Algorytmy i Struktury Danych

Wykład 2

Zadania offline – można wysyłać dopóki nie oceniono dwóch zadań danej osoby

Problem: Sortowanie

Dane: ciąg $a_1, \dots, a_n$ razem z operatorem $\leqslant$

Zadanie: Znaleźć ciąg $a_1', \dots, a_n'$ będący permutacją wejściowego ciągu, taki że

$a_1' \leqslant a_2' \leqslant \dots \leqslant a_n'$

$a_i = a_j \iff a_i \leqslant a_j \ i \ a_j \leqslant a_i$

Warianty

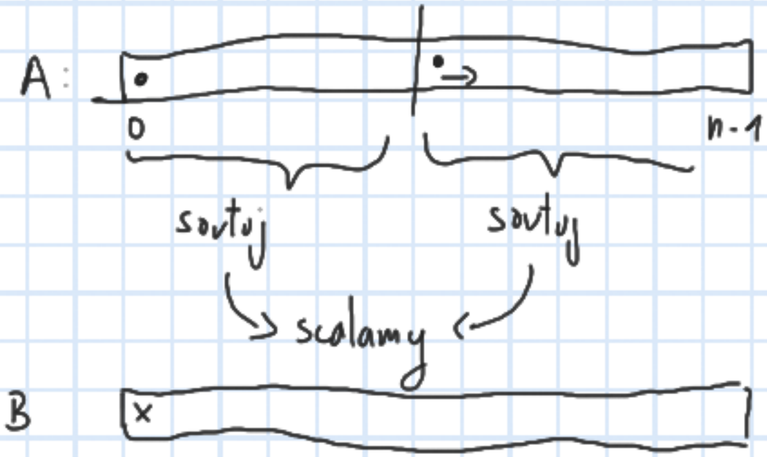
reprezentacja danych
- tablica
- lista (1-kier., 2-kier.)
- plik

działanie "w miejscu" – alg. sort. działa w miejscu jeśli poza pamięcią potrzebną na dane wejściowe używa $O(1)$ pamięci

Stabilność – algorytm sortowania jest stabilny, jeśli ma tę własność, że dla każdych dwóch a_i, a_j t.ż. $a_i = a_j$, $i < j$, w posortowanych danych a_i występuje przed a_j

Algorytm Merge Sort - sortowanie przez scalanie

- wykorzystuje technikę dziel i zwyciężaj

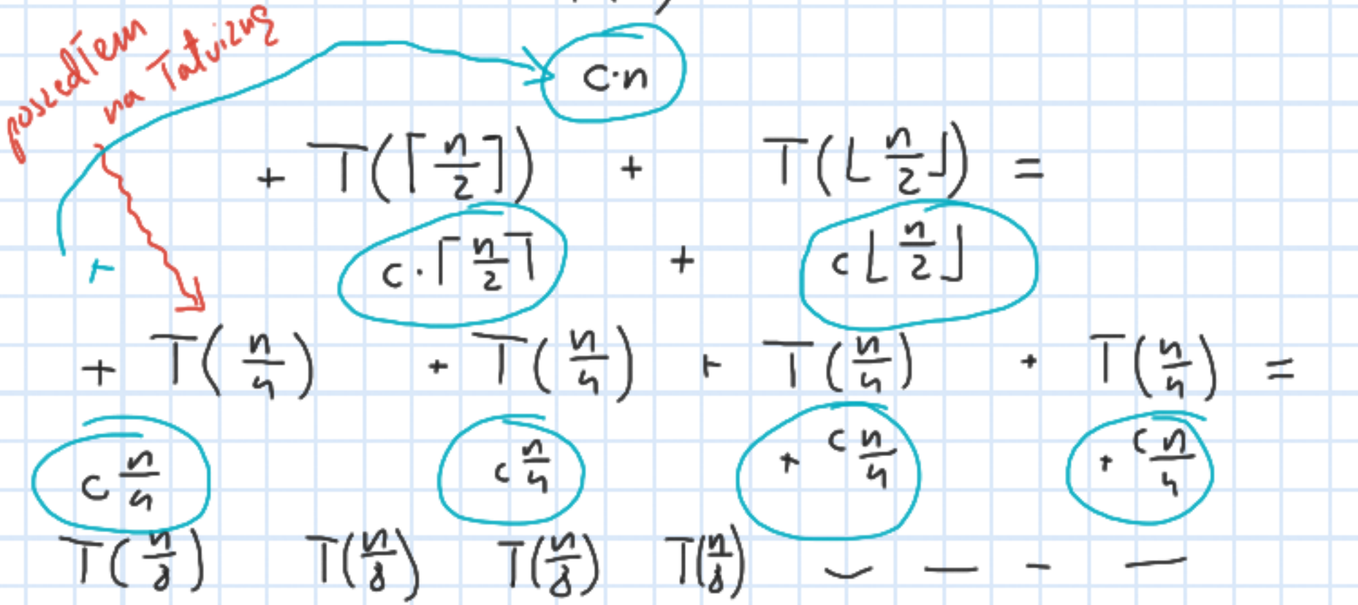


Chcemy oszacować złożoność obliczeniową MergeSorta

$$T(n) = \begin{cases} c & , \text{ dla } n \leq 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + cn, & \text{upp.} \end{cases}$$

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

$T(n) =$



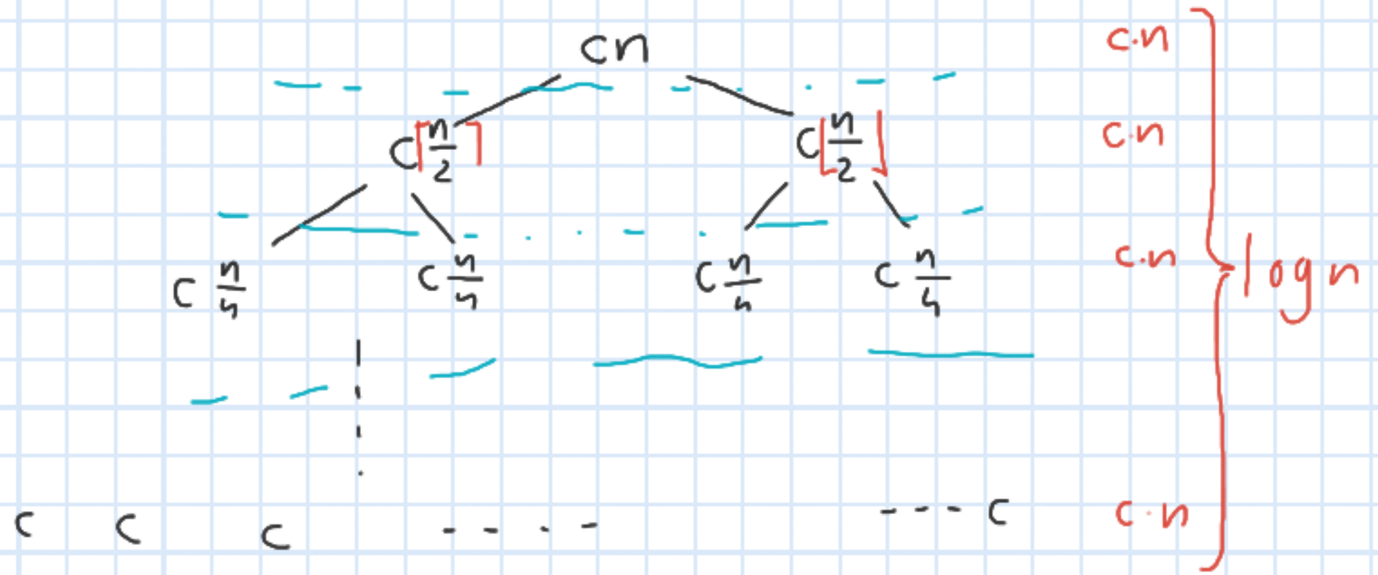
$c \cdot n + T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) =$

$c \cdot \lceil \frac{n}{2} \rceil + c \lfloor \frac{n}{2} \rfloor +$

$+ T(\frac{n}{4}) + T(\frac{n}{4}) + T(\frac{n}{4}) + T(\frac{n}{4}) =$

$c\frac{n}{4} \quad c\frac{n}{4} \quad + c\frac{n}{4} \quad + c\frac{n}{4}$

$T(\frac{n}{8}) \quad T(\frac{n}{8}) \quad T(\frac{n}{8}) \quad T(\frac{n}{8}) \quad \smile \; - \; - \; -$



Algorytm QuickSort

Także działa na zasadzie "dziel i zwyciężaj"

A: | 7 | 2 | 5 | 23 | 19 | 7 |

① wybierz pivot, np $x = 7$

② podziel tablicę na elementy $\leq$ od pivota oraz $>$ od pivota

7, 2, 5	7	19, 23
$\leq$	=	$>$
L		P

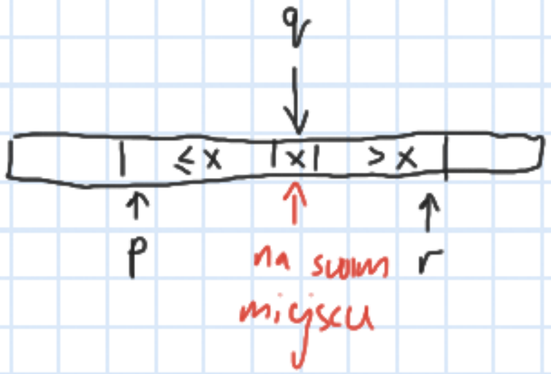
③ posortuj rekurencyjnie L i P

Gdyby partition zawsze dzielił tablicę na połowy, to QuickSort działałby w czasie:

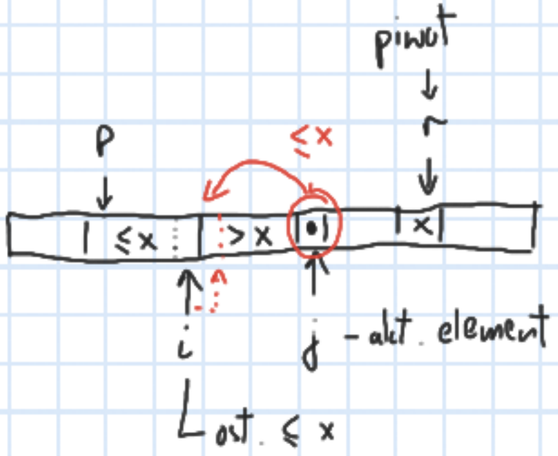
$$T(n) = \begin{cases} c, & n \leq 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + cn, & \text{w p.p.} \end{cases}$$

czyli w czasie $O(n \log n)$

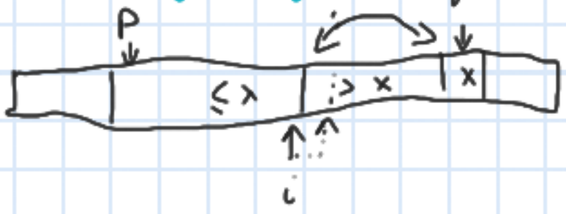
```
def quicksort(A, p, r):
    if p < r:
        q = partition(A, p, r)
        quicksort(A, p, q-1)
        quicksort(A, q+1, r)
```



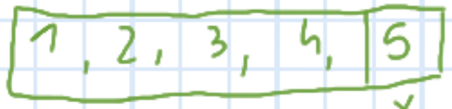

```
def partition(A, p, r):
    x = A[r]
    i = p-1
    for j in range(p, r):
        if A[j] ≤ x:
            i += 1
            swap(A[i], A[j])
    swap(A[i+1], A[r])
    return i+1
```

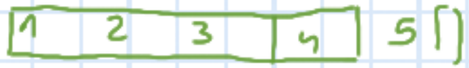


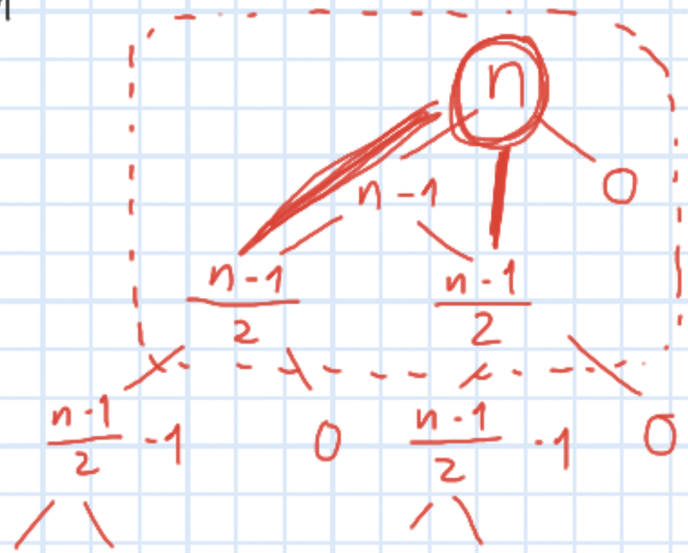
A[i], A[j] = A[j], A[i]



Dla niekorzystnych danych QuickSort może działać w czasie $O(n^2)$





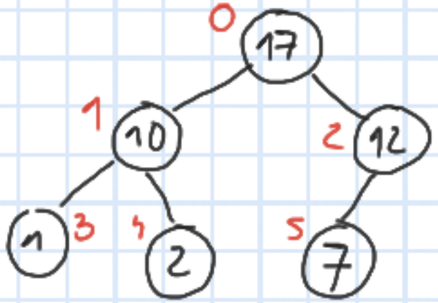


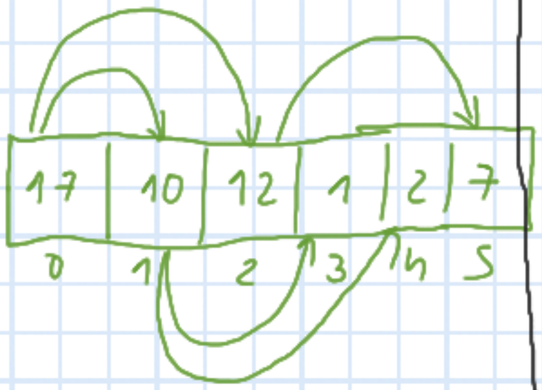
Zadanie obowiązkowe: proszę zaimplementować QuickSort tak, żeby używał najwyżej $O(\log n)$ dodatkowej pamięci

```
def quick sort(A, p, r):
while p < r :
    q = partition(A, p, r)
    QuickSort(A, p, q-1)
    p = q+1
```

Heap Sort - sortowanie przez kopcowanie

Kopiec - drzewo binarne, gdzie każdy węzeł ma wartość ≥ niż jego dzieci





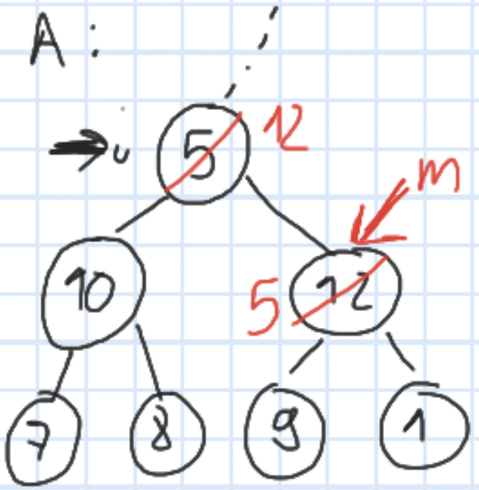
$left(i) = 2i + 1$

$right(i) = 2i + 2$

$parent(i) = \lfloor (i-1)/2 \rfloor$

Zadanie obowiązkowe: Proszę zaimplementować ustawianie dowolnego elementu do kopca

```
def heapify(A, n, i):
    l = left(i)
    r = right(i)
    m = i
    if l < n and A[l] > A[m]: m = l
    if r < n and A[r] > A[m]: m = r
    if m ≠ i:
        swap(A[i], A[m])
        heapify(A, n, m)
```



$O(\log n)$

```
def buildheap(A):
    n = len(A)
    for i in range(parent(n-1), -1, -1):
        heapify(A, n, i)
```

for i = parent(n-1) to 0

$O(n \log n)$ / $O(n)$

```
def heapsort(A):
    n = len(A)
    build_heap(A)
    for i in range(n-1, 0, -1):
        swap(A[0], A[i])
        heapify(A, i, 0)
```

$O(n \log n)$

Czas działania build_heap

$\left\lceil \frac{n}{2^{h+1}} \right\rceil$ – w kopcu o n elementach tyle jest "podkopców" o wysokości h

$$\sum_{h=0}^{\lfloor \log n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil \cdot h = O\left(\sum_{h=0}^{\lfloor \log n \rfloor} \frac{n \cdot h}{2^{h+1}} \right) = O(n)$$

$$n \cdot \sum_{h=0}^{\infty} \frac{h}{2^{h+1}}$$

$$f(x) = 1 + x + x^2 + x^3 + \ldots = \frac{1}{1-x}$$

$$f'(x) = 1 + 2x + 3x^2 + \ldots = \frac{1}{(1-x)^2}$$

$$x f'(x) = x + 2x^2 + 3x^3 + \ldots = \frac{x}{(1-x)^2} = 2$$

$x = \frac{1}{2}$