ASD, Wyk. 04, 2023.3.17

Sortowanie

Definicja problemu

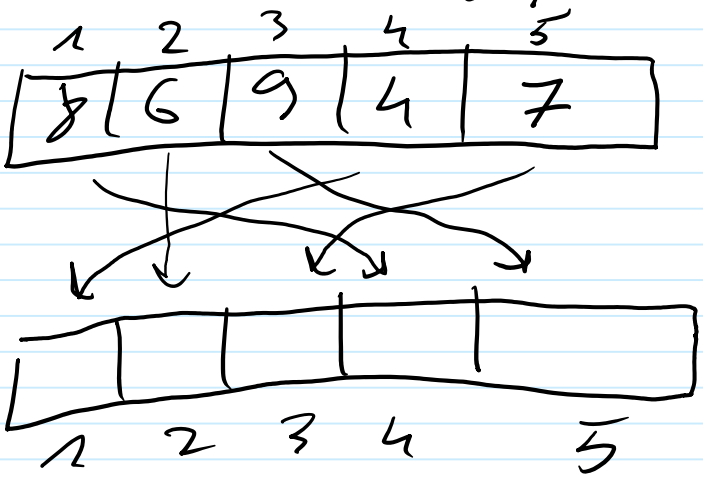
Dane we: tablica $t[1..n]$ obiektów, na których mamy zdefiniowany porządek liniowy $\leq$.

Szukamy permutacji $\pi \in S_n$, $\pi : \{1,\dots,n\} \xrightarrow[\text{na}]{1-1} \{1,\dots,n\}$ takiej, że π porządkuje tablicę t:

$$\forall_{1 \leq i < j \leq n} \left(\pi(i) \leq \pi(j) \iff t[i] \leq t[j] \right)$$

Czyli $\pi(i)$ mówi, na jakie miejsce przenieść element $t[i]$.

Np.



$\pi(1) = 4$

$\pi(2) = 2$

$\pi(3) = 5$

$\pi(4) = 1$

$\pi(5) = 3$

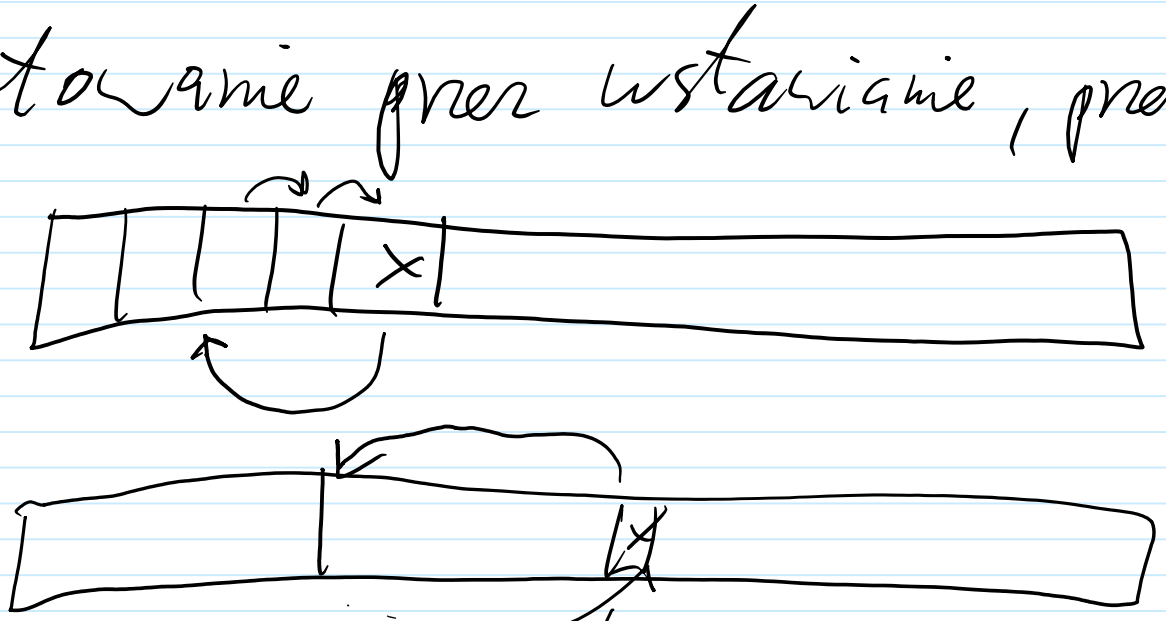
Def Tablica $t[1..n]$ nazywamy posortowaną gdy $\forall i<j\leq n \; (t[i] \leq t[j])$

Po wykonaniu sortowania mamy posortowaną tablicę.

Jeżeli elementy tablicy są parami różne, to permutacja jest wyznaczona jednoznacznie. (oczywiście zawsze istnieje taka permutacja)

Przykłady sortowań

1. Sortowanie przez wstawianie, przez wybór



szukamy minimum w niposortowanej części

sortowanie bąbelkowe,

$O(n^2)$

Dla większych zbiorów danych będziemy sortować w czasie $O(n \cdot \log_2(n))$

Osiągniemy ten czas sortowaniem przez scalanie (merge sort).

Algorytm merge sort

WE: t[begin .. end-1]

chcemy posortować tę część tablicy

① Znajdź środek tablicy

$$mid = begin + \frac{end - begin}{2}$$

if(begin >= end-1) return.

② Sortujemy rekurencyjnie t[begin, mid-1] oraz t[mid, end-1]

③ Scalamy dwie posortowane części tablicy w jedną.

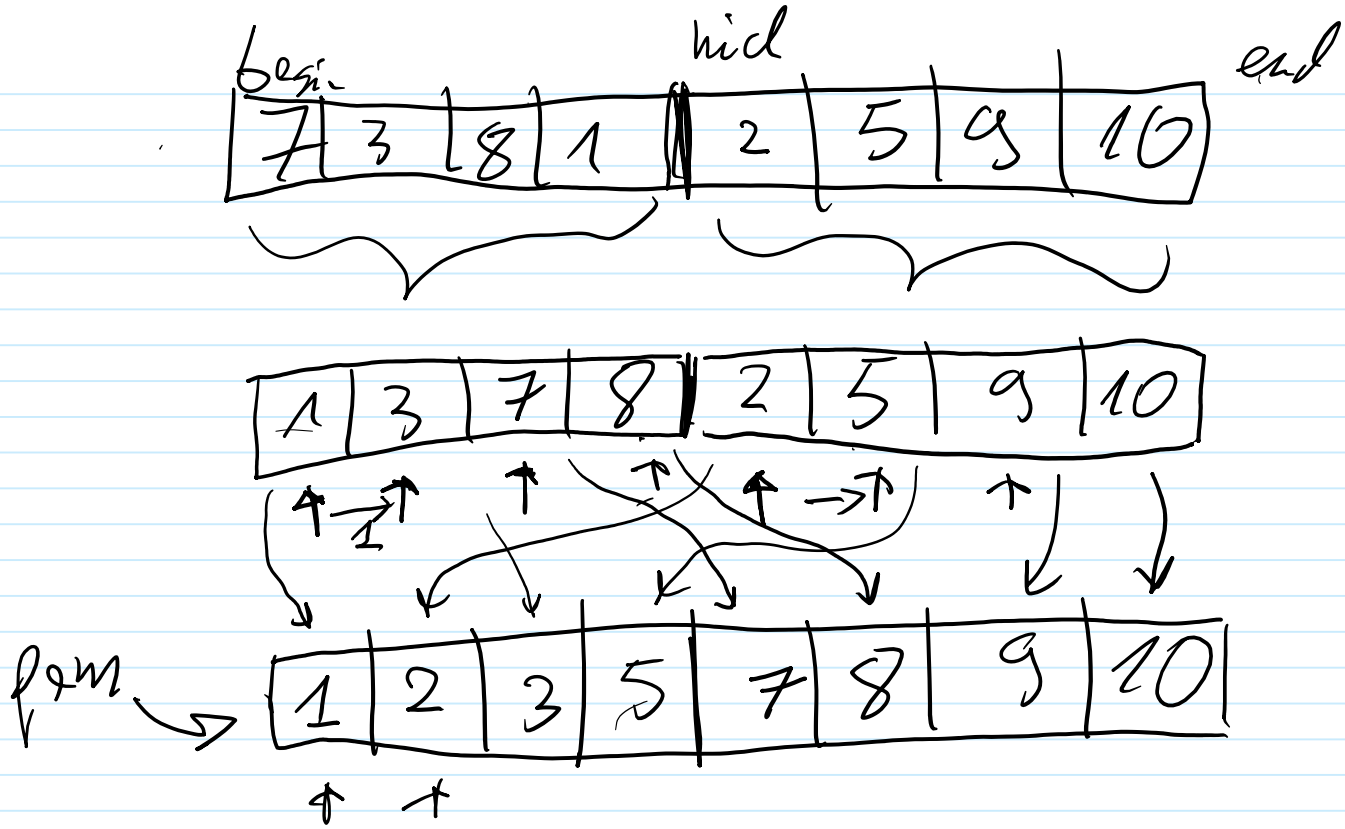
$$T(n) = 2T\left(\frac{n}{2}\right) + f(n)$$

Jeżeli $f(n) = O(n)$, to

$$T(n) = O(n \cdot \log_2(n)).$$

Ale musimy wykonać scalanie w czasie liniowym.

Przykład



begin < end - 1

Scalenie odbywa się w czasie liniowym.

kod merge sort

```
%sortujemy tablicę t[begin,end-1]

void merge_sort(int *t, int begin, int end){
%puste i jednoelemenowe tablice są posortowane
   if(begin>= end-1) then return;
%znajdujemy środek tablicy;
   int mid=begin + floor((end-begin)/2);
%rekurencyjnie sortujemy obie części tablicy
   merge_sort(t,begin,mid);
   merge_sort(t,mid,end);
%scalamy dwie posortowane tablice
   merge(t,begin,mid, end);
}
```

Kod scalania

```
%scalanie dwóch posortowanych podtablic w czasie liniowym
void merge(int *t, int begin, int mid, int end){
   int t_aux[end];
   int i=begin, j=mid, k=begin;
   while(i<mid && j<end){
      if(t[i]<=t[j]){
         t_aux[k]=t[i];
         i++; k++;
      }
      if(t[i]>t[j]){
         t_aux[k]=t[j];
         j++; k++;
      }
   }
   if(i==mid){
      while(j<end){
         t_aux[k]=t[j];
         k++;j++;
      }
   }
   else{   // j==end
      while(i<mid){
         t_aux[k]=t[i];
         k++;i++;
      }
   }
   t[begin,...,end]=t_aux[begin,...,end];
}
```

Scalamy posortowane podtablice
t[begin, mid-1], i
t[mid, end-1], j

Koszt liniowy

1) Sortowanie przez scalanie dzieli bardzo regularnie

a) zawsze równomiernie dzieli problem na podproblemy

b) sortowanie to wykonuje na każdych danych wejściowych tą samą ilość przestawień i przepisania do tablicy pomocniczej (i z powrotem)

(2) Czas działania i wykorzystana pamięć.

czas $O(n \cdot \log(n))$, n - rozmiar tablicy

pamięć:

w klasycznej implementacji $O(n)$

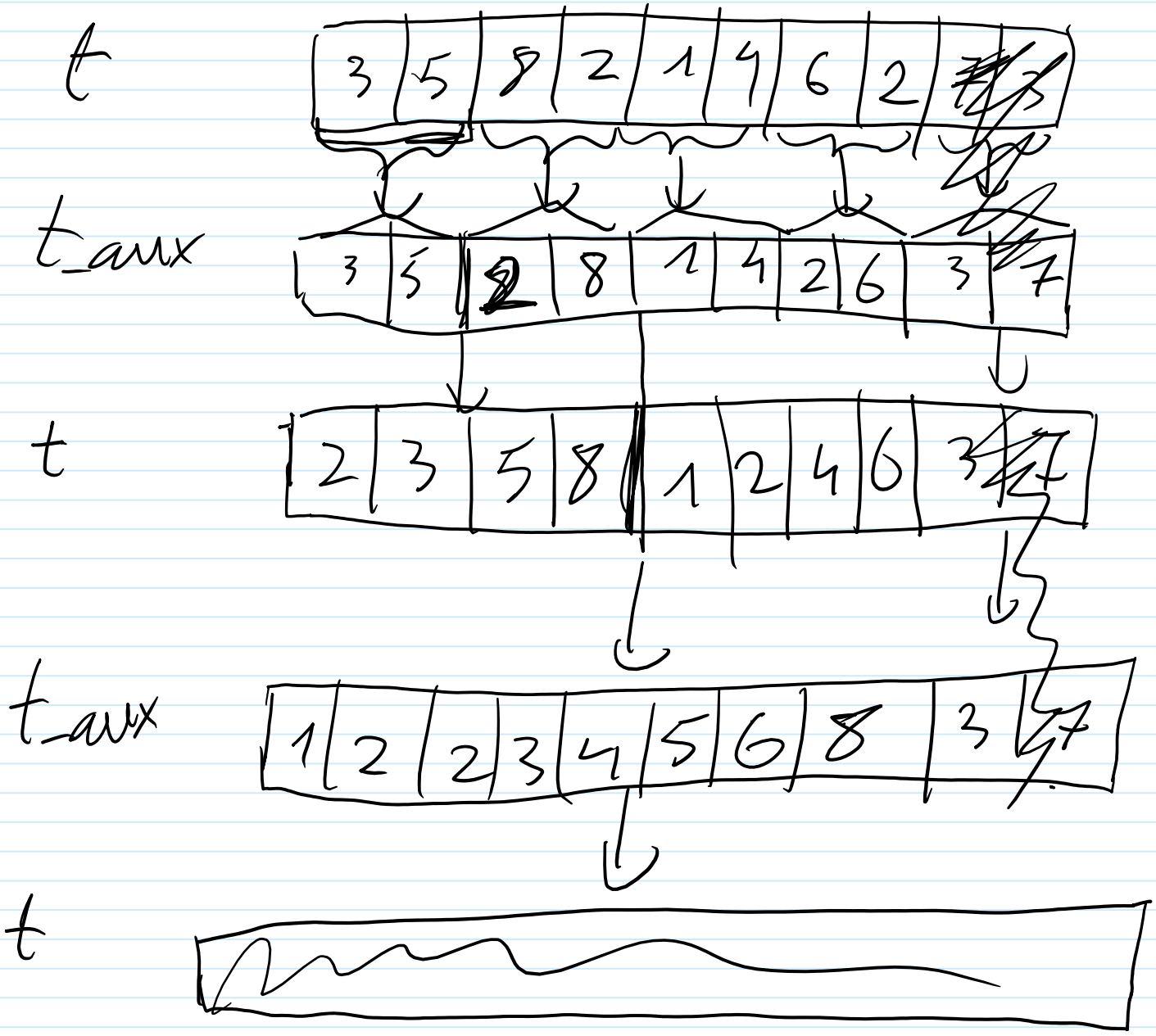
bo potrzebujemy tablicy pomocniczej

Sama rekurencja też wymaga dodatkowej pamięci $O(\log_2(n))$

(3) Jak można ulepszyć merge sort?

można pozbyć się przepisywania scalonej tablicy w tablicy pomocniczej t_aux z powrotem do t

czyli nasz algorytm mergesort raz sortowałby tablicę t, a raz tablicę pomocniczą t_aux, i.t.d.



W ten sposób zmniejszamy ilość przepisywanych o połowę

Możemy pozbyć się rekurencji.

technika Bottom-up

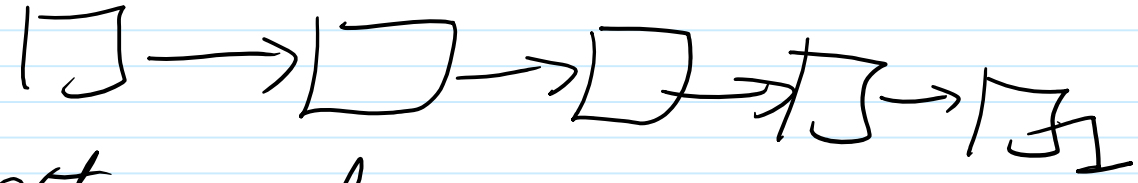
Zaczynamy od scalania tablic jedno-elementowych w dwuelementowe

potem dwuelementowych w cztero-elementowe

itd.

(4) Nie potrzebujemy tablicy pomocniczej.
Możemy wykonać scalanie
w pamięci stałej ("w miejscu")
("in place")
ale wtedy algorytm działa dłużej
i algorytm scalania trudniej zaimplementować.

(5) Sortowanie przez scalanie
da się zaimplementować na listach
bez straty w czasie działania.
w tej mamy $O(n \log_2(n))$



dostęp sekwencyjny
koszt od czytu i-tego elementu to $O(i)$

⑥ To sortowanie da się zaimplementować stabilnie.

Def Sortowanie jest stabilne, jeśli nie zmienia kolejności równych sobie elementów.