DUOMENŲ STRUKTŪROS IR ALGORITMAI

Rūšiavimo algoritmai
(įterpimo, burbulo, išrinkimo)

Rūšiavimo veiksmas

Kasdieniniame gyvenime mes dažnai rūšiuojame:

- Failus kataloguose
- Katalogus lokaliame diske
- Kasdienines užduotis
- Knygas lentynose
- Muzikinius grojaraščius (*playlist*)

Rūšiavimas – viena iš dažniausiai kompiuterio atliekamų procedūrų.

Rūšiavimo uždavinys

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

Galime palyginti duomenis: $a_i \le a_j$, kai $i \; != j$.

Ir suformuoti modifikuotą masyvą $A' = (a'_1; a'_2 \ldots a'_N)$

kuriame $a'_i \le a'_{i+1}$ kur $i = 0, 1, 2 \ldots N-1$

Rūšiavimas yra stabilus, jei jis nepakeičia vienodų aibės elementų vietomis rūšiavimo metu t.y.:

Duota: (4, 2) (**3, 7**) (**3, 1**) (5, 6)
Nepakeista: (**3, 7**) (**3, 1**) (4, 2) (5, 6)
Pakeista: (**3, 1**) (**3, 7**) (4, 2) (5, 6)

Topologinis rūšiavimas

Sakykime, jog turime aibę užduočių:

$$U = (u_1, u_2, \ldots, u_N)$$

Jei ryšys tarp aibės elementų yra toks:

$$C = (u_{i_1} \prec u_{j_1},\ u_{i_2} \prec u_{j_2},\ \ldots,\ u_{i_M} \prec u_{j_M}).$$

$$u_i' \prec u_j', \qquad i < j.$$

tuomet sakome, kad turime **topologinį rūšiavimą.**

Pavyzdys

Sakykime, jog turime 9 užduotis (darbus):

$$U = \left(d_1, d_2, d_3, p_1, p_2, p_3, u_1, u_2, u_3 \right)$$

Ir ryšys tarp užduočių apibrėžiamas taip:

$$C = \left(d_j \prec p_j, \;\; p_j \prec u_j, \;\; j = 1, 2, 3 \right).$$

Tuomet išrūšiuoti užduotis galime taip:

$$U' = \left(d_1, p_1, u_1, \;\; d_2, p_2, u_2, \;\; d_3, p_3, u_3 \right).$$

$$U'' = \left(d_1, d_2, d_3, \;\; p_3, p_2, p_1, \;\; u_2, u_3, u_1 \right).$$

Rūšiavimo sudėtingumas

Rūšiavimo algoritmų sudėtingumui įvertinti dažniausiai naudojamas **elementų lyginimo veiksmų skaičius**.

Tačiau jei lyginame didelius įrašus, jų sukeitimas vietomis gali būti atliekamas ne per vieną žingsnį (skaldomas į mažesnius žingsnius).

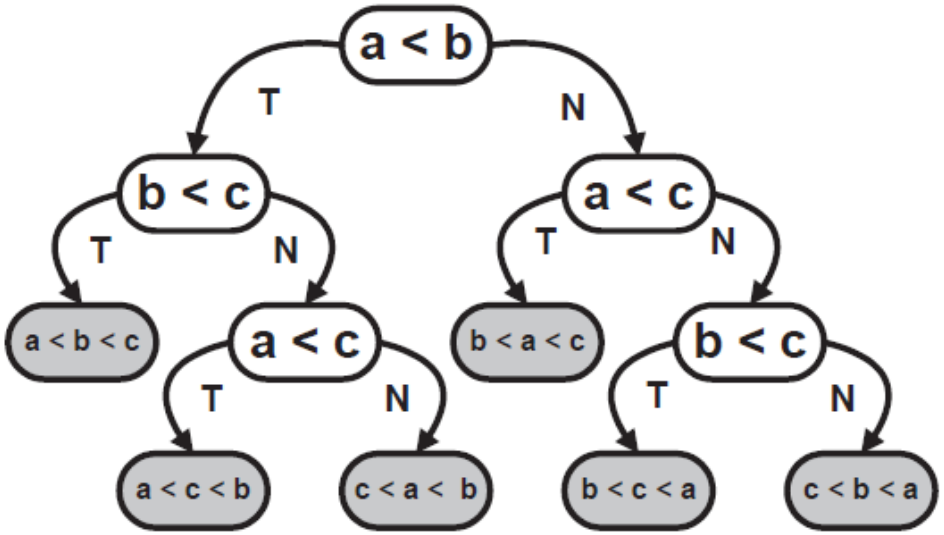
Siekiant įvesti objektyvų rūšiavimo algoritmų sudėtingumo matą, naudojamas ir **elementų sukeitimų skaičius** rūšiavimo metu.

Rūšiavimo sudėtingumas

Sakykime turime 3 elementus: a, b, c.

Rūšiavimo veiksmai gali būti užrašyti kaip dvejetainio medžio struktūra.

Medžio gylis lygus 3 ir parodo blogiausią rūšiavimo atvejį t.y. kiek daugiausiai veiksmų reiks atlikti norint surūšiuoti 3 skaičius.



Lyginimo veiksmų skaičius K randamas:

$$2^K \geq N!$$

kur N – elementų skaičius.

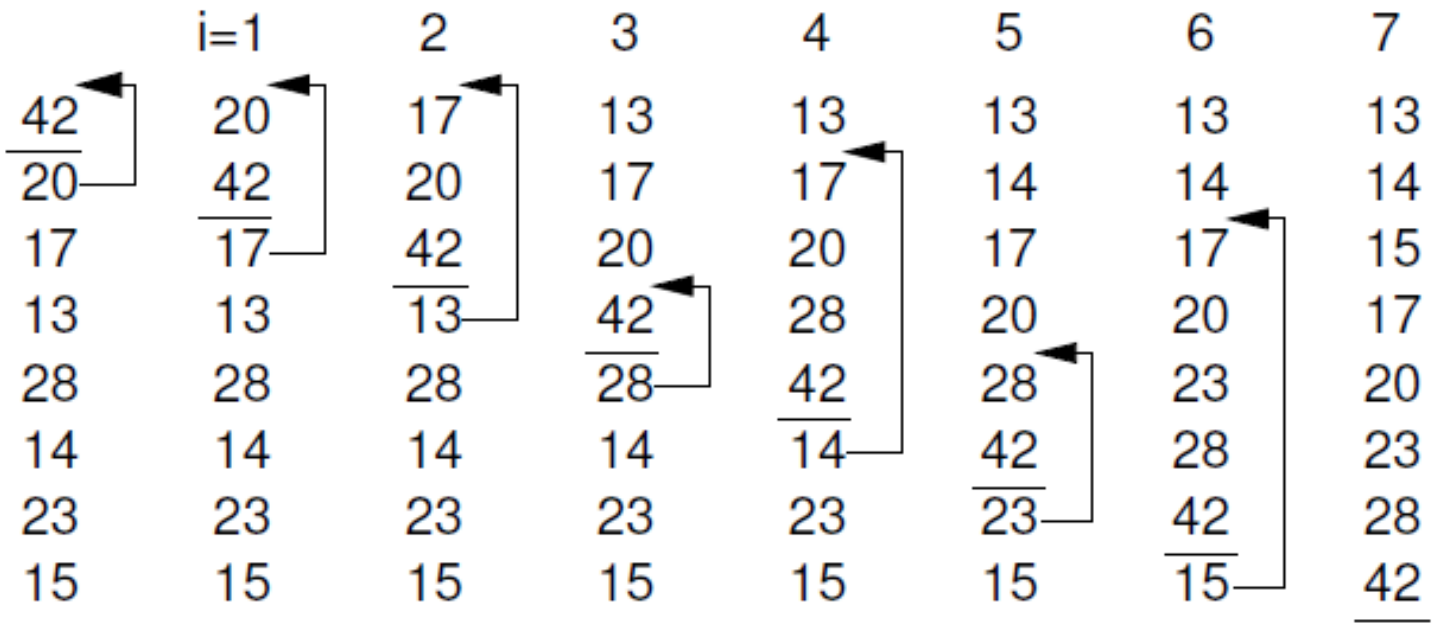
Įterpimo rūšiavimo algoritmas

Sakykime, kad turime paskutinių metų telefono sąskaitas ir norime jas išrūšiuoti pagal datą.

Galimas rūšiavimo variantas:

1. Paimame bet kurias dvi sąskaitas, palyginame jų datas ir sudedame reikiama tvarka.
2. Paimame trečią sąskaitą ir padedame tarp dviejų išrūšiuotų į reikiamą vietą.
3. Tęsiame analogiškus veiksmus su kitomis sąskaitomis, kol gauname išrūšiuotų sąskaitų eilę.

Įterpimo rūšiavimo algoritmas



Įterpimo algoritmo realizacija

```
void insertion_Sort (int masyvas [ ], int dydis) {
    int i, j, tmp;
    for (i = 1;  i < dydis;  i++) {
        j = i;
        while ( j > 0  &&  masyvas [ j – 1 ]  > masyvas [ j ]) {
            tmp = masyvas [ j ];
            masyvas [ j ] = masyvas [ j – 1 ];
            masyvas [ j – 1 ] = tmp;
            j - -;
        }
    }
}
```

Įterpimo algoritmo sudėtingumas

Algoritmo sudėtingumo įvertinimas.
Kiekviename įterpimo algoritmo ciklo žingsnyje atliekame elementų lyginimo veiksmų vienu mažiau nei keitimo vietomis. Bendras veiksmų skaičius tokiu atveju lygus:

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

Geriausiu atveju, kai pradinė elementų aibė jau yra surūšiuota, gauname, kad lyginimų skaičius yra lygus L(N) = N-1. Jei elementai yra išdėstyti atvirkštine tvarka, tai kiekviename ciklo žingsnyje atliekame visus tikrinimo veiksmus, todėl blogiausiu atveju gauname:

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

Vidutiniu atveju, lyginimo veiksmų skaičius lygus:

$$L_V(N) = \sum_{i=2}^{N} \frac{i+1}{2} = \frac{N^2 + 3N + 4)}{4} = \frac{N^2}{4} + \mathcal{O}(N)\,.$$

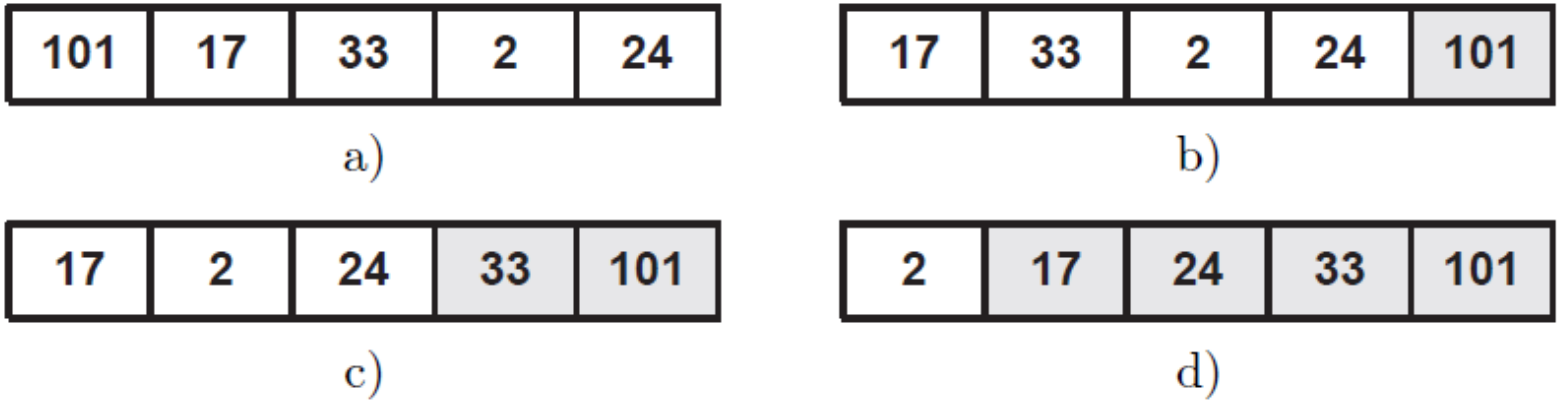
Burbulo rūšiavimo algoritmas

Tai sąlyginai lėtas, bet stabilus rūšiavimo algoritmas. Rūšiavimo procedūra artima oro burbulo kilimui vandenyje.

Rūšiavimo uždavinį sprendžiame iš eilės lygindami du gretimus aibės A elementus (a1; a2); (a2; a3)...
Jeigu $a_i > a_{i+1}$, tai tokius elementus sukeičiame vietomis.

Nesunku pastebėti, kad po pirmojo visų elementų patikrinimo, didžiausias sekos elementas užims paskutiniojo elemento vietą. Todėl antrajame cikle užtenka tikrinti trumpensę duomenų seką tik iki (N - 1)-ojo elemento. Procesą kartojame tol, kol eilinio ciklo metu nei karto nepakeičiame elementų vietomis. Kadangi po kiekvieno ciklo seka sutrumpėja vienu elementu, tai blogiausiu atveju algoritmą teks kartoti (N-1) kartą.

Pavyzdys



Bubble sort

```
void Bubble_Sort(int array[ ], int length) {
{
     int i, j, flag = 1;          // sukeitimų indikatorius flag
     int temp;
     for (i = 0; (i < length -1 ) && flag; i++)
    {    flag = 0;
           for ( j = 0; j < (length - i -1); j++)
           {      if ( array [j+1] > array[j])
                 {     temp  = array [j];            // elementų sukeitimas
                       array [j] = array [j+1];
                       array [j+1] = temp;
                        flag = 1;              // parodo, kad atliktas sukeitimas
                 }
           }
    }
    return;
}
```

Burbulo algoritmo sudėtingumas

Geriausiu atveju, kai pradinis elementų masyvas jau surūšiuotas, užteks tik vieną kartą įvykdyti burbulo algoritmo ciklą.

$$L_G(N) = N - 1, \quad S_G(N) = 0\,.$$

Blogiausiu atveju, kai pradiniame masyve elementai išdėstyti mažėjančia tvarka, reikės atlikti N-1 algoritmo žingsnį, todėl tokio algoritmo vykdymo sąnaudos bus:

$$L_B(N) = \frac{N(N-1)}{2}, \quad S_G(N) = \frac{N(N-1)}{2}\,.$$

Vidutiniu atveju, rūšiavimo algoritmo sudėtingumas bus:

$$L_V(N) = \frac{N(N - \log N)}{2} + \mathcal{O}(N)\,,$$

Išrinkimo rūšiavimo algoritmas

Sakykime, kad turime paskutinių metų telefono sąskaitas ir norime jas išrūšiuoti pagal datą.

Galimas rūšiavimo variantas:

1. Surandame sausio mėn. sąskaitą ir ją atidedame.
2. Surandame vasario mėn. sąskaitą ir ją dedame prie sausio mėn.
3. Ciklą kartojame, kol visos sąskaitos būna surūšiuotos.

Išrinkimo algoritmas turi privalumų tuomet, kai sukeitimo veiksmas yra brangus, pvz. aibės elementai yra eilutės arba dideli įrašai.

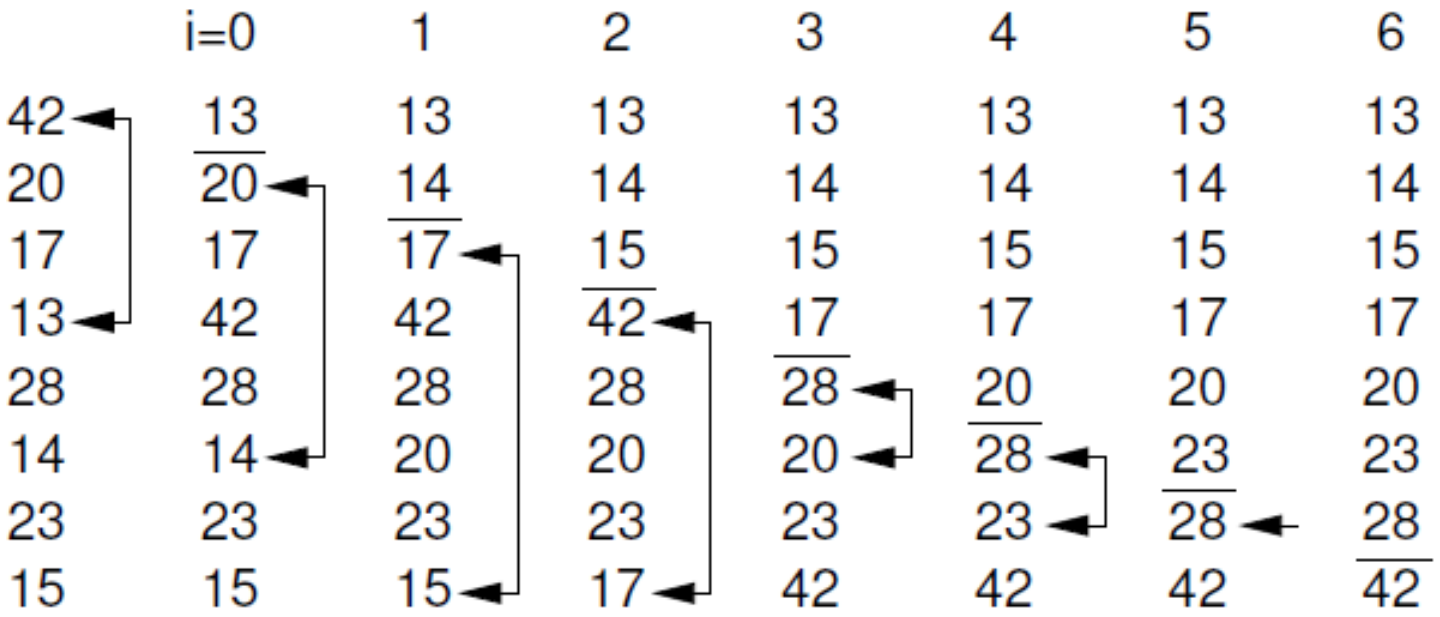
Išrinkimo rūšiavimo algoritmas

Praktiškai išrinkimo algoritmas realizuojamas taip:

1. Kiekviename žingsnyje ***i*** randamas mažiausias elementas (tiksliau elementas, kurio reikšmė – dar vadinama raktu yra mažiausia) tarp aibės i; .... N elementų.
2. Tarkime, kad tai yra *p* elementas. Jei *i* >= *p*, tai šiuos elementus sukeičiame vietomis.

Išrinkimo algoritmą sudaro *N - 1* žingsnis.

Išrinkimo rūšiavimo algoritmas



Išrinkimo rūšiavimo algoritmas

```
void Selection_Sort ( int *array,  int iLength)
{
    if (iLength <= 1)
    return;
  for ( int i = 0; i < iLength – 1; i++)
  { int iSmallest = i;
      for ( int  j = i+1; j < iLength; j++ )
    {    if (array[iSmallest]  > array[j])
           {  iSmallest = j;
            }
    }
    int tmp  = array[iSmallest];
    array[iSmallest] = array[i];
    array[i] = tmp;
  }
}
```

Išrinkimo algoritmo sudėtingumas

Išrinkimo rūšiavimo algoritme vykdomi du ciklai. Jie vykdomi nepriklausomai nuo elementų pradinio pasiskirstymo, todėl visais atvejais lyginimo veiksmų skaičius bus toks:

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

Elementų keitimo vietomis skaičius priklauso nuo pradinio duomenų pasiskirstymo. Geriausiu atveju, kai aibės elementai jau surūšiuoti, nereikia atlikti nei vieno keitimo:

$$S_G(N) = 0.$$

Blogiausiu atveju, tokius keitimus reiks atlikti kiekviename ciklo žingsnyje, todėl sukeitimų skaičius bus lygus:

$$S_B(N) = N - 1.$$