Algorytmy i struktury danych

Wykład 2

sortowanie wewnętrzne

prof. dr hab. inż. Andrzej Obuchowicz

Instytut Sterowania i Systemów Informatycznych
Uniwersytet Zielonogórski
a.obuchowicz@issi.uz.zgora.pl
p. 424 A2

14 października 2017

Spis treści

Sformułowanie problemu

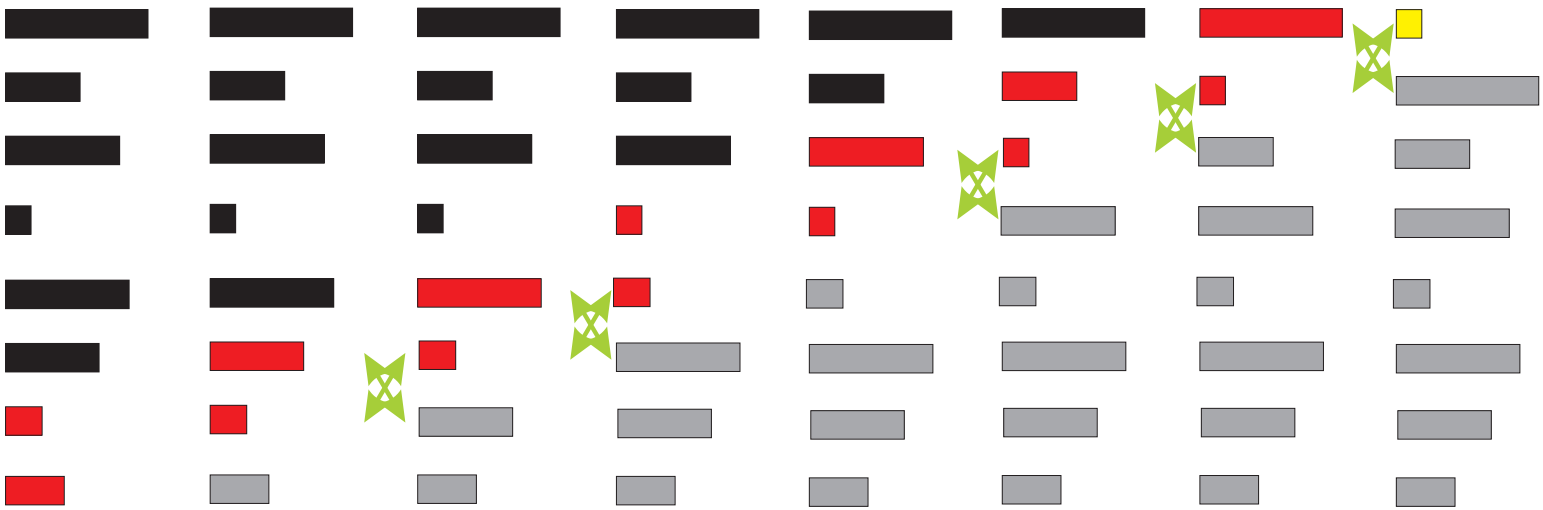
problem sortowania

Dane: tablica $(a_1, a_2, \ldots, a_n)$ o n elementach typu porządkowego;
Szukane: tablica o tych samych elementach ale uporządkowana niemalejąco.

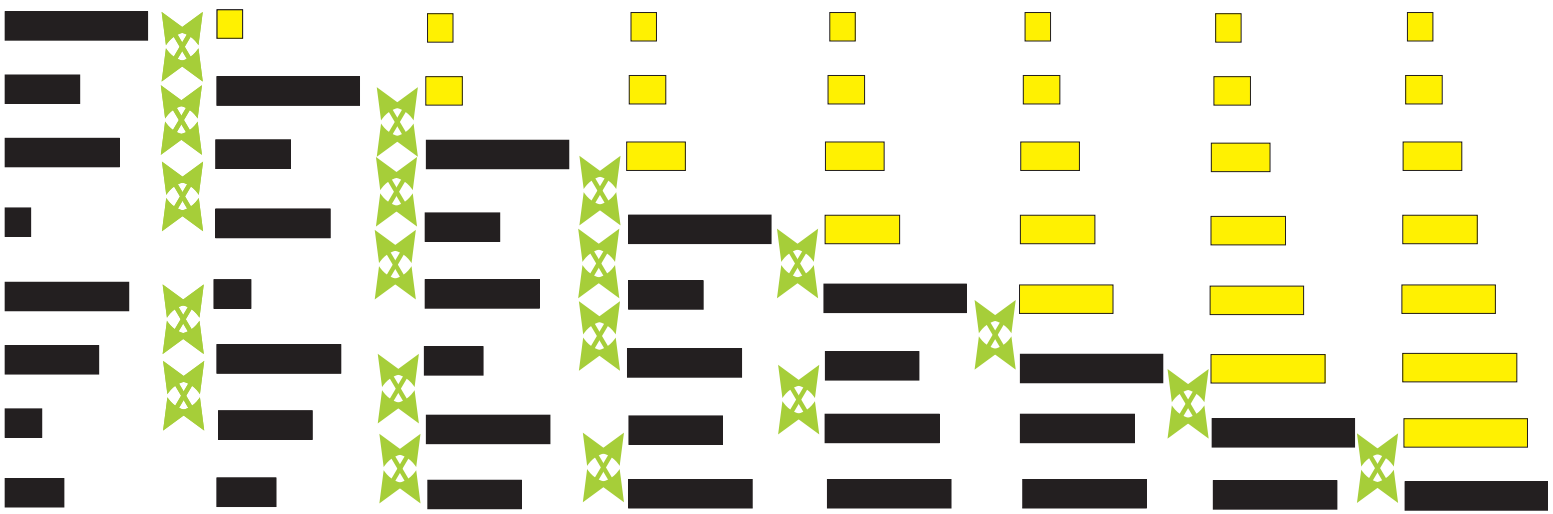
sortowanie wewnętrzne – założenia:

- bezpośredni dostęp do każdego elementu (np. poprzez jego wskaźnik) – tablica mieści się w pamięci RAM;
- sortowanie za pomocą porównań i wymian elementów wewnątrz jednej tablicy, nie tworzy się nowych tablic i innych struktur pomocniczych.

Pojedyncza iteracja wewnętrzna w sortowaniu bąbelkowym



Pełne sortowanie bąbelkowe

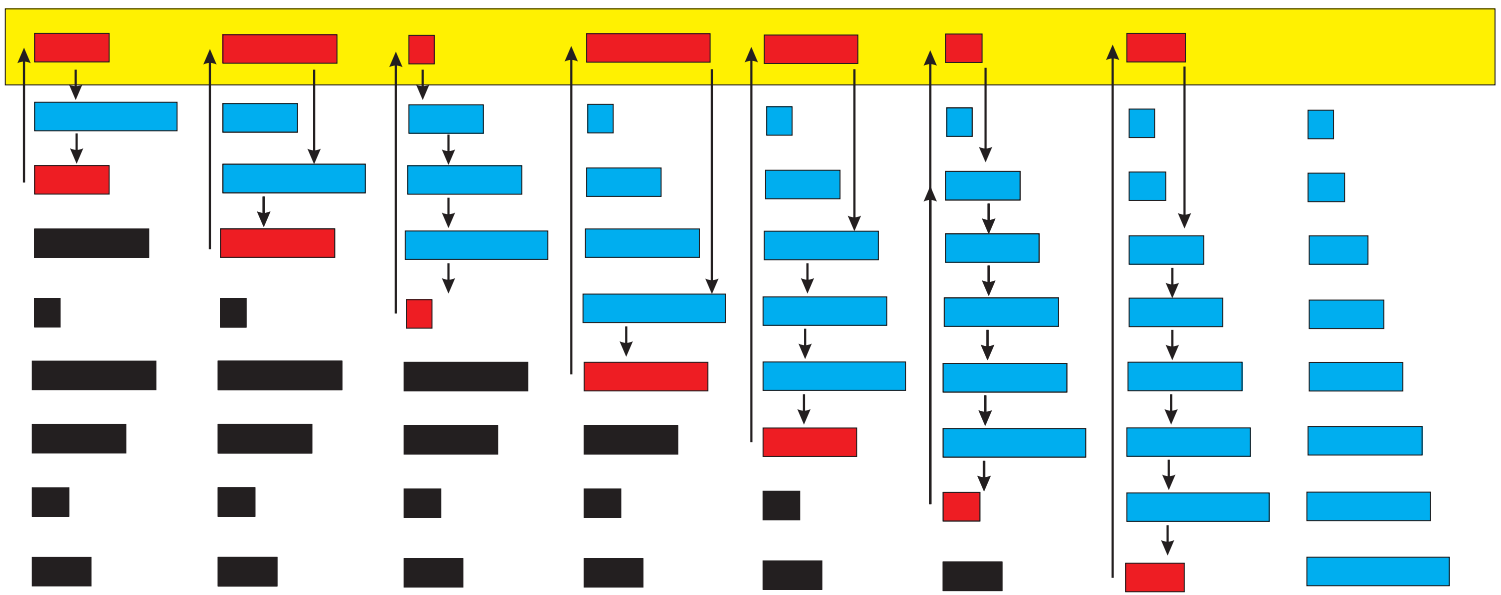


Kod algorytmu sortowania bąbelkowego

```
bubblesort(n,t)
    for i:=1 to n-1
       do for j:= n-1 downto i
            do if t[j+1]<t[j]
                 then help:=t[j]
                      t[j]:=t[j+1]
                      t[j+1]:=help
```

- liczba porównań elementów
  $l_{por}(bubblesort) = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2}$
- liczba przepisań elementów $l_{prz}(bubblesort) = 3 \times l_{por}(bubblesort) = \frac{3n(n-1)}{2}$

Zasada działania sortowania przez wstawianie



Kod algorytmu sortowania przez wstawianie

```
insertsort(n,t)
  for i:=2 to n
    do t[0]:=t[i]
       j:=i-1
       while t[j]>t[0]
         do t[j+1]:=t[j]
            j:=j-1
       t[j+1]:=t[0]
```

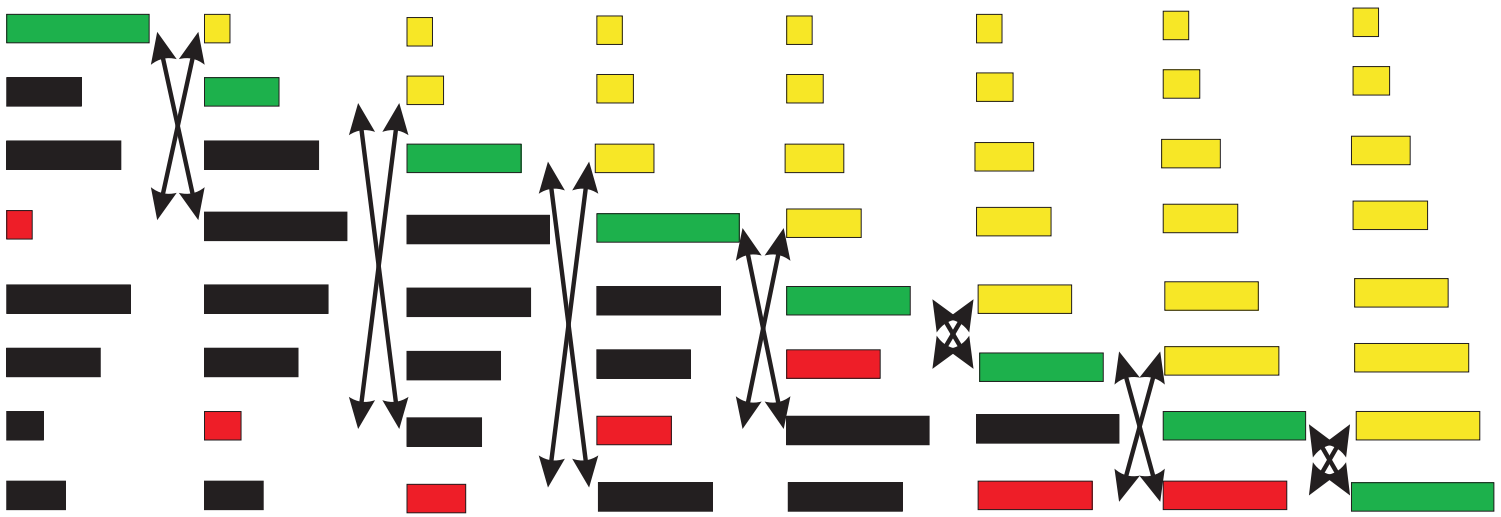
- liczba porównań elementów
  $l_{por}(insertsort) = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2}$
- liczba przepisań elementów
  $l_{prz}(insertsort) = (2 + (n-1)) + (2 + (n-2)) + \ldots + (2+2) + (2+1) =$
  $2(n-1) + \frac{n(n-1)}{2} = \frac{(n-1)(n+4)}{2}$

Kod algorytmu sortowania przez wstawianie połówkowe

```
insertsortBS(n,t)
  for i:=2 to n
    do help:=t[i]
       first:=1
       last:=i-1
       while first<last
         do medium:=(first+last)div 2
            if help<t[medium]
              then last:=medium-1
              else first:=medium+1
       for j:=i-1 downto first
       do t[j+1]:=t[j]
       t[first]:=help
```

- liczba porównań elementów $l_{por}(insertsortBS) \sim n\log(n)$
- liczba przepisań elementów $l_{prz}(insertsortBS) = l_{prz}(insertsort)$

Zasada działania sortowania przez wybieranie



Kod algorytmu sortowania przez wybieranie

```
selectsort(n,t)
  for i:=1 to n-1
    do min:=i
       for j:= i+1 to n
         do if t[j]<t[min]
              then min:=j
       help:=t[i]
       t[i]:=t[min]
       t[min]:=help
```

- liczba porównań elementów
  $l_{por}(selectsort) = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2}$
- liczba przepisań elementów $l_{prz}(selectsort) = 3(n-1)$

Porównanie sortowań prostych

nazwa	l_{por}	l_{prz}
bubblesort	$\frac{n(n-1)}{2}$	$3\frac{n(n-1)}{2}$
insertsort	$\frac{n(n-1)}{2}$	$\frac{(n+4)(n-1)}{2}$
insertsortBS	$\sim n\log(n)$	$\frac{(n+4)(n-1)}{2}$
selectsort	$\frac{n(n-1)}{2}$	$3(n-1)$

Zasada działania sortowania szybkiego

	5	7	3	8	4	0	1	9	2	6
5	5	7	3	8	4	0	1	9	2	6
5	2	7	3	8	4	0	1	9	5	6
5	2	1	3	8	4	0	7	9	5	6
5	2	1	3	0	4	8	7	9	5	6
	2	1	3	0	4	8	7	9	5	6

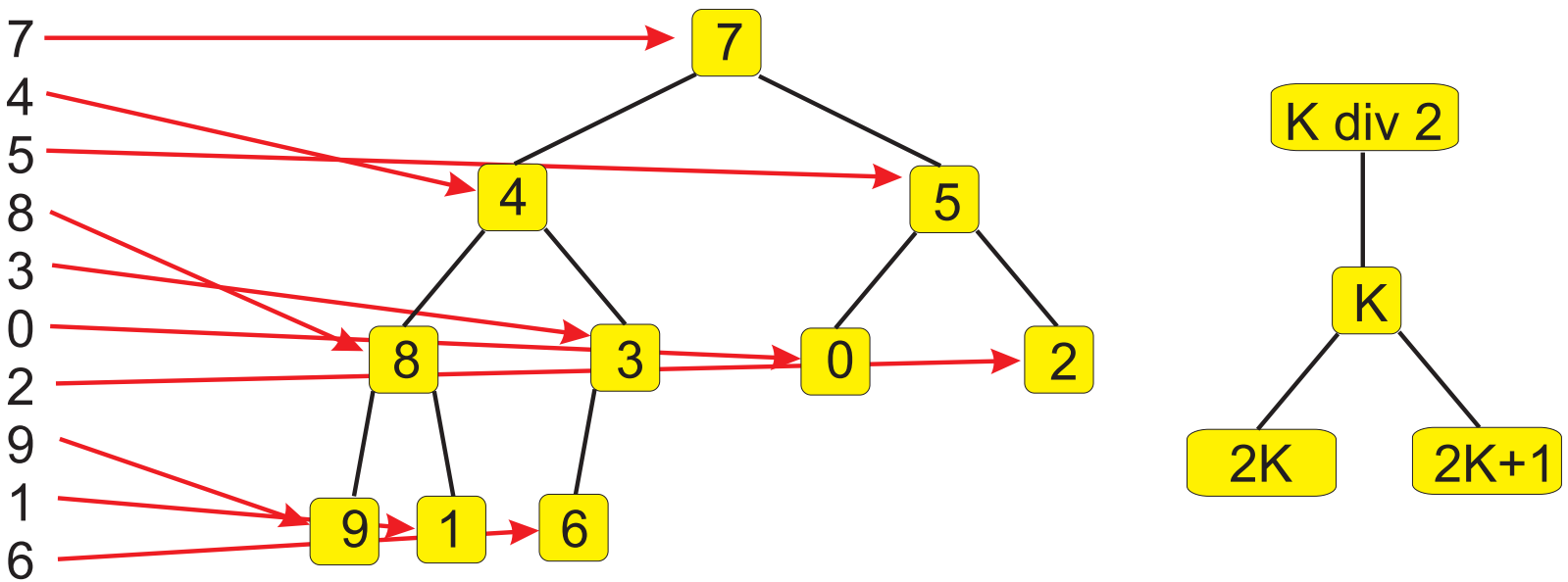
	2	1	3	0	4		8	7	9	5	6
2	2	1	3	0	4	8	8	7	9	5	6
2	0	1	3	2	4	8	6	7	9	5	8
						8	6	7	5	9	8
	0	1	3	2	4		6	7	5	9	8

- liczba porównań elementów
  $l_{por}(quicksort) = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2}$
- średnia liczba porównań elementów $\langle l_{prz}(quicksort) \rangle \sim n \log(n)$

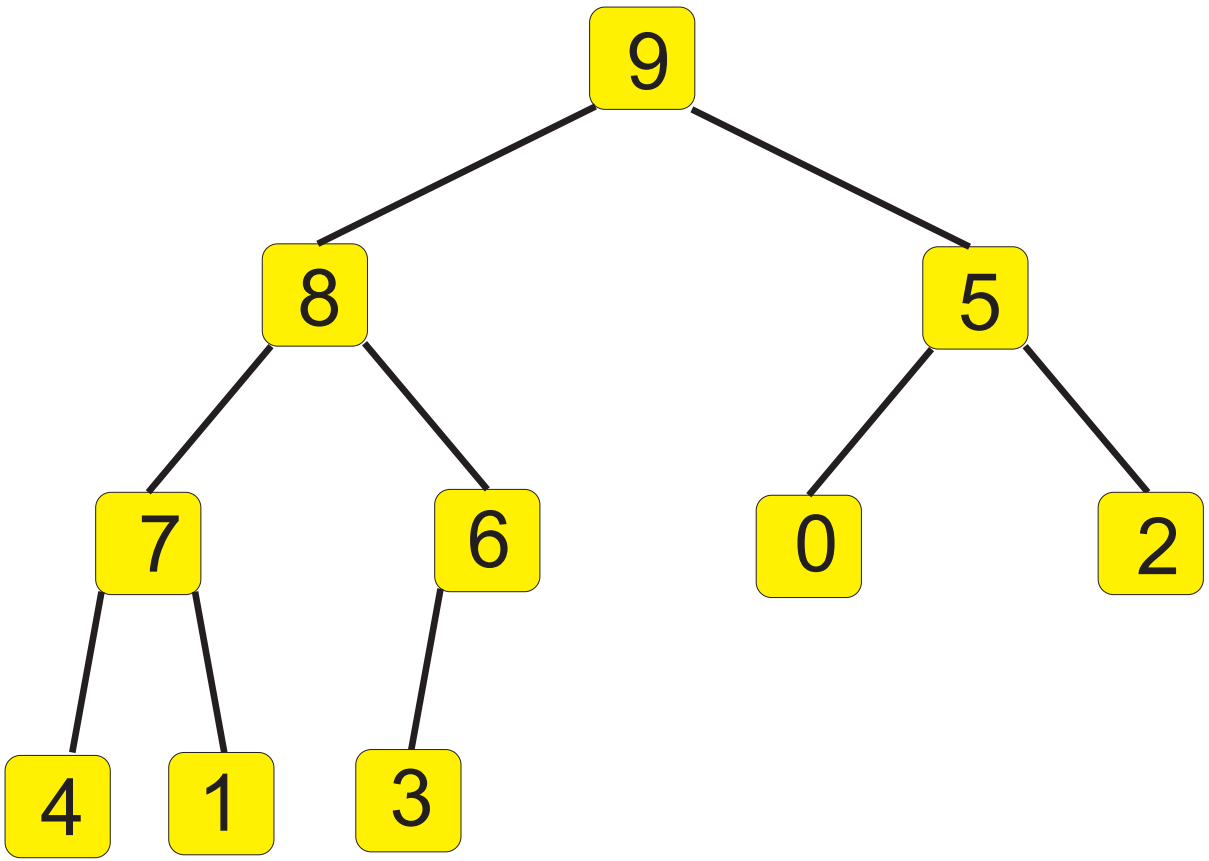
Kod algorytmu sortowania szybkiego

```
quicksort(first,last,t)
  main:=t[first]
  i:=first
  j:=last
  while i<=j
  do while t[i]<main
       do i:=i+1
     while t[j]>main
       do j:=j-1
     if i<=j
       then help:=t[i]
            t[i]:=t[j]
            t[j]:=help
            i:=i+1
            j:=j-1
  if first<j then quicksort(first,j,t)
  if i<last then quicksort(i,last,t)
```

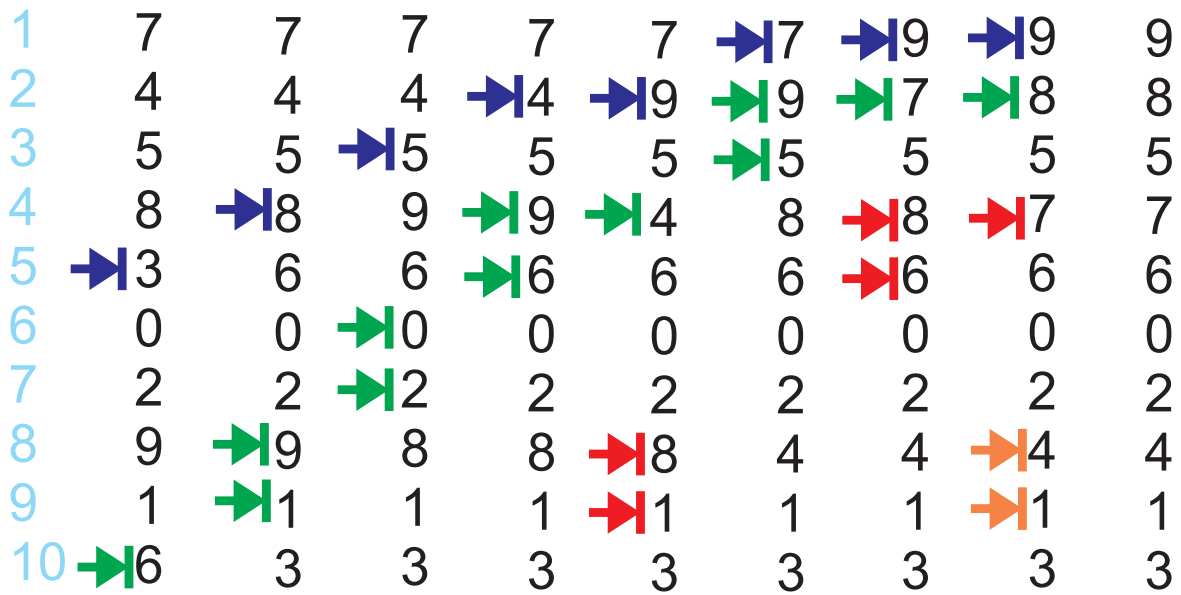
Tablicowa reprezentacja drzewa binarnego



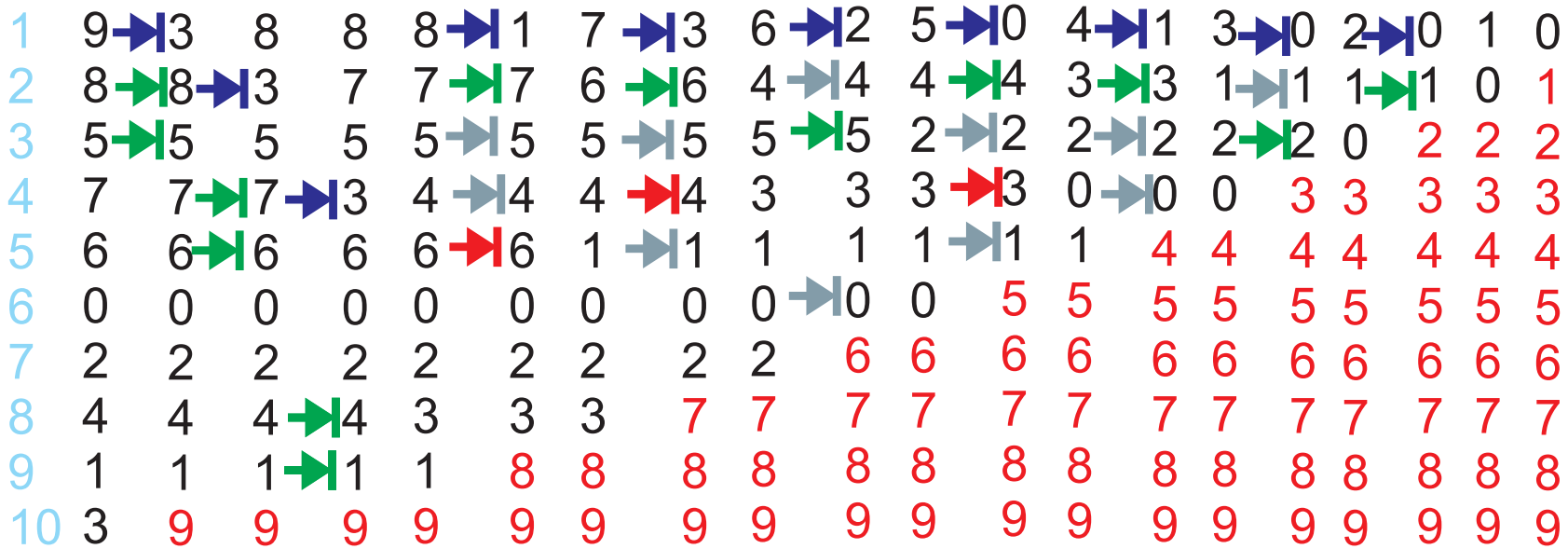
Stóg: potomkowie niewięksi od rodzica



Tworzenie stogu z dowolnej tablicy



Sortowanie właściwe



Sortowanie stogowe – algorytm przeczesania

```
brush(r,end,t)
  ok:=true
  while 2r<=end & ok
    do if 2r=end
         then oldson=2r
         else if t[2r]>t[2r+1]
                then oldson=2r
                else oldson=2r+1
       if t[r]<t[oldson]
         then help:=t[r]
              t[r]:=t[oldson]
              t[oldson]:=help
              r:=oldson
         else ok:=false
```

Sortowanie stogowe – algorytm

```
heapsort(n,t)
  for i:=n div 2 downto 1
    do brush(i,n,t)
  for i:=n downto 2
    do help:=t[1]
       t[1]:=t[i]
       t[i]:=help
       brush(1,i-1,t)
```

- liczba porównań elementów $l_{por}(heapsort) \sim n\log(n)$

A w następnym tygodniu między innymi

Tematy jakie zostaną poruszone na następnym wykładzie:

1. sortowanie zewnętrzne – sformułowanie problemu,
2. scalanie plików,
3. scalanie Fibonacci'ego,

Dziękuję za uwagę!!!