (|f(n)| < c \cdot |g(n)|) \quad (26)$$

Równoważnie:

$$f(n) = o(g(n)) \equiv \lim_{n \rightarrow \infty} \left| \frac{f(n)}{g(n)} \right| = 0 \quad (27)$$

Przykład:

$$n = o(n^2) \quad (28)$$

$$n^2 = o(n^3) \quad (29)$$

$$n^3 = o(2^n) \quad (30)$$

### 3.6 Notacja small- $\omega$

**Definition. Notacja small- $\omega$ .** Funkcja  $f(n) \in \omega(g(n))$ , gdy:

$$f(n) = \omega(g(n)) \equiv (\forall c > 0) (\exists n_0 \in \mathbb{N}) (\forall n \geq n_0) (|f(n)| > c \cdot |g(n)|) \quad (31)$$

Równoważnie:

$$f(n) = \omega(g(n)) \equiv \lim_{n \rightarrow \infty} \left| \frac{f(n)}{g(n)} \right| = \infty \quad (32)$$

Przykład:

$$3.14n^2 + n = O(n^3) = \omega(n) \quad (33)$$

### 3.7 Metody rozwiązywania rekurencji

- Metoda podstawienia (indukcji) - Cormen
- Metoda drzewa rekursji
- Metoda master theorem



### 3.8 Rozwiązywanie rekurencji

1. Zgadnij odpowiedź (wiodący składnik)
2. Sprawdź przez indukcję, czy dobrze zgadliśmy
3. Wylicz stałe

**Information. Historyjka.** Dwóch przyjaciół zgubiło się podczas podróży balonem.

- "Gdzie jesteście?"
- "W balonie."

Osoba, którą spotkali, była matematykiem.

Odpowiedź była precyzyjna, dokładna i całkowicie bezużyteczna.

### 3.9 Metoda podstawiania - Metoda dowodu indukcyjnego

Przykład 1. Rozwiążmy równanie rekurencyjne:

$$T(n) = 4T\left(\frac{n}{2}\right) + n \quad T(1) = \Theta(1) \quad (34)$$

Założmy, że  $T(n) = O(n^3)$  - pokazać, że  $T(n) \leq c \cdot n^3$  dla dużych  $n$ .

1. Krok początkowy  $T(1) = \Theta(1) \leq c \cdot 1^3 = c$  ok.
2. Założmy, że  $\forall_{k < n} T(k) \leq c \cdot k^3$  (zał. indukcyjne, nie  $\Theta(k^3)$  - chcemy konkretną stałą  $c$ )
3.  $T(n) = 4T\left(\frac{n}{2}\right) + n \leq 4c\left(\frac{n}{2}\right)^3 + n = \frac{1}{2}cn^3 + n = cn^3 - \frac{1}{2}cn^3 + n \leq cn^3$ .
4. Wystarczy wskazać  $c$ , takie że  $\frac{1}{2}cn^3 - n \geq 0$ , np  $c \geq 2$
5. Pokazaliśmy, że  $T(n) = O(n^3)$

Założmy, że  $T(n) = O(n^2)$  - pokazać, że  $T(n) \leq c \cdot n^2$  dla dużych  $n$ .

1. Krok początkowy  $T(1) = \Theta(1) \leq c \cdot 1^2 = c$  ok.
2. Założmy, że  $\forall_{k < n} T(k) \leq c \cdot k^2$  (zał. indukcyjne)
3.  $T(n) = 4T\left(\frac{n}{2}\right) + n \leq 4c\left(\frac{n}{2}\right)^2 + n = cn^2 + n = cn^2 - cn^2 + n \leq cn^2$ .
4. Tego się nie da pokazać - nie jest prawdą, że  $T(n) = O(n^2)$

Wzmocnijmy zatem założenie indukcyjne:

1.  $T(n) \leq c_1 n^2 - c_2 n$  (zał. indukcyjne)
2.  $T(n) = 4T\left(\frac{n}{2}\right) + n \leq 4(c_1 \frac{n^2}{2} - c_2 \frac{n}{2}) + n$
3.  $= c_1 n^2 - 2c_2 n + n = c_1 n^2 - (2c_2 - 1)n \leq$
4.  $\leq c_1 n^2 - c_2 n$
5. Weźmy  $c_1 = 1, c_2 = 2$ , wtedy  $T(n) \leq n^2 - 2n = O(n^2)$

Przykład 2. Weźmy paskudną rekursję  $T(n) = 2T(\sqrt{n}) + \log n$ .  
Założmy, że  $n$  jest potęgą 2 oraz oznaczmy  $n = 2^m, m = \log_2 n$ .

$$T(2^m) = 2T((2^m)^{\frac{1}{2}}) + m \quad (35)$$

Oznaczmy  $T(2^m) = S(m)$ . Wtedy:

$$S(m) = 2S\left(\frac{m}{2}\right) + m \quad (36)$$

(dobrze znana rekurencja -  $S(n) = O(m \log m)$ ) - patrz Lecture 2. Przejdźmy z powrotem na  $T, n$ :

$$T(2^m) = S(m)T(2^m) = O(m \log m)T(n) = O(\log n \log \log n) \quad (37)$$

Formalnie pokazaliśmy to tylko dla potęg 2 - musielibyśmy jeszcze indukcyjnie to udowodnić.

*Kiedy podłogi i sufitu mają znaczenie?*

## 4 Lecture IV - Metoda drzewa rekursji

### 4.1 Metoda drzewa rekursji

W danym węźle wstawiamy koszt operacji. Sumujemy koszty węzłów na danym poziomie.

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + n^2, \quad T(1) = \Theta(1) \quad (38)$$

Chcemy sumować koszty na danym poziomie, a potem napisać pełną sumę.

$$\begin{array}{rcl} & n^2 & | \quad n^2 \\ & / \quad \backslash & \\ (n/2)^2 & & (n/4)^2 \quad | \quad 5/16 \quad n^2 \\ / \quad \backslash & & / \quad \backslash \\ (n/4)^2 \quad (n/8)^2 & (n/8)^2 \quad (n/16)^2 & | \quad 25/256 \quad n^2 = (5/16)^k \quad n^2 \\ \dots & & \end{array}$$

$$T^*(n) = \sum_{k=0}^{\infty} \left(\frac{5}{16}\right)^k n^2 = \quad (39)$$

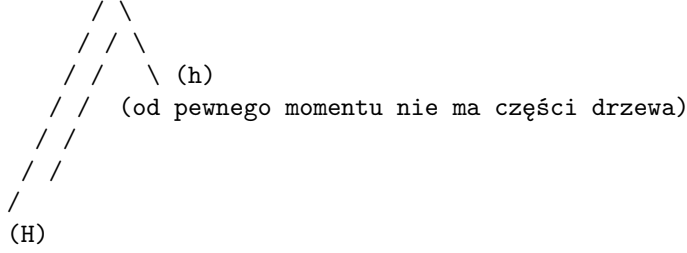
$$= n^2 \sum_{k=0}^{\infty} \left(\frac{5}{16}\right)^k = \quad (40)$$

$$= n^2 \cdot \left(\frac{1}{1 - \frac{5}{16}}\right) = \quad (41)$$

$$= \frac{16}{11} n^2 \quad (42)$$

Nie mogłoby być mniej niż  $n^2$ , bo już w pierwszym rzędzie jest  $n^2$ .  
Nie jest to dokładne, ale dostaliśmy górne ograniczenie.

$$T(n) = O(n^2) \quad (43)$$



Wysokości różnią się o stałą:

$$\frac{n}{2^H} = 1 \implies H = \log_2 n \quad (44)$$

$$\frac{n}{4^h} = 1 \implies h = \log_4 n \quad (45)$$

*Za chwilę będę dodawał rzeczy, które nie istnieją*

Pamiętajmy, że:

$$a^{\log_b n} = n^{\log_b a}$$

$$\hat{T}(n) = \sum_{k=0}^{H=\log_2(n)} \left(\frac{5}{16}\right)^k n^2 = \quad (46)$$

$$= n^2 \sum_{k=0}^H \left(\frac{5}{16}\right)^k = \quad (47)$$

$$= n^2 \cdot \frac{1}{11} \left(16 - 5 \left(\frac{5}{16}\right)^{\log_2 n}\right) = \quad (48)$$

$$= \frac{16}{11} n^2 - \frac{5}{11} n^{2-1.67} \quad (49)$$

Rozważmy ograniczenie dolne:

$$\check{T}(n) = \sum_{k=0}^{h=\log_4(n)} \left(\frac{5}{16}\right)^k n^2 = n^2 \frac{1}{11} \left(16 - C \cdot \left(\frac{5}{16}\right)^{\log_4 n}\right) \quad (50)$$

Zatem wiemy, że:

$$T(n) = O(\hat{T}(n)) = O(T^*(n)) \quad (51)$$

$$T(n) = \Omega(\check{T}(n)) \quad (52)$$

$$T(n) = \Theta(n^2) = \frac{16}{11}n^2 + o(n^2) \quad (53)$$

## 4.2 Metoda iteracyjna

$$T(n) = 3T\left(\frac{n}{4}\right) + n = \quad (54)$$

$$T(n) = 3\left(3T\left(\frac{n}{16}\right) + \frac{n}{4}\right) + n = 9T\left(\frac{n}{16}\right) + \frac{3}{4}n + n = \quad (55)$$

$$T(n) = n + \frac{3}{4}n + 9\left(3T\left(\frac{n}{64}\right) + \frac{n}{16}\right) = \quad (56)$$

$$T(n) = n + \frac{3}{4}n + \frac{9}{16}n + 27T\left(\frac{n}{64}\right) = \quad (57)$$

$$T(n) = n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \left(\frac{3}{4}\right)^3 n + \dots + 3^j T\left(\frac{n}{4^j}\right) = \quad (58)$$

$$(59)$$

Wyznaczmy koniec iteracji:

$$\frac{n}{4^j} = 1 \implies j = \log_4 n \quad (60)$$

To jest nic innego jak:

$$\sum_{j=0}^{\log_4 n} \left(\frac{3}{4}\right)^j = O(n) \quad (61)$$

## 4.3 Master Theorem

**Theorem. Master Theorem.** Jeśli  $T(n) = a \cdot T(\lceil \frac{n}{b} \rceil) + \Theta(n^d)$  dla pewnych stałych  $a > 0, b > 1, d > 0$ , oraz  $T(1) = \Theta(1)$  to:

$$T(n) = \begin{cases} \Theta(n^d) & \text{jeśli } d > \log_b a \\ \Theta(n^d \log n) & \text{jeśli } d = \log_b a \\ \Theta(n^{\log_b a}) & \text{jeśli } d < \log_b a \end{cases}$$

$$\hat{T}(n) = a \cdot \hat{T}\left(\frac{n}{b} + 1\right) + \Theta(n^d) \quad (62)$$

$$\check{T}(n) = a \cdot \check{T}\left(\frac{n}{b}\right) \quad (63)$$

Dowód

wielkość	.	liczba podproblemów
$n$	$c \cdot n^d$	1
$n/b$	$c \cdot (n/b)^d$	$a$
$n/b^2$	$c \cdot (n/(b^2))^d$	$a^2$

...

koszt na poziomie ' $k$ ' =  $c \cdot (n/b^k)^d$

liczba podproblemów na poziomie ' $k$ ' =  $a^k$

suma kosztów ' $k$ '-tym wierszu =  $c \cdot (a/b^d)^k \cdot n^d$

Wysokość drzewa rekursji

$$\frac{n}{b^h} = 1 \implies h = \log_b n \quad (64)$$

Zatem:

$$T(n) = \Theta \left( \sum_{k=0}^{\log_b n} \cdot \left( \frac{a}{b^d} \right)^k n^d \right) \quad (65)$$

*Mogę wziąć  $\Theta$  zamiast  $o$ , bo dość dokładnie robię - ale trochę nie*

$$\sum_{k=0}^h q^k = \frac{1 - q^{h+1}}{1 - q} \quad \sum_{h=0}^h 1^k = (h + 1)$$

$$T(n) = \Theta \left( n^d \sum_{k=0}^{\log_b n} \cdot \left( \frac{a}{b^d} \right)^k \right) \quad (66)$$

(1) Jeśli  $\frac{a}{b^d} < 1$ , to:

$$a < b^d \quad (67)$$

$$\log_b(a) < d \quad \text{zatem} \quad (68)$$

$$T(n) = \Theta(n^d) \quad (69)$$

(większość pracy dzieje się z korzenia - okolic korzenia)

(2) Jeśli  $\frac{a}{b^d} = 1$ , to:

$$a = b^d \quad (70)$$

$$\log_b(a) = d \quad (71)$$

$$T(n) = \Theta(n^d \log n) \quad (72)$$

(suma kosztów w  $k$ -tym wierszu - każdy wiersz kontrybuuje równie mocno)

(3) Jeśli  $\frac{a}{b^d} > 1$ , to:

$$a > b^d \quad (73)$$

$$\log_b(a) > d \quad (74)$$

$$T(n) = \Theta(n^{\log_b a}) \quad (75)$$

(z każdym kolejnym poziomem koszt rośnie - większość złożoności kryje się na dole drzewa rekursji)

*Z tego co dzieje się na początku... albo na końcu, bo to może być scalanie  
Stworzyliście za dużo podproblemów.*

Co jeśli rekurencja nie ma  $n^d$ , a ma  $n \log(n)$ ? - możemy przybliżać

Przykład

$$T(n) = 4T\left(\frac{n}{2}\right) + 11n \quad a = 4, b = 2, d = 1 \quad (76)$$

$$\log_b a = \log_2 4 = 2 > 1 = d \quad \text{to jest przypadek (3)} \quad (77)$$

$$T(n) = \Theta(n^{\log_a b}) = \Theta(n^{\log_2 4}) = \Theta(n^2) \quad (78)$$

Przykład

$$T(n) = 4T\left(\frac{n}{3}\right) + 3n^2 \quad a = 4, b = 3, d = 2 \quad (79)$$

$$\log_b a = \log_3 4 > 2 = d \quad \text{to jest przypadek (1)} \quad (80)$$

$$T(n) = \Theta(n^d) = \Theta(n^2) \quad (81)$$

Przykład

$$T(n) = 27T\left(\frac{n}{3}\right) + 0.3n^3 \quad a = 27, b = 3, d = 3 \quad (82)$$

$$\log_b a = \log_3 27 = 3 = d \quad \text{to jest przypadek (2)} \quad (83)$$

$$T(n) = \Theta(n^d \log n) = \Theta(n^3 \log n) \quad (84)$$

## 4.4 Divide and Conquer

1. Podział problemu na mniejsze podproblemy.
2. Rozwiąż rekurencyjnie mniejsze (rozłączne) podproblemy.
3. Połącz rozwiązania problemów w celu rozwiązania problemu wejściowego.

#### 4.5 Wyszukiwanie elementów w posortowanej tablicy

- Input - posortowana tablica  $A[1..n]$ , element  $x$
- Output - indeks  $i$  taki, że  $A[i] = x$  lub błąd, gdy  $x$  nie występuje w  $A$

#### 4.6 Binary search

1. if  $n = 1, A[n] = x$  return  $n$ , else  $A$  does not contain  $x$
2. porównujemy  $x$  z  $A[\frac{n}{2}]$
3. jeśli  $x = A[\frac{n}{2}]$  return  $\frac{n}{2}$
4. jeśli  $x < A[\frac{n}{2}]$ , BinarySearch( $A[1..\frac{n}{2} - 1], x$ )
5. jeśli  $x > A[\frac{n}{2}]$ , BinarySearch( $A[\frac{n}{2} + 1..n], x$ )

*Wy nie patrzcie na pseudokody na tablicy, tylko w książce*

$$T(n) = 1T\left(\frac{n}{2}\right) + \Theta(1) \tag{85}$$

$$T(n) = \Theta(\log n) \tag{86}$$