# Duomenų rūšiavimo algoritmai

Raimondas Čiegis

Matematinio modeliavimo katedra, e-paštas: rc@vgtu.lt

Rugsėjo 1 d., 2023

## Duomenų rūšiavimo uždaviniai

Jau matėme, kad informacijos paieška efektyviausia, kai saugome surūšiuotus duomenis.

Susipažinsime su svarbiausiais rūšiavimo algoritmais ir jų sudėtingumo įvertinimo metodais. Yra sukurta daug tokių algoritmų, todėl sprendžiant taikomuosius uždavinius, svarbu mokėti pasirinkti tinkamiausią algoritmą.

## Uždavinio formulavimas

Tarkime, kad turime duomenų aibę $A = (a_1, a_2, \ldots, a_N)$.

Šiuos duomenis galime palyginti, t. y. patikrinti ar teisingas teiginys, kad $a_i < a_j$, kai $i \neq j$.

## Uždavinio formulavimas

Tarkime, kad turime duomenų aibę $A = (a_1, a_2, \dots, a_N)$.

Šiuos duomenis galime palyginti, t. y. patikrinti ar teisingas teiginys, kad $a_i < a_j$, kai $i \neq j$.

Reikia taip pertvarkyti $A$, kad gautoje aibėje $A^{'} = (a_1^{'}, a_2^{'}, \dots, a_N^{'})$ visi elementai būtų išdėstyti didėjimo tvarka, t. y.:

$$a_i^{'} \leq a_{i+1}^{'}, \ \ \text{kai} \ \ i = 0, 1, \dots, N - 1.$$

## Uždavinio formulavimas

Tarkime, kad turime duomenų aibę $A = (a_1, a_2, \dots, a_N)$.

Šiuos duomenis galime palyginti, t. y. patikrinti ar teisingas teiginys, kad $a_i < a_j$, kai $i \neq j$.

Reikia taip pertvarkyti $A$, kad gautoje aibėje $A' = (a'_1, a'_2, \dots, a'_N)$ visi elementai būtų išdėstyti didėjimo tvarka, t. y.:

$$a'_i \leq a'_{i+1}, \ \text{kai} \ i = 0, 1, \dots, N-1.$$

Aišku, galime nagrinėti ir aibes, kuriose elementai išdėstyti atvirkščia – mažėjimo tvarka.

Kasdien naudojame surūšiuotą informaciją:

Kasdien naudojame surūšiuotą informaciją:

duomenis yra pateikiami abėcėlės tvarka,

Kasdien naudojame surūšiuotą informaciją:

duomenis yra pateikiami abėcėlės tvarka,

tvarkaraščiai sudaromi atsižvelgiant į įvykio pradžios laiką (pvz. traukinio išvykimo laiką),

Kasdien naudojame surūšiuotą informaciją:

duomenis yra pateikiami abėcėlės tvarka,

tvarkaraščiai sudaromi atsižvelgiant į įvykio pradžios laiką (pvz. traukinio išvykimo laiką),

kataloguose gaminiai išdėstomi pagal pasirinktą požymį, pvz. jų kainą (pirmiausia pateikiami pigiausi gaminiai, bilietai, variantai).

## Rūšiavimo uždavinio sudėtingumas

Nagrinėsime tokius rūšiavimo algoritmus, kuriuos realizuojame naudodami dvi operacijas:

## Rūšiavimo uždavinio sudėtingumas

Nagrinėsime tokius rūšiavimo algoritmus, kuriuos realizuojame naudodami dvi operacijas:

- dviejų aibės $A$ elementų lyginimą,

## Rūšiavimo uždavinio sudėtingumas

Nagrinėsime tokius rūšiavimo algoritmus, kuriuos realizuojame naudodami dvi operacijas:

- dviejų aibės $A$ elementų lyginimą,
- dviejų elementų sukeitimą vietomis.

## Rūšiavimo uždavinio sudėtingumas

Nagrinėsime tokius rūšiavimo algoritmus, kuriuos realizuojame naudodami dvi operacijas:

- dviejų aibės $A$ elementų lyginimą,
- dviejų elementų sukeitimą vietomis.

## Rūšiavimo uždavinio sudėtingumas

Nagrinėsime tokius rūšiavimo algoritmus, kuriuos realizuojame naudodami dvi operacijas:

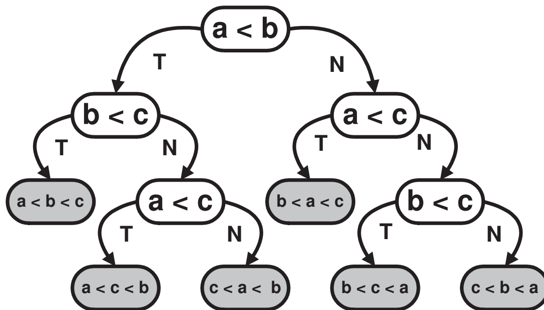
- dviejų aibės $A$ elementų lyginimą,
- dviejų elementų sukeitimą vietomis.

Tokius algoritmus galime pavaizduoti dvejetainiu medžiu.

Medžio viršūnės žymi dviejų konkrečių elementų lyginimo veiksmą, šio medžio lapai apibrėžia surūšiuotas $N$ elementų aibes.

Imkime paprastą pavyzdį, kai turime tris elementus $A = (a, b, c)$.

Šią aibę rūšiuojame lygindami elementų poras. Visos galimos situacijos yra pavaizduotos paveiksle.




T žymi briauną, kai tikrinamoji sąlyga yra teisinga, N – neteisinga

Matome, kad blogiausiu atveju tenka atlikti tris elementų lyginimo veiksmus. Todėl toks ir yra bet kurio rūšiavimo algoritmo *blogiausiojo atvejo* sudėtingumas, kai rūšiuojame tris elementus.

Matome, kad blogiausiu atveju tenka atlikti tris elementų lyginimo veiksmus. Todėl toks ir yra bet kurio rūšiavimo algoritmo *blogiausiojo atvejo* sudėtingumas, kai rūšiuojame tris elementus.

Taigi dvejetainio medžio lapų skaičius turi būti ne mažesnis už maksimalų elementų išdėstymų skaičių.

Matome, kad blogiausiu atveju tenka atlikti tris elementų lyginimo veiksmus. Todėl toks ir yra bet kurio rūšiavimo algoritmo *blogiausiojo atvejo* sudėtingumas, kai rūšiuojame tris elementus.

Taigi dvejetainio medžio lapų skaičius turi būti ne mažesnis už maksimalų elementų išdėstymų skaičių.

$h$ aukščio dvejetainis medis turi $2^h$ viršūnes-lapus. Imdami $N$ elementų, galime sudaryti $N!$ skirtingų kombinacijų,
todėl bet kurio rūšiavimo algoritmo elementų porų lyginimo skaičius $K$ yra nemažesnis už nelygybės

$$2^K \geq N!$$

sprendinį.

Matome, kad blogiausiu atveju tenka atlikti tris elementų lyginimo veiksmus. Todėl toks ir yra bet kurio rūšiavimo algoritmo *blogiausiojo atvejo* sudėtingumas, kai rūšiuojame tris elementus.

Taigi dvejetainio medžio lapų skaičius turi būti ne mažesnis už maksimalų elementų išdėstymų skaičių.

$h$ aukščio dvejetainis medis turi $2^h$ viršūnes-lapus. Imdami $N$ elementų, galime sudaryti $N!$ skirtingų kombinacijų,
todėl bet kurio rūšiavimo algoritmo elementų porų lyginimo skaičius $K$ yra nemažesnis už nelygybės

$$2^K \geq N!$$

sprendinį.
Trijų elementų atveju gauname:

$$2^3 \geq 3!$$

Bendruoju atveju, panaudodami Stirlingo formulę gauname, kad skirtingų elementų porų lyginimo skaičius yra nemažesnis už

$$K \geq N \log N - N \log e\,.$$

Bendruoju atveju, panaudodami Stirlingo formulę gauname, kad skirtingų elementų porų lyginimo skaičius yra nemažesnis už

$$K \geq N \log N - N \log e \,.$$

Todėl bet kurio rūšiavimo algoritmo blogiausio atvejo sudėtingumas

$$W_b \geq N \log N.$$

Svarbi ir gera žinia, kad vieną efektyvų rūšiavimo algoritmą jau sudarėme ankstesnėse paskaitose.

Svarbi ir gera žinia, kad vieną efektyvų rūšiavimo algoritmą jau sudarėme ankstesnėse paskaitose.

Prisiminkime, kad spausdindami dvejetainio paieškos medžio elementus InOrder tvarka, mes gauname surūšiuotą aibę.

Svarbi ir gera žinia, kad vieną efektyvų rūšiavimo algoritmą jau sudarėme ankstesnėse paskaitose.

Prisiminkime, kad spausdindami dvejetainio paieškos medžio elementus InOrder tvarka, mes gauname surūšiuotą aibę.

Tai rekursinis viršūnių aplankymo algoritmas ir jo sudėtingumas tik $O(N)$ veiksmų. Netikėta, kad šis įvertis yra geresnis už mūsų gautą apatinį bet kokio bendrojo rūšiavimo algoritmo sudėtingumo įvertį.

Svarbi ir gera žinia, kad vieną efektyvų rūšiavimo algoritmą jau sudarėme ankstesnėse paskaitose.

Prisiminkime, kad spausdindami dvejetainio paieškos medžio elementus InOrder tvarka, mes gauname surūšiuotą aibę.

Tai rekursinis viršūnių aplankymo algoritmas ir jo sudėtingumas tik O(N) veiksmų. Netikėta, kad šis įvertis yra geresnis už mūsų gautą apatinį bet kokio bendrojo rūšiavimo algoritmo sudėtingumo įvertį.

Kur padarėme klaidą?

Svarbi ir gera žinia, kad vieną efektyvų rūšiavimo algoritmą jau sudarėme ankstesnėse paskaitose.

Prisiminkime, kad spausdindami dvejetainio paieškos medžio elementus InOrder tvarka, mes gauname surūšiuotą aibę.

Tai rekursinis viršūnių aplankymo algoritmas ir jo sudėtingumas tik O(N) veiksmų. Netikėta, kad šis įvertis yra geresnis už mūsų gautą apatinį bet kokio bendrojo rūšiavimo algoritmo sudėtingumo įvertį.

Kur padarėme klaidą?

Klaidos nepadarėme, reikia papildomai įvertinti paieškos medžio sudarymo sudėtingumą. Tarp mūsų išnagrinėtų algoritmų sparčiausias buvo AVL medžio formavimo algoritmas, jo sudėtingumas $O(N \log N)$.

Išnagrinėsime du paprastus rūšiavimo algoritmus, kuriuos naudojame, kai rūšiuojame nedaug elementų.

Išnagrinėsime du paprastus rūšiavimo algoritmus, kuriuos naudojame, kai rūšiuojame nedaug elementų.

Jie yra labai naudingi ir mokantis įvertinti algoritmų sudėtingumą.

## Išrinkimo algoritmas

Išrinkimo algoritmą (angl. *selection sort*) sudaro $(N - 1)$ žingsnis.

## Išrinkimo algoritmas

Išrinkimo algoritmą (angl. *selection sort*) sudaro $(N-1)$ žingsnis.

Jo $i$-tuoju žingsniu randame mažiausią elementą tarp aibės $i, i+1, \dots, N$ elementų. Tarkime, kad tai yra $k$-tasis elementas. Jei $k \neq i$, tai šiuos elementus sukeičiame vietomis.

## Išrinkimo rūšiavimo algoritmas

```
SelectionSort ()
begin
  (1)  for  ( i = 1;  i < N ;  i++ ) do
  (2)       k = i;
  (3)       for  ( j = i+1;  j <= N;  j++ ) do
  (4)            if  ( a_j < a_k )  k = j;
                 end do
  (5)       if  ( k > i )  swap ( a_i, a_k );
       end do
end SelectionSort
```

| 101 | 17 | 33 | 2 | 24 |
|---|---|---|---|---|

a)

| **2** | 17 | 33 | 101 | 24 |
|---|---|---|---|---|

b) $i = 1,\ k = 4$

| **2** | **17** | 33 | 101 | 24 |
|---|---|---|---|---|

c) $i = 2,\ k = 2$

| **2** | **17** | **24** | 101 | 33 |
|---|---|---|---|---|

d) $i = 3,\ k = 5$

| **2** | **17** | **24** | **33** | **101** |
|---|---|---|---|---|

e) $i = 4,\ k = 5$

Skaičių masyvo rūšiavimas išrinkimo algoritmu: $i$ – ciklo žingsnis, $k$ – mažiausiojo elemento indeksas, $a_i$ ir $a_k$ elementai sukeičiami vietomis

## Algoritmo sudėtingumo įvertinimas

Vykdydami rūšiavimo algoritmus, atliekame dvi pagrindines operacijas:

lyginame du elementus,

elementus sukeičiame vietomis.

Pažymėkime $L(N)$ elementų lyginimų skaičių, o $S(N)$ – elementų sukeitimų skaičių, kai rūšiuojame aibę $A$, turinčią $N$ elementų.

Išrinkimo algoritme elementų palyginimo ciklus įvykdome nepriklausomai nuo elementų pradinio pasiskirstymo, todėl visais atvejais

$$L(N) = \sum_{i=1}^{N-1} (N-i) = \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2} .$$

Išrinkimo algoritme elementų palyginimo ciklus įvykdome nepriklausomai nuo elementų pradinio pasiskirstymo, todėl visais atvejais

$$L(N) = \sum_{i=1}^{N-1} (N - i) = \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}.$$

Elementų keitimo vietomis skaičius priklauso nuo pradinio duomenų pasiskirstymo. Geriausiu atveju, kai aibės elementai jau surūšiuoti, nereikia atlikti nei vieno keitimo, t. y. $S_G(N) = 0$. Blogiausiu atveju tokius keitimus reikės atlikti kiekvienu ciklo žingsniu, todėl $S_B(N) = N - 1$.

Išrinkimo algoritme elementų palyginimo ciklus įvykdome nepriklausomai nuo elementų pradinio pasiskirstymo, todėl visais atvejais

$$L(N) = \sum_{i=1}^{N-1} (N - i) = \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2} .$$

Elementų keitimo vietomis skaičius priklauso nuo pradinio duomenų pasiskirstymo. Geriausiu atveju, kai aibės elementai jau surūšiuoti, nereikia atlikti nei vieno keitimo, t. y. $S_G(N) = 0$. Blogiausiu atveju tokius keitimus reikės atlikti kiekvienu ciklo žingsniu, todėl $S_B(N) = N - 1$.

Todėl šis rūšiavimo metodas dažnai naudojamas, kai rūšiuojame nedaug elementų, kurių informaciniai laukai yra ilgi.

## Įterpimo algoritmas

Įterpimo algoritmą (angl. *insertion sort*) sudaro $(N-1)$ žingsnis. Jo $i$-tuoju žingsniu įmame $i$-tąjį elementą ir jį įterpiame tarp jau surūšiuotų pirmųjų $(i-1)$ elementų.

## Įterpimo algoritmas

Įterpimo algoritmą (angl. *insertion sort*) sudaro $(N-1)$ žingsnis. Jo $i$-tuoju žingsniu įmame $i$-tąjį elementą ir jį įterpiame tarp jau surūšiuotų pirmųjų $(i-1)$ elementų.

Tikrinimą pradedame nuo $(i-1)$-jo elemento. Jeigu $a_i < a_{i-1}$, tai šiuos elementus sukeičiame vietomis ir pereiname prie $(i-2)$-jo elemento. Šį procesą tęsiame tol, kol surandame vietą, į kurią reikia įterpti $a_i$ elementą.

## Įterpimo rūšiavimo algoritmas

```
InsertionSort ()
begin
  (1)  for  ( i = 2;  i <= N;  i++ ) do
  (2)       v = a_i;   a_0 =v;   j = i;
  (3)       while  ( v < a_{j-1} )  do
  (4)            a_j = a_{j-1};
  (5)            j = j-1;
            end do
  (6)       if  (i ≠ j)  a_j = v;
       end do
end InsertionSort
```

## Įterpimo rūšiavimo algoritmas

```
InsertionSort ()
begin
  (1)  for  ( i = 2;  i <= N;  i++ ) do
  (2)      v = a_i;   a_0 =v;   j = i;
  (3)      while  ( v < a_{j-1} )  do
  (4)           a_j = a_{j-1};
  (5)           j = j-1;
           end do
  (6)      if  (i ≠ j)  a_j = v;
       end do
end InsertionSort
```

Taikome barjero metodą – prieš pradėdami (3) ciklą pagalbiniam nuliniam masyvo elementui priskiriame $a_i$ reikšmę, taip garantuojame, kad ciklas visada baigsis teisingai ir nereikia tikrinti, ar pasiektas pirmasis masyvo elementas.

Įterpimo algoritmu surūšiuokime skaičių masyvą
$A = (101, 17, 33, 2, 24)$.

Įterpimo algoritmu surūšiuokime skaičių masyvą
$A = (101, 17, 33, 2, 24)$.

| **101** | 17 | 33 | 2 | 24 |
|---|---|---|---|---|

a)

| **17** | **101** | 33 | 2 | 24 |
|---|---|---|---|---|

b)

| **17** | **33** | **101** | 2 | 24 |
|---|---|---|---|---|

c)

| **2** | **17** | **33** | **101** | 24 |
|---|---|---|---|---|

d)

| **2** | **17** | **24** | **33** | **101** |
|---|---|---|---|---|

e)

Pilka spalva pavaizduotas elementas, kuris eiliniame algoritmo žingsnyje buvo įterptas į jau surūšiuotą seką.

## Algoritmo sudėtingumo įvertinimas

Kiekviename algoritmo (1) ciklo žingsnyje elementų sukeitimų skaičius yra vienu mažesnis, nei jų lyginimų skaičius

$$L(N) = S(N) + N - 1.$$

## Algoritmo sudėtingumo įvertinimas

Kiekviename algoritmo (1) ciklo žingsnyje elementų sukeitimų skaičius yra vienu mažesnis, nei jų lyginimų skaičius

$$L(N) = S(N) + N - 1.$$

Jeigu elementai yra išdėstyti atvirkštine tvarka, tai kiekviename (1) ciklo žingsnyje atliekame visus tikrinimus, todėl *blogiausiuoju* atveju

$$L_B(N) = \sum_{i=2}^{N} i = \frac{N^2 + N - 2}{2} = \frac{N^2}{2} + \mathcal{O}(N) \, .$$

*Vidutinis* įvertinimas priklauso nuo pradinio duomenų pasiskirstymo.

*Vidutinis* ¿vertinimas priklauso nuo pradinio duomenų pasiskirstymo.

Tarsime, kad vykdant eilin¿ (1) ciklo žingsn¿, elementas su vienoda tikmybe gali būti ¿terptas ¿ bet kurią jau surūšiuoto masyvo vietą.

*Vidutinis* įvertinimas priklauso nuo pradinio duomenų pasiskirstymo.

Tarsime, kad vykdant eilinį (1) ciklo žingsnį, elementas su vienoda tikmybe gali būti įterptas į bet kurią jau surūšiuoto masyvo vietą.

Tada nesunku suskaičiuoti, kad

$$L_V(N) = \frac{N^2}{4} + \mathcal{O}(N)\,.$$

*Vidutinis* įvertinimas priklauso nuo pradinio duomenų pasiskirstymo.

Tarsime, kad vykdant eilinį (1) ciklo žingsnį, elementas su vienoda tikmybe gali būti įterptas į bet kurią jau surūšiuoto masyvo vietą.

Tada nesunku suskaičiuoti, kad

$$L_V(N) = \frac{N^2}{4} + \mathcal{O}(N)\,.$$

Tai tik du kartus geriau už blogiausiojo atvejo veiksmų skaičių.

Jau pripratome, kad išanalizavę sudaryto algoritmo sudėtingumą, ieškome būdo, kaip galima būtų algoritmą pagerinti.

Jau pripratome, kad išanalizavę sudaryto algoritmo sudėtingumą, ieškome būdo, kaip galima būtų algoritmą pagerinti.

Tokios modifikacijos gali būti naudingos ne visiems duomenims, o tik specifinei jų daliai.

Tarkime, kad rūšiuojame duomenis, kai elementų palyginimo kaštai yra daug didesni už elementų sukeitimo vietomis kaštus.

Jau pripratome, kad išanalizavę sudaryto algoritmo sudėtingumą, ieškome būdo, kaip galima būtų algoritmą pagerinti.

Tokios modifikacijos gali būti naudingos ne visiems duomenims, o tik specifinei jų daliai.

Tarkime, kad rūšiuojame duomenis, kai elementų palyginimo kaštai yra daug didesni už elementų sukeitimo vietomis kaštus.

Tada reikiamos ¿terpimo vietos paiešką vykdysime naudodami skaldyk ir valdyk metodą.

Tarkime reikia įterpti $i$-ąjį elementą. Tada $a_i$ pirmiausia lyginame su viduriniu elementu $a_{i/2}$. Jeigu

$$a_{i/2} \leq a_i,$$

tai procesą kartojame intervale $[i/2+1, i]$, priešingu atveju tiriame intervalą $[1, i/2-1]$.

Tarkime reikia ¿terpti $i$-ąj¿ elementą. Tada $a_i$ pirmiausia lyginame su viduriniu elementu $a_{i/2}$. Jeigu

$$a_{i/2} \leq a_i,$$

tai procesą kartojame intervale $[i/2+1, i]$, priešingu atveju tiriame intervalą $[1, i/2-1]$.

Sakykime, kad suradome, jog $a_i$ elementas turi būti ¿terptas ¿ $j$-ąją vietą. Tada perstumiame elementus $a_{j+1}$, ..., $a_{i-1}$ ir patalpiname $a_i$ ¿ $j$-ąją vietą.

Tarkime reikia ¿terpti $i$-ą¿j¿ elementą. Tada $a_i$ pirmiausia lyginame su viduriniu elementu $a_{i/2}$. Jeigu

$$a_{i/2} \leq a_i,$$

tai procesą kartojame intervale $[i/2 + 1, i]$, priešingu atveju tiriame intervalą $[1, i/2 - 1]$.

Sakykime, kad suradome, jog $a_i$ elementas turi būti ¿terptas ¿ $j$-ąją vietą. Tada perstumiame elementus $a_{j+1}$, ..., $a_{i-1}$ ir patalpiname $a_i$ ¿ $j$-ąją vietą.

Tokio rūšiavimo algoritmo lyginimų skaičius yra optimalus $L_N = \mathcal{O}(N \log N)$.