AiSD

Rafał Włodarczyk

INA 4, 2025

Contents

I welcome you on the path to insanity.

Good luck :)

1 Lecture I - Sortowanie

Definiujemy problem:

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

1.1 Worst-case analysis

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

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.

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

Później możemy badać wariancję, oraz koncentrację.

1.3 Analiza losowego sortowania

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

1.4 Insertion Sort (A, n)

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

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

- $8_i, 2_j, 4, 9, 3, 6 \quad j = 2, i = 1, key = 2$ while
- $2, 8_j, 4, 9, 3, 6$
- $2, 8_i, 4_j, 9, 3, 6 \quad j = 3, i = 2, key = 4$ while

- $2, 4, 8, 9, 3, 6$
- $2, 4, 8_i, 9_j, 3, 6 \quad j = 4, i = 3, key = 9$ no while
- $2, 4, 8, 9_i, 3_j, 6 \quad j = 5, i = 4, key = 3$ while
- $2, 3, 4, 8, 9, 6$
- $2, 3, 4, 8, 9_i, 6_j \quad j = 6, i = 5, key = 6$ 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 ustaliśmy że liczba operacji w while zależy od j, wtedy:

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

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

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:

$$\mathbf{E} \sim \frac{n^2}{4} \tag{5}$$

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

Pseudokod bez przykładu jest słaby.

1.5 Przykład złożoności

Patrzymy na wiodący czynnik.

$$13n^2 + 91n\log n + 4n + 13^{10} = O(n^2) \tag{6}$$
$$= 13n^2 + O(n\log n) \tag{7}$$

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) - złożność 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 .
```

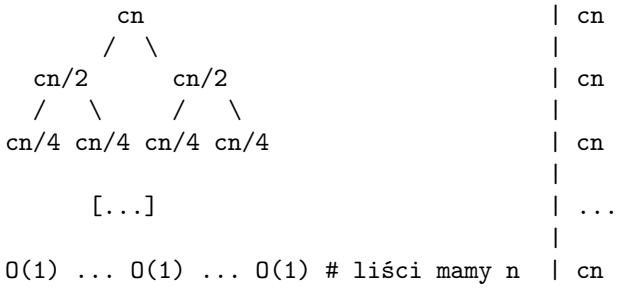
```
... ---> [1,2,7,9,10,11,13,14]
19 .
20 .

... ---> [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.

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

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

Rozpiszmy tzw drzewo rekursji:



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

$$\frac{n}{2^h} = 1 \implies 2^h = n \implies h = \log_2 n \tag{10}$$

Zatem złożność:

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

3 Lecture III - Narzędzia do analizy algorytmów

Dzisiejszy wykład prowadzi GODfryd

3.1 Notacja asymptotyczna

- Big-O (O-duże) $f : \mathbb{N} \to \mathbb{R}$
- Big-Ω (Ω-duże) $f : \mathbb{N} \to \mathbb{R}$
- Big-Θ (Θ-duże) $f : \mathbb{N} \to \mathbb{R}$
- Small-o (o-małe) $f : \mathbb{N} \to \mathbb{R}$

3.2 Notacja Big-O

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

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

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

Pomijamy tutaj stałe - interesuje nas rząd wielkości

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

$O(g(n))$ 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:

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

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

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

Przykłady: $\begin{cases} f(n) = n^2 \\ g(n) = (-1)^n n^2 \end{cases}$
Granica nie istnieje, ale $\limsup = 1$
$\left\{\frac{f(n)}{g(n)} = \begin{cases} 1, & 2 \mid n \\ \frac{1}{n}, & 2 \nmid n \end{cases}\right.$
Granica nie istnieje.

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

$$f(n) = n^3 + O(n^2) \equiv \left(\exists h(n) = O(n^2)\right)\left(f(n) = n^3 + h(n)\right) \tag{12}$$

Rozważmy następnie stwierdzenie:

$$n^2 + O(n) = O(n^2) \equiv (\forall f(n) = O(n))\left(\exists h(n) = O(n^2)\right)\left(n^2 + f(n) = h(n)\right) \tag{13}$$

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:

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

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

$$2n^2 = O(n^3) \tag{15}$$
$$n^3 = \Omega(2n^2) \tag{16}$$
$$n = \Omega(\log n) \tag{17}$$

Każda funkcja jest Omega od siebie samej.

3.4 Notacja Big-Θ

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

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

Przykład:

$$n^2 = \Theta(2n^2) \tag{19}$$
$$n^3 = \Theta(n^3) \tag{20}$$
$$n^4 + 3n^2 + \log n = \Theta(n^4) \tag{21}$$

Fact. Dokładność zapisu Theta.

$$f(n) = \Theta(g(n)) \equiv f(n) = O(g(n)) \wedge f(n) = \Omega(g(n)) \tag{22}$$
$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n)) \tag{23}$$

Rozważmy przypadek patologiczny

$$f(n) = n^{1+\sin \frac{\pi \cdot n}{2}} \quad g(n) = n \tag{24}$$
$$f \neq O(g), g \neq O(f) \tag{25}$$

3.5 Notacja small-o

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

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

Równoważnie:

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

Przykład:

$$n = o(n^2) \tag{28}$$
$$n^2 = o(n^3) \tag{29}$$
$$n^3 = o(2^n) \tag{30}$$

3.6 Notacja small-ω

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

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

Równoważnie:

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

Przykład:

$$3.14n^2 + n = O(n^3) = \omega(n) \tag{33}$$

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 jesteśmy?"
- "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:

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

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

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

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

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

Wzmocnijmy zatem założenie indukcyjne:

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

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

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

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

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

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

$$T(2^m) = S(m)T(2^m) = O(m \log m)T(n) = O(\log n \log \log n) \tag{37}$$

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

Kiedy podłogi i sufity 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.

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

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

```
             n^2                  | n^2
         /         \
   (n/2)^2          (n/4)^2        | 5/16 n^2
  /      \          /     \
(n/4)^2 (n/8)^2 (n/8)^2 (n/16)^2   | 25/256 n^2 = (5/16)^k n^2
```

...

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

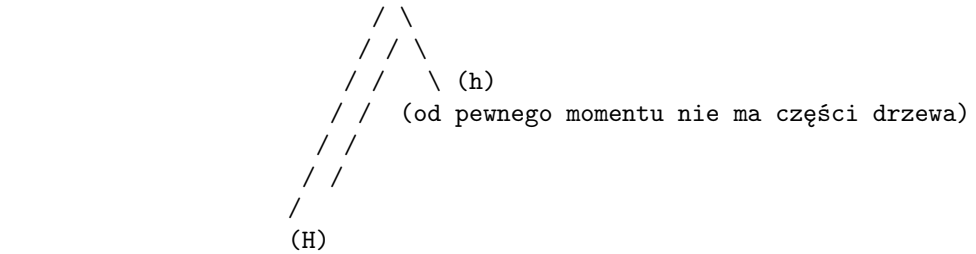
$$= n^2 \sum_{k=0}^{\infty} \left(\frac{5}{16}\right)^k = \tag{40}$$

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

$$= \frac{16}{11} n^2 \tag{42}$$

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

$$T(n) = O(n^2) \tag{43}$$



Wysokości różnią się o stałą:

$$\frac{n}{2^H} = 1 \implies H = \log_2 n \tag{44}$$

$$\frac{n}{4^h} = 1 \implies h = \log_4 n \tag{45}$$

Za chwilę będę dodawał rzeczy, które nie istnieją

Pamiętajmy, że:

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

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

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

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

$$= \frac{16}{11}n^2 - \frac{5}{11}n^{2-1.67} \tag{49}$$

Rozważmy ograniczenie dolne:

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

Zatem wiemy, że:

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

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

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

4.2 Metoda iteracyjna

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

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

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

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

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

$$\tag{59}$$

Wyznaczmy koniec iteracji:

$$\frac{n}{4^j} = 1 \implies j = \log_4 n \tag{60}$$

To jest nic innego jak:

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

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:

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

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

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

Dowód

```
wielkość                .                   liczba podproblemów
n                       c n^d               1
n/b                     c (n/b)^d           a
n/b^2                   c (n/(b^2))^d       a^2

...

koszt na poziomie 'k' = c (n/b^k)^d
liczba podproblemów na poziomie 'k' = a^k

suma kosztów 'k'-tym wierszu = c (a/b^d)^k * n^d
```

Wysokość drzewa rekursji

$$\frac{n}{b^h} = 1 \implies h = \log_b n \tag{64}$$

Zatem:

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

Mogę wziąć thetę zamiast o, bo dość dokładnie robię - ale trochę nie

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

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

(1) Jeśli $\frac{a}{b^d} < 1$, to:

$$a < b^d \tag{67}$$

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

$$T(n) = \Theta(n^d) \tag{69}$$

(większość pracy dzieje się z korzenia - okolic korzenia)

(2) Jeśli $\frac{a}{b^d} = 1$, to:

$$a = b^d \tag{70}$$

$$\log_b(a) = d \tag{71}$$

$$T(n) = \Theta(n^d \log n) \tag{72}$$

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

(3) Jeśli $\frac{a}{b^d} > 1$, to:

$$a > b^d \tag{73}$$
$$\log_b(a) > d \tag{74}$$
$$T(n) = \Theta(n^{\log_b a}) \tag{75}$$

(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

$$T(n) = 4T\left(\frac{n}{2}\right) + 11n \quad a = 4, b = 2, d = 1 \tag{76}$$
$$\log_b a = \log_2 4 = 2 > 1 = d \quad \text{to jest przypadek (3)} \tag{77}$$
$$T(n) = \Theta\left(n^{\log_a b}\right) = \Theta\left(n^{\log_2 4}\right) = \Theta\left(n^2\right) \tag{78}$$

Przykład

$$T(n) = 4T\left(\frac{n}{3}\right) + 3n^2 \quad a = 4, b = 3, d = 2 \tag{79}$$
$$\log_b a = \log_3 4 > 2 = d \quad \text{to jest przypadek (1)} \tag{80}$$
$$T(n) = \Theta\left(n^d\right) = \Theta\left(n^2\right) \tag{81}$$

Przykład

$$T(n) = 27T\left(\frac{n}{3}\right) + 0.(3)n^3 \quad a = 27, b = 3, d = 3 \tag{82}$$
$$\log_b a = \log_3 27 = 3 = d \quad \text{to jest przypadek (2)} \tag{83}$$
$$T(n) = \Theta\left(n^d \log n\right) = \Theta\left(n^3 \log n\right) \tag{84}$$

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 portowanej 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

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

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

5 Lecture V - Divide and Conquer

5.1 Potęgowanie liczby

- Input - liczba x, liczba całkowita n
- Output - x^n

Bazowo zachodzi $n - 1$ mnożeń x przez siebie. (czyli $\Theta(n)$ operacji)

$$x \cdot x \cdot \cdots \cdot x = x^n \tag{87}$$

Zróbmy to sprytniej:

$$x^n = \begin{cases} x^{\frac{n}{2}} \cdot x^{\frac{n}{2}} & \text{dla parzystego} \quad n \\ x^{\frac{n-1}{2}} \cdot x^{\frac{n-1}{2}} \cdot x & \text{dla nieparzystego} \quad n \end{cases} \tag{88}$$

Z liniowej liczby mnożeń zeszliśmy do logarytmicznej liczby mnożeń.

$$T(n) = 1 \cdot T\left(\frac{n}{2}\right) + \Theta(1) \tag{89}$$

$$T(n) = \Theta(\log n) \tag{90}$$

5.2 Wyliczenie n-tej liczby Fibonacciego

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2), & n > 1 \end{cases} \tag{91}$$

Normalne wywołanie funkcji to $\Theta(\varphi^n)$

Wykorzystajmy podejście bottom-up, liczymy i zapamiętujmy każdorazowo $F_2, F_3, \ldots, F_n$
Osiągnęliśmy złożoność liniową $\Theta(n)$

Istnieje jednak zwarty wzór na $F(n) = \frac{1}{\sqrt{5}} \left(\frac{\varphi^n + \varphi^n}{2} \right)$ a to możemy policzyć logarytmicznie.

Tu pojawiają się liczby - jak one się nazywały - (z sali) niewymierne.

Istnieje macierz, która mnożona pozwala na policzenie n-tej liczby Fibonacciego.

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \tag{92}$$

Algorytm używający tego wzoru - połączony z szybkim potęgowaniem, ma złożoność $\Theta(\log n)$.

5.3 Mnożenie Liczb

- Input: x, y (liczby n-bitowe)
- Output: $x \cdot y$

Standardowe mnożenie w słupku to $\Theta(n^2)$ mnożeń i $\Theta(n)$ dodawań.
Załóżmy, że n jest parzyste:

$$x = x_L \cdot 2^{\frac{n}{2}} + x_R \tag{93}$$
$$y = y_L \cdot 2^{\frac{n}{2}} + y_R \tag{94}$$
$$x \cdot y = \left(x_L \cdot 2^{\frac{n}{2}} + x_R \right) \cdot \left(y_L \cdot 2^{\frac{n}{2}} + y_R \right) = \tag{95}$$
$$= x_L \cdot y_L \cdot 2^n + (x_L y_R + x_R y_L) \cdot 2^{\frac{n}{2}} + x_R y_R \tag{96}$$

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n) \tag{97}$$
$$a = 4, b = 2, d = 1 \tag{98}$$
$$\log_b a = \log_2 4 = 2 > 1 = d \tag{99}$$
$$T(n) = \Theta(n^2) \tag{100}$$

Asymptotycznie nie zyskaliśmy nic.
Ten przypadek pokazuje, że czasami nie wystarczy bezmyślnie podzielić a potem scalić.

A co o tym myślał Gauss - tu jest dużo mnożeń - cztery.

$$(a+ib)(c+id) = ac - bd + i(bc+ad) \tag{101}$$
$$bc + ad = (a+b)(c+d) - ac - bd \tag{102}$$

Zobaczmy, że ac, bd są już policzone wyżej - zamiast 4 mnożeń, mamy 3 mnożenia.

$$x \cdot y = x_L y_L 2^n + ((x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R) + x_R y_R \tag{103}$$

Wykonujemy i zapamiętujemy mnożenia $x_L y_L, x_R y_R, (x_L + x_R)(y_L + y_R)$ - zamiast 4 mnożeń, mamy 3 mnożenia.

$\Theta(n)$ - wynika z przeunięć bitowych oraz dodawań.

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) \tag{104}$$
$$a = 3, b = 2, d = 1 \tag{105}$$
$$\log_b a = \log_2 3 > 1 = d \tag{106}$$
$$T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.59}) \tag{107}$$

Najszybszy znany algorytm - na podstawie szybkiej transformaty fouriera $\sim O(n \cdot \log n \cdot \log\log n)$

```
mutiply(x, y)
    n = max {|x|, |y|}
    if n == 1 return x * y
    x_L, x_R = leftmost(ceil(n/2),x), rightmost(floor(n/2),x)
    y_L, y_R = leftmost(ceil(n/2),y), rightmost(floor(n/2),y)

    p1 = multiply(x_L, y_L)
    p2 = multiply(x_R, y_R)
    p3 = multiply(x_L + x_R, y_L + y_R)

    return p1 << n + (p3 - p1 - p2) << ceil(n/2) + p2
```

Podobnie możemy mnożyć macierze.

5.4 Mnożenie macierzy

- Input: A, B - n-wymiarowe macierze
- Output: $A \cdot B$

Naiwne mnożenie macierzy wykonuje $\Theta(n^3)$ mnożeń.

Podzielmy macierz na 4 równe częsci:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix} \tag{108}$$

$$T(n) = 8T\left(\frac{n}{2}\right) + O(n^2) \tag{109}$$

$$T(n) = O(n^3) \tag{110}$$

Znowu nic nie zyskaliśmy. Jesteśmy w stanie wyeliminować jedno mnożenie - osiągając ostatecznie $\Theta(n^{\log_2 7}) \sim \Theta(n^{2.81})$.

Algorytmy state of the art - $\Theta(n^2 \text{polylog}(n))$.

5.5 Quick Sort

Algorytm na podział - scalanie już posortowanych. Pozwala na sortowanie w miejscu.

```
A[1..n]
|       |------|      |
1       p      q      n

A[1..n]
|       |   <=  |     | <    |      |
1       p       |pivot|      q      n
```

1. Podziel $A[p..q]$ na dwie tablice: $A[p..k-1]$, *pivot*, $A[k+1..q]$ takie, że:

$$\forall_{i \in [p..k-1]} A[i] \leqslant pivot, \forall_{j \in [k+1..q]} A[j] > pivot$$

2. Quicksort$(A, p, k-1)$
   Quicksort$(A, l-1, q)$

Przykład - weźmy nieposortowaną tablicę:

```
Quicksort(A,1,n)
[6, 1, 4, 3, 5, 7, 2, 8] # pivot = 6
->
[1, 4, 3, 5, 2, 6, 7, 8]
                .
Quicksort(A,1,5)
Quicksort(A,7,8) ->
[1, 4, 3, 2, 5, 6, 7, 8] # pivot = 1
 .              .  .

Quicksort(A,2,5) ->
[1, 3, 2, 4, 5, 6, 7, 8] # pivot = 4
 .        .     .  .

Quicksort(A,2,3) ->
[1, 2, 3, 4, 5, 6, 7, 8] # pivot = 3
 .  .  .  .  .  .  .  .
```

6 Lecture VI - Quicksort

Rozważmy algorytmy służące do dzielenia tablicy w Quicksorcie

6.1 Lomuto Partition

```
Lomato Partition(A, p, q) # A[p..q]
    pivot = A[p]
    i = p
    for j = p + 1 to q
        if A[j] <= pivot # expensive |A[p..q]| = n, then (n-1) comparisons ~ Theta(n)
            i = i + 1
            swap (A[i], [j]) # expensive, but if dependent
    swap (A[i], A[p]) # pivot in between A[p..i] and A[i+1..q]
    return i

A
|*| <= pivot |i| pivot < |j| ? |
p                             q

We either put the ? element in the '<= pivot' part, or '> pivot' part

A
| <= pivot | * | pivot < |
p                     q

Example
6, 10, 13, 5, 8, 3, 2, 11
*  i       j

swap(5,10)

6, 5, 13, 10, 8, 3, 2, 11
*  i          j

do nothing

6, 5, 13, 10, 8, 3, 2, 11
*     i          j

swap(3, 13)

6, 5, 3, 10, 8, 13, 2, 11
*     i          j

6, 5, 3, 2, 8, 13, 10, 11
*        i         j

6, 5, 3, 2, 8, 13, 10, 11
```

```
*        i              j

swap(6, 2)

2, 5, 3, 6, 8, 13, 10, 11
*        i              j

return i = 3
```

Biorąc pod uwagę, że dokonujemy $n - 1$ porównań, złożoność Lomuto Partition wynosi $\Theta(n)$.

6.2 Hoare Partition

```
Hoare Partition(A, p, q) # A[p..q]
    pivot = A[floor((p+q)/2)]
    i = p - 1
    j = q + 1
    while True
        do
            i++
        while A[i] < pivot

        do
            j--
        while A[j] > pivot

        if i >= j return j
        swap(A[i], A[j])

* - pivot

Example
  6, 10, 13, 5, 8, 3, 2, 11
i p          *           q  j
  i          *        j         # swap(6, 2)

  2, 10, 13, 5, 8, 3, 6, 11
     i       *     j            # swap(10, 3)

  2, 3, 13, 5, 8, 10, 6, 11
            *

  2, 3, 13, 5, 8, 10, 6, 11
        i   j                   # swap (13, 5)

  2, 3, 5, 13, 8, 10, 6, 11
        *
```

```
        j  i

A
| <= pivot | < pivot |
p          j         q

return j
```

W Hoare Partition tracimy pivot który może ulec przesunięciu. Porównań robimy więcej o stałą $n \pm c, c = 1$. Złożoność $\Theta(n)$ - zdecydowanie mniej swapów, 2-3 razy mniej niż Lomuto partition.

```
QS(A,p,q)
    if p < q
        r = Partition(A,p,q)
        QS(A,p,r-1)
        QS(A,r+1,q)
```

6.3 Worst Case Analysis for QS

Najgorzej będzie jak każdorazowo będziemy nierówno dzielić po 1-szym elemencie (odwrotnie posortowana tablica).

```
       cn
   /       \
  Theta(1)  c(n-1)
           /    \
      Theta(1)  c(n-2)
                  ...
                 /   \
           Theta(1)   Theta(1)
```

$$T(n) = T(n-1) + T(0) + \Theta(n) \tag{111}$$

$$T(n) = T(n-1) + \Theta(n) \leqslant \sum_{i=0}^{n} c(n-i) + \Theta(1) = \tag{112}$$

$$= c\sum_{i=0}^{n}(n-i) + \Theta(n) = \tag{113}$$

$$= c\frac{(n)(n+1)}{2} + \Theta(n) = \tag{114}$$

$$= O(n^2) \tag{115}$$

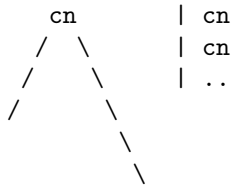
6.4 Best case Analysis for QS

Najlepiej będzie jak dzielimy na pół.

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + \Theta(n) \tag{116}$$
$$T(n) = 2T(\frac{n}{2}) + \Theta(n) \tag{117}$$
$$T(n) = \Theta(n \log n) \tag{118}$$

6.5 Specific case analysis for QS

$$T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + \Theta(n) \tag{119}$$



Po zsumowaniu każde piętro będzie miało koszt cn. Zchodzimy końca wysokości drzewa.

$$\left(\frac{9}{10}\right)^h n = 1 \tag{120}$$
$$n = \left(\frac{10}{9}\right)^h \tag{121}$$
$$h = \log_{\frac{10}{9}} n \tag{122}$$

6.6 Best/Worst case analysis for QS - Intuition

$$L(n) = 2U\left(\frac{n}{2}\right) + \Theta(n) \tag{123}$$
$$U(n) = L(n-1) + L(0) + \Theta(n) \tag{124}$$
$$\tag{125}$$

Zatem rozwiążmy układ równań:

$$L(n) = 2(L(\frac{n}{2} - 1) + \Theta(n)) + \Theta(n) L(n) \quad = 2L(\frac{n}{2} - 1) + \Theta(n) L(n) = \Theta(n \log n) \tag{126}$$

6.7 Average case analysis for QS

Rozkład T_n nie jest znany do dziś.

$$T_n = \#\text{ porównań elementów sortowanej tablicy}, |A| = n \tag{127}$$
$$X_k(n) = \begin{cases} 1 & \text{jeśli partition podzieli tablicę n-elementową na (k, n-k-1)} \\ 0 & \text{w p.p.} \end{cases} \tag{128}$$
$$0 \leqslant k \leqslant n-1 \tag{129}$$

$$E(X_k) = 1 \cdot P(X_k = 1) + 0 \cdot P(X_k = 0) = 1 \cdot P(X_k = 1) = \frac{(n-1)!}{n!} = \frac{1}{n} \tag{130}$$

Zapiszmy wobec tego równanie na T_n

$$T_n =^{distr.} \begin{cases} T_0 + T_{n-1} + n - 1 & \text{if (0,n-1) - split} \\ T_1 + T_{n-2} + n - 1 & \text{if (1,n-2) - split} \\ \vdots & \\ T_k + T_{n-1-k} + n - 1 & \text{if (k,n-k-1) - split} \\ T_{n-1} + T_0 + n - 1 & \text{if (n-1,0) - split} \end{cases} \tag{131}$$

$$T_n =^{distr.} \sum_{k=0}^{n-1} X_k(T_k + T_{n-k-1} + n - 1) \tag{132}$$

$$E(T_n) = E\left(\sum_{k=0}^{n-1} X_k(T_k + T_{n-k-1} + n - 1)\right) = \tag{133}$$

$$E(T_n) = \sum_{k=0}^{n-1} E\left(X_k(T_k + T_{n-k-1} + n - 1)\right) = \tag{134}$$

$$E(T_n) = \sum_{k=0}^{n-1} E\left(X_k\right) \cdot E(T_k + T_{n-k-1} + n - 1) = \tag{135}$$

$$E(T_n) = \frac{1}{n} \sum_{k=0}^{n-1} E(T_k) + E(T_{n-k-1}) + n - 1 = \tag{136}$$

$$E(T_n) = \frac{1}{n} \left(\sum_{k=0}^{n-1} E(T_k) + \sum_{k=0}^{n-1} E(T_{n-k-1}) + \sum_{k=0}^{n-1} n - 1\right) = \tag{137}$$

$$E(T_n) = \frac{1}{n} \sum_{k=0}^{n-1} E(T_k) + \frac{1}{n} \sum_{k=0}^{n-1} E(T_{n-k-1}) + \frac{1}{n} \sum_{k=0}^{n-1} n - 1 = \tag{138}$$

$$E(T_n) = \frac{1}{n} \sum_{k=0}^{n-1} E(T_k) + \frac{1}{n} \sum_{k=0}^{n-1} E(T_{n-k-1}) + n - 1 \tag{139}$$

$$E(T_n) = \frac{2}{n} \sum_{k=0}^{n-1} E(T_k) + n - 1 \tag{140}$$

$$t_n = \frac{2}{n} \sum_{k=0}^{n-1} t_k + n - 1 \quad \text{rekurencja z pełną historią} \tag{141}$$

$$\tag{142}$$

Możemy usunąć historię pisząc:

$$nt_n = 2\sum_{k=0}^{n-1} t_k + (n-1)n \tag{143}$$

$$(n-1)t_{n-1} = 2\sum_{k=0}^{n-2} t_k + (n-2)(n-1) \tag{144}$$

$$nt_n - (n-1)t_{n-1} = 2\sum_{k=0}^{n-1} t_k + (n-1)n - 2\sum_{k=0}^{n-2} t_k - (n-2)(n-1) \tag{145}$$

$$nt_n - (n-1)t_{n-1} = 2t_{n-1} + 2(n-1) \tag{146}$$

$$nt_n = (n+1)t_{n-1} + 2(n-1) \tag{147}$$

$$\frac{t_n}{n+1} = \frac{t_{n-1}}{n} + 2\frac{n-1}{n(n-1)} \tag{148}$$

$$f_n = f_{n-1} + 2\frac{n-1}{n(n+1)}, f_0, f_1 = 0 \tag{149}$$

$$f_n = 2\sum_{k=1}^{n} \frac{k-1}{k(k+1)} = \tag{150}$$

$$f_n = 2\sum_{k=1}^{n} \frac{2}{k+1} - \frac{1}{k} = \tag{151}$$

$$f_n = 4\sum_{k=1}^{n} \frac{1}{k+1} + 2\sum_{k=1}^{n} \frac{1}{k} = \tag{152}$$

$$f_n = 4(H_{n+1} - 1) + 2H_n \tag{153}$$

$$f_n = 4H_{n+1} + 2H_n - 1 \tag{154}$$

$$f_n = 2H_n - 4 + \frac{4}{n+1} \tag{155}$$

$$E(T_n) = t_n = (n+1)f_n = 2nH_n + 2H_n - 4(n+1) + 4 \tag{156}$$

$$H_n = \ln n + \gamma + \frac{1}{2n} + \Theta\left(\frac{1}{n^2}\right) \tag{157}$$

Widzimy, że wiodący czynnik $T_n = 2n\ln n + \Theta(n)$. Wiemy dlaczego QS jest dobry - średnio wykona $2n\ln n$ porównań asymptotycznie.