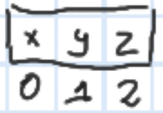
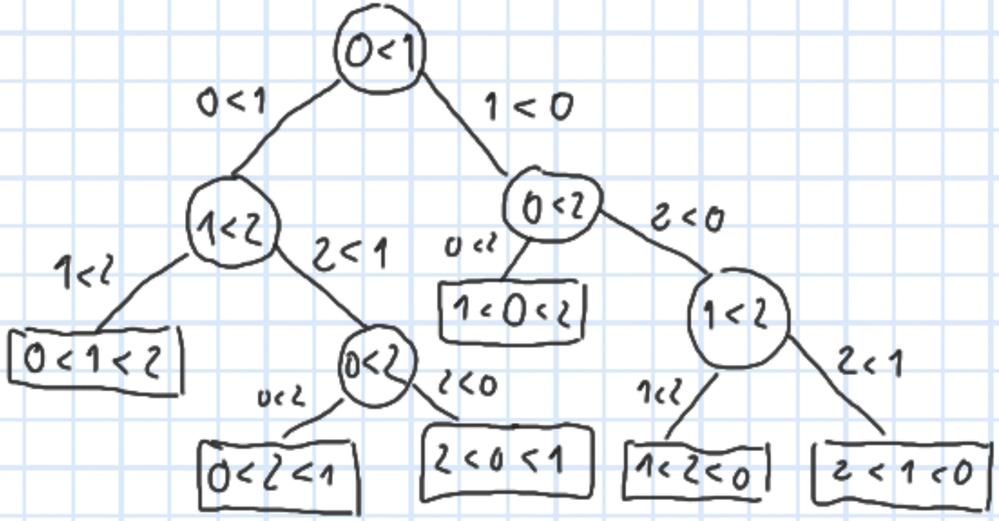
Algorytmy i Struktury Danych

Wykład 4

Dolne ograniczenie na szybkość sortowania





Jeśli drzewo opisuje sortowanie tablicy n elementowej to musi mieć co najmniej $n!$ liści

Drzewo (liniowe) o wysokości h ma najwyżej 2^h liści

$$n! \leq 2^h$$

wysokość drzewa = liczba porównań!

wysokość naszego drzewa to co najmniej

$$n! = n(n-1)(n-2)\cdot\ldots\cdot 1 = \text{2 skł. pamięte}$$

$$n(n-1)(n-2)\ldots\left(\frac{n}{2}\right)\left(\frac{n}{2}-1\right)\ldots 1 \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$$n\log n = \log n^n \geq \log n! \geq \log\left(\frac{n}{2}\right)^{\frac{n}{2}} = \frac{n}{2}\left(\log n - 1\right)$$

Sortowanie przez zliczanie (Counting Sort)

Mamy tablicę A, która zawiera liczby naturalne z zakresu $\{0, 1, \ldots, k-1\}$

złożoność $O(n+k)$

```
def countingsort(A, k):
    n = len(A)
    C = [0] * k
    B = [0] * n
    for i in range(n):
        C[A[i]] += 1
    for i in range(1, k):
        C[i] = C[i] + C[i-1]
    for i in range(n-1, -1, -1):
        B[C[A[i]] - 1] = A[i]
        C[A[i]] -= 1
```

```
for i in range(n):
    A[i] = B[i]
```

k = 4

	0	1	2	3	4	5	6	7
A	2	0	3	1	1	2	3	2

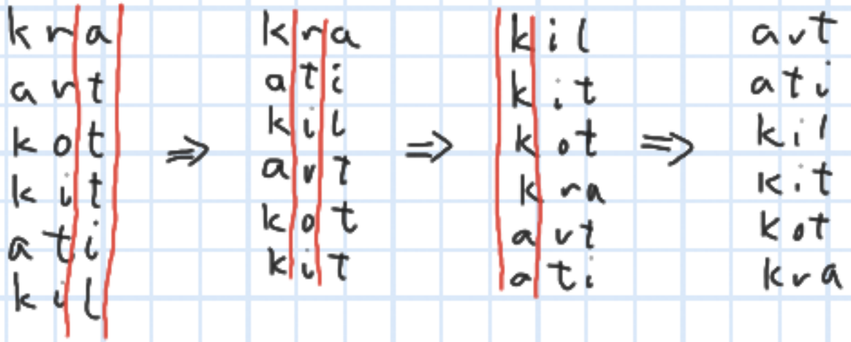
	0	1	2	3
C	1	2	3	2

	0	1	2	3
C	1	3	6	8

	0	1	2	3	4	5	6	7
B	0	1	1	2	2	2	3	3

0	1	3	6

Radix Sort – Sortowanie pozycyjne



Sortowanie n słów, każde o długości t zajęło $\Theta(nt)$ czasu

---

Kubełków może być mniej niż n, ale powinna to być liniowa postać $\propto n$

Bucket Sort – sortowanie kubełkowe

A - tablica liczb wymiernych

$0 \leq A[i] < 1$, elementy A były wygenerowane zgodnie z rozkładem jednostajnym

n – rozmiar tablicy

Tworzymy n kubełków (list na elementy A)

kubełek 0	kubełek 1	...	kubełek $n-1$
$[0, \frac{1}{n})$	$[\frac{1}{n}, \frac{2}{n})$		$[\frac{n-1}{n}, \frac{n}{n})$

Przeglądamy tablicę A i aktualną liczbę $A[i]$ umieszczamy w kubełku $\lfloor n \cdot A[i] \rfloor$

Każdy kubełek sortujemy (np. używając sortowania prostego)

Przepisujemy dane z kubełków do A w kolejności posortowanej

Złożoność $\Theta(n)$

Magiczne 5ki – wybór k-tego elementu w czasie $O(n)$ bez użycia "ukwadratowienia"

A – tablica n liczb

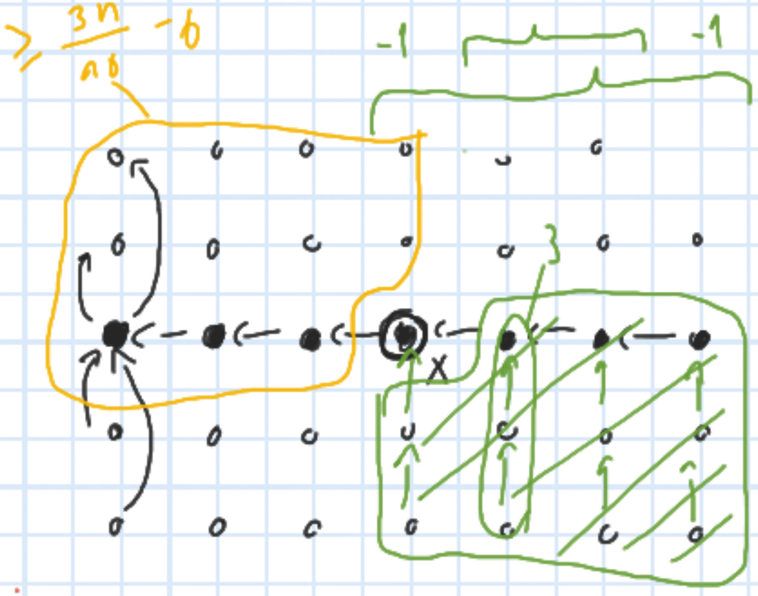
Chcemy znaleźć liczbę, która po posortowaniu tablicy A stałaby na pozycji k

1. Podziel A na $\lceil \frac{n}{5} \rceil$ grup po 5 elementów
2. Rekurencyjnie znajduje x, medianę wśród median naszych piątek

← wybór pivota

3. Wykonaj "partition" używając x jako pivot
4. Jeśli x jest na pozycji k, to zwróć x
   Jeśli x jest na pozycji dalszej niż k to szukaj rekurencyjnie w lewej części tablicy, w przeciwnym razie w prawej

klasyczny select



$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6$$

$$T(n) = \begin{cases} \Theta(1) & , n \leq \text{pewna stała} \\ T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + \Theta(n) & \end{cases}$$

można wybrać c tak, że ta suma jest ujemna

Twierdzimy, że $T(n) \leq cn$ dla pewnej $c \geq 1$

$$T(n) \leq c\left\lceil \frac{n}{5} \right\rceil + c\left(\frac{7n}{10} + 6\right) + an$$

$$\leq \frac{cn}{5} + \frac{7cn}{10} + 6c + an + c = cn + \left(-\frac{1}{10}cn + 7c + an\right)$$