Algorytmy i Struktury Danych

Wykład 3

Analiza złożoności procedury Build Heap

wysokość kopca $\Theta(\log n)$

liczba elementów o wysokości h: $\left\lceil \frac{n}{2^{h+1}} \right\rceil$

Szacowanie kosztu obliczeniowego:

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

$$= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^{h+1}}\right) = O(n)$$

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

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

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

$$x = \frac{1}{2} \Rightarrow \sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$$

$$= \frac{1}{2} + 2\left(\frac{1}{2}\right)^2 + 3\cdot\left(\frac{1}{2}\right)^3 + \ldots$$

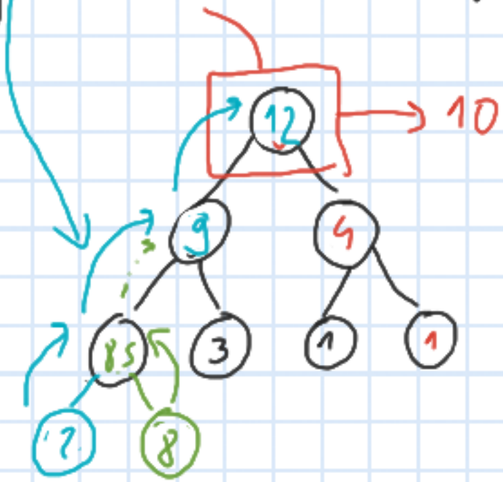
$$= \frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \ldots$$

Kopiec jako kolejka priorytetowa

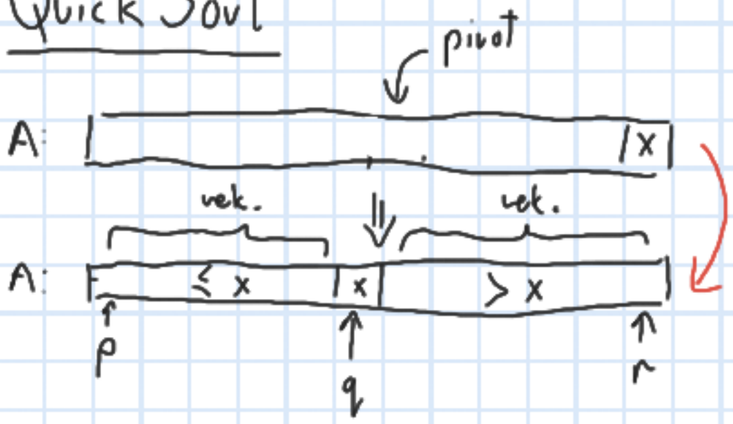
Struktura danych, do której można układać elementy i wyciągać je zgodnie z porządkiem

Operacje

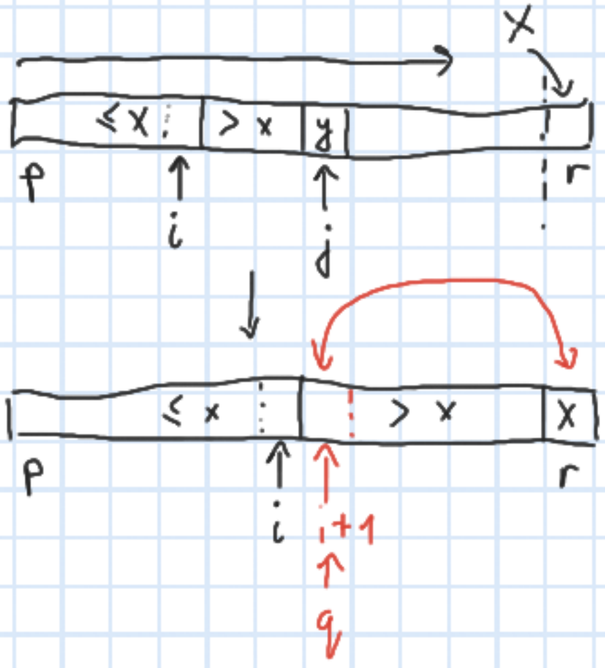
- insert – ustaw element $\Theta(\log n)$
- extractmax – wyjmij element największy



Quick Sort




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



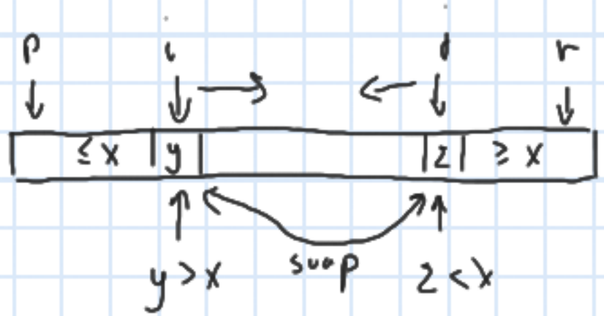
partition Lomuto

```
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
```

zł. liniowa

$O(n)$

w miejsce A[r] umieszczamy "lepszy pivot"
- losowy element tablicy
- mediana z pierwszego, ostatniego i środkowego elementu
- ...

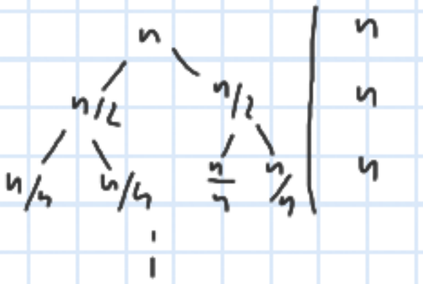


partition Hoare'a

Analiza złożoności obliczeniowej QuickSorta

- idealne podziały $\longrightarrow$ $T(n) = \begin{cases} c, & n \leq 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + cn, & n > 1 \end{cases}$

$= \Theta(n \log n)$



- pechowe podziały $\longrightarrow$ $T(n) = \begin{cases} c, & n \leq 1 \\ T(n-1) + cn, & n > 1 \end{cases}$

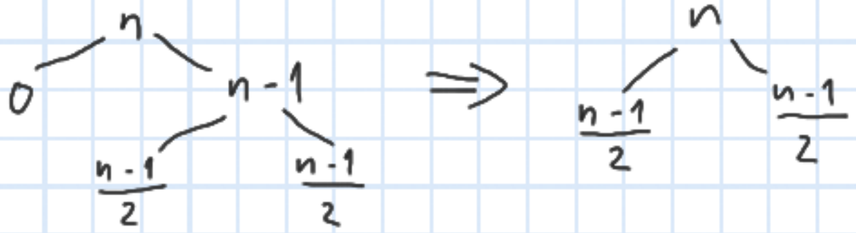
$T(n) = cn + T(n-1)$

$= cn + c(n-1) + T(n-2)$

$\vdots$

$= cn + c(n-1) + c(n-2) + \ldots + c = \Theta(n^2)$

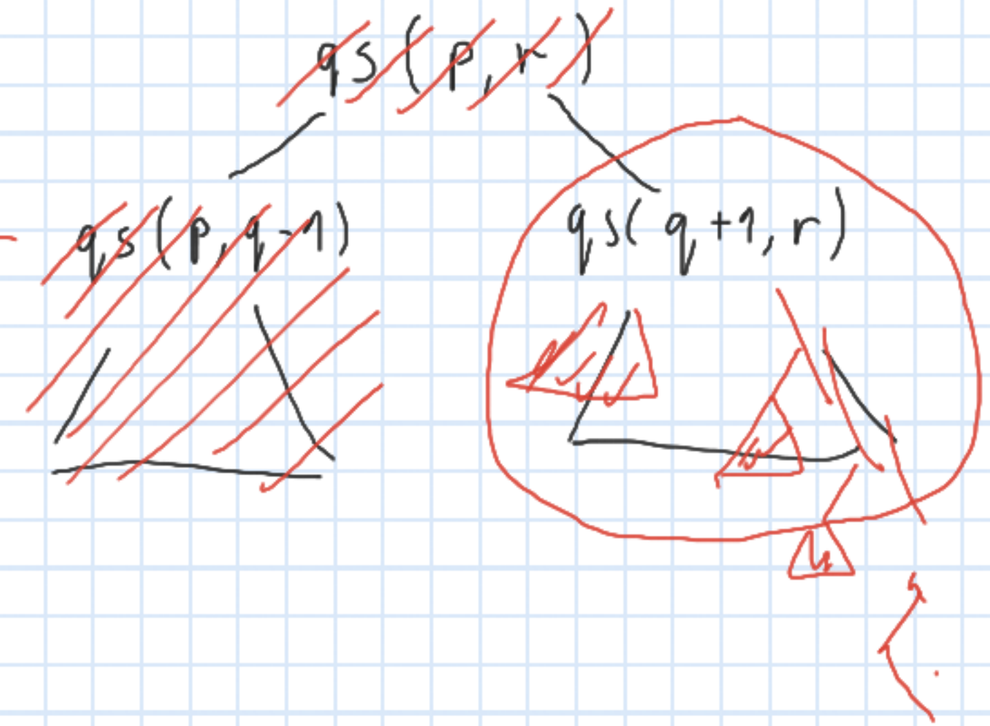
- mieszanka szczęścia i pecha
  - co drugi poziom $\rightarrow$ podziały idealne
  - co drugi najgorsze



Usunięcie rekurencji ogonowej

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

quicksort(A, q+1, r)



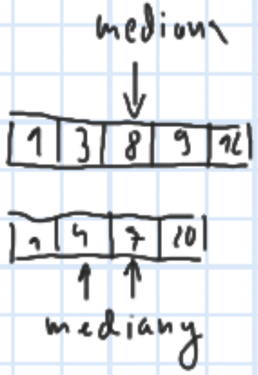
Statystyki pozycyjne

select (A, k) – k-ta statystyka pozycyjna, czyli element A, który byłby na k-tej pozycji po posortowaniu

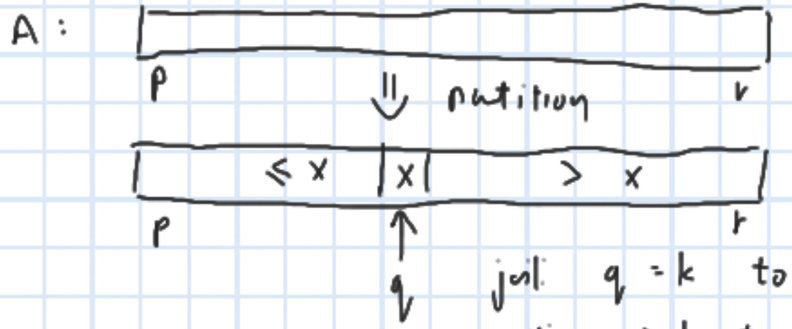
select (A, 0) = min (A)

select (A, len(A) - 1) = max (A)

select (A, len(A)/2) = mediana A



select (A, k)



jeśli $q = k$ to zwracamy x

jeśli $q > k$ to rekurencyjnie, ale na "połowie" tablicy

$q < k$

złożoność czasowa przy idealnych podziałach

$$T(n) = \begin{cases} c, & n = 1 \\ T(\frac{n}{2}) + cn, & n > 1 \end{cases}$$

$$T(n) = cn + \frac{cn}{2} + \frac{cn}{4} + \frac{cn}{8} + \dots$$

$$= O(n)$$