

3.3. Greitieji rūšiavimo algoritmai

Ankstesniame poskyryje nagrinėjome paprastus rūšiavimo algoritmus ir parodėme, kad atliekamų operacijų skaičius yra proporcingas N^2 . Jie nėra efektyvūs, nes rūšiavimo algoritmų apatinis operacijų įvertis yra $N \log N$. Šiame poskyryje susipažinsime su greitaisiais rūšiavimo algoritmais, kurių sudėtingumas yra artimas optimaliam.

3.3.1. Spartusis rūšiavimo algoritmas

Šis algoritmas labai plačiai naudojamas daugelyje taikomųjų programų. Parodysime, kad jo vidutinis sudėtingumas yra $\mathcal{O}(N \log N)$, o realizacija yra paprasta ir nereikia papildomos atminties rūšiuojamų elementų saugojimui. Todėl jis ir vadinamas *sparčiuoju* algoritmu (angl. *quick sort*).

Spartusis rūšiavimo algoritmas yra sukonstruotas remiantis *skaldyk ir valdyk* metodu. Paaiškinsime, kaip realizuojami trys pagrindiniai jo etapai.

Uždavinio skaidymas. Visą rūšiuojamą aibę dalijame į du poaibius. Tuo tikslu parenkame *pagrindinį* elementą a_j (angl. *pivot*), tada pirmajam poaibiui priskiriame elementus mažesnius už a_j , o antrajam poaibiui – ne mažesnius už pagrindinį elementą.

Dalinio uždavinio sprendimas. Jei poaibį sudaro vienas elementas, tai jis jau surūšiuota, priešingu atveju ir poaibį rūšiuojame sparčiuoju algoritmu. Dažnai rekursiją užbaigiame anksčiau, pasirenkame nedidelį skaičių M ir jei poaibio elementų skaičius yra nedidesnis už M , tai poaibį surūšiuojame kuriuo nors paprastu algoritmu.

Viso uždavinio sprendinio radimas. Kadangi visi pirmojo poaibio elementai yra mažesni už antrojo poaibio elementus, tai išsprendę dalinius uždavinius mes surūšiuojame ir visą aibę. Taigi šiame etape nereikia atlikti jokių papildomų veiksmų.

Spartusis rūšiavimo algoritmas

```

QuickSort (l, r)
begin
  (1) if ( l < (r - M) ) {
    (2) Partition (l, r, k );
    (3) QuickSort ( l, k-1 );
    (4) QuickSort ( k, r );
  } else
    (5) if ( l < r ) SelectionSort ( l, r );
end QuickSort

```

Paaškinsime algoritme panaudotus žymėjimus. Funkcijos *QuickSort* argumentai l ir r parodo, kad rūšiuojama nuo l -ojo iki r -ojo pradinio masyvo elemento. Skaidymo procedūroje *Partition* aibę padaliname į du atskirtus poaibius, pirmajam priklauso masyvo elementai nuo l -ojo iki $(k - 1)$ -ojo, o antrajam – nuo k -ojo iki r -ojo. Kai aibė yra nedidelė, jos elementus rūšiuojame paprastu įterpimo algoritmu *SelectionSort*.

Beveik visi veiksmai yra atliekami vykdant aibės skaidymą į du poaibius, todėl reikia stengtis, kad šios funkcijos realizavimas būtų efektyvus. Pateiksime tokį algoritmą.

Skaidymo algoritmas

```

Partition (l, r, i )
begin
  (1) v = Pivot (l, r);
  (2) i = l;    j = r;
  (3) while ( i ≤ j ) {
    (4) if ( i == j )
      (5) if ( ai < v ) i = i + 1;
      (6) else    j = j - 1;
    (7) else
      (8) while ( ai < v ) i = i + 1;
      (9) while ( ( i < j ) && aj ≥ v ) j = j - 1;
      (10) if ( i == j ) j = j-1;
      (11) else swap ( ai, aj ); i = i + 1;    j = j - 1;
    }
  (12) if ( i == l ) i = i+1;
end Partition

```

Procedūroje *Pivot* parenkame pagrindinį elementą.

3.6 pavyzdys. Skaičių masyvo rūšiavimas. Sparčiuoju algoritmu surūšiuokime masyvą $E = (11, 10, 16, 8, 19, 37, 9, 22, 19, 11)$.

Algoritmo sudėtingumo įvertinimas. Nagrinėsime aibės A elementų lyginimų skaičių L_N . Uždavinio skaidymo metu visus elementus palyginame su pagrindiniu elementu. Todėl bendras lyginimų skaičius priklauso tik nuo dalinių uždavinių apimties.

Analizę pradėkime nuo *blogiausio* atvejo, kada, pavyzdžiui, pagrindiniu elementu parenkame mažiausią aibės elementą. Tada gauname tokį sąryšį

$$L_B(N) = L_B(N - 1) + N.$$

Tarsime, kad rekursiją vykdome iki tol, kol $N > 2$:

$$L(1) = 0, \quad L(2) = 1.$$

Pritaikę rekurentinę lygybę $N - 2$ kartus, apskaičiuojame elementų lyginimų skaičių

$$L_B(N) = \sum_{i=3}^N i + 1 = \frac{N^2 + N - 4}{2}.$$

Taigi blogiausiu atveju spartusis rūšiavimo algoritmas yra toks pat lėtas, kaip ir mūsų anksčiau išnagrinėti metodai. Ypač netikėta yra tai, kad tokį blogą rezultatą gauname, pavyzdžiui, kai rūšiuojame jau sutvarkytą aibę, o pagrindiniu aibės elementu renkamės pirmąjį elementą.

Dabar nagrinėkime *geriausią* atvejį, kai kiekviename žingsnyje pavyksta aibę padalinti į dvi lygias dalis. Kad skaičiavimo rezultatai būtų paprastesni, imkime $N = 2^m$. Tada gauname tokį sąryšį, susiejantį elementų lyginimų skaičius:

$$L_G(2^m) = \begin{cases} 2L_G(2^{m-1}) + 2^m, & \text{kai } m > 1, \\ 1, & \text{kai } m = 1. \end{cases}$$

Pritaikę šią lygybę $m - 1$ kartą, apskaičiuojame elementų lyginimų skaičių

$$\begin{aligned} L_G(N) &= 2^m + 2 \cdot 2^{m-1} + 2^2 \cdot 2^{m-2} + \dots + 2^{m-2} \cdot 2^2 + 2^{m-1} \\ &= (m - 1)2^m + 2^{m-1} = N \log N - \frac{N}{2}. \end{aligned}$$

Taigi geriausiu atveju sparčiojo rūšiavimo algoritmo skaičiavimų apimtis yra optimali.

Spartusis rūšiavimo algoritmas tapo tokiu populiariu todėl, kad ir *vidutiniu* atveju skaičiavimų apimtis nedaug skiriasi nuo geriausio atvejo. Jeigu turime N elementų, tai viso gali būti $N!$ skirtingų jų pasiskirstymų. Tarkime, kad jie visi yra vienodai tikėtini. Skaičiuodami vidutinį elementų lyginimų skaičių, atsižvelgiame į tai, kad uždavinio skaidymo etape atliekame $N - 1$ lyginimą, o po to su vienoda tikimybe gali tekti rūšiuoti $N - 1$ skirtingus dalinius poaibius, todėl

$$L_V(N) = N + \frac{1}{N-1} \left((L_V(1) + L_V(N-1)) + (L_V(2) + L_V(N-2)) \right. \\ \left. + \dots + (L_V(N-1) + L_V(1)) \right).$$

Šią lygtį galime suprastinti:

$$L_V(N) = N + \frac{2}{N-1} \left(L_V(1) + L_V(2) + \dots + L_V(N-1) \right).$$

Imdami $N - 1$ elementą turime lygtį

$$L_V(N-1) = N-1 + \frac{2}{N-2} \left(L_V(1) + L_V(2) + \dots + L_V(N-2) \right).$$

Iš šių dviejų lygybių gauname tiesinę skirtumų lygtį

$$(N-1)L_V(N) = NL_V(N-1) + 2(N-1),$$

kurią galime užrašyti ir simetrine forma:

$$\frac{L_V(N)}{N} = \frac{L_V(N-1)}{N-1} + \frac{2}{N}.$$

Užrašę tokias lygtis visoms N reikšmėms ir jas sudėję, bei atsižvelgę į tai, kad $L_V(1) = 0$, gauname lygybę

$$\frac{L_V(N)}{N} = 2 \left(\frac{1}{N} + \frac{1}{N-1} + \dots + \frac{1}{2} + 1 \right) - 2.$$

Nesunku įrodyti, kad harmoninės eilutės dalinė suma tenkina nelygybę:

$$1 + \frac{1}{2} + \dots + \frac{1}{N} \leq \ln N + 1,$$

todėl

$$L_V(N) = 2N \ln N + \mathcal{O}(1).$$

Atsižvelgę į lygybę

$$\ln N = 0.693 \log N,$$

gauname sparčiojo rūšiavimo algoritmo vidutinį elementų lyginimų skaičių

$$L_V(N) = 1.386N \log N + \mathcal{O}(1),$$

kuris rodo, kad vidutinis atvejis yra tik 40 procentų blogesnis už geriausią atvejį.

3.3.2. Medianos radimas sparčiuoju algoritmu

Statistinėje analizėje dažnai tenka spręsti tokius uždavinius

- iš duotosios skaičių sekos išrinkti medianą;
- rasti k -ąjį mažiausią skaičių.

Pastebėsime, kad medianos radimas yra bendresnio antrojo uždavinio atskiras atvejis, kai $k = \frac{N}{2}$.

Aišku, šiuos uždavinius lengvai išsprendžiame, kai jau turime surūšiuotą aibę. Sukonstruosime kitą algoritmą, kurio skaičiavimų apimtis daug mažesnė už rūšiavimo algoritmų kaštus. Vėl remsimės skaldyk ir valdyk metodu, reikiamo elemento išrinkimo algoritmą sudarysime modifikuodami spartųjį rūšiavimo algoritmą.

Tarsime, kad išpildytos sąlygos $l = 1 \leq k \leq r$, apibrėžiančios pradinės masyvo indeksų reikšmes.

Sparčiosios paieškos algoritmas

```
int QuickFind (l, r, k)
begin
  (1) if ( l == r ) return (l);
  (2) else {
    (3) Partition (l, r, j);
    (4) if ( k < j ) QuickFind ( l, j-1, k );
    (5) else QuickFind ( j, r, k );
  }
end QuickFind
```

Pastebėsime, kad šio algoritmo kiekviename cikle yra realizuojamas tik vienas iš dviejų galimų variantų, todėl faktiškai rekursija čia nebūtina.

Paieškos algoritmo sudėtingumas tiriamas panašiai kaip ir sparčiojo rūšiavimo algoritmo sudėtingumas. Todėl apsiribosime tik *geriausio* atveju analize. Tada po kiekvieno dalijimo žingsnio dalinio uždavinio elementų skaičius sumažėja dvigubai, todėl gauname tokį elementų lyginimo skaičių

$$L_G(N) = N + \frac{N}{2} + \frac{N}{4} + \dots + 4 + 1 = 2N + \mathcal{O}(1).$$

Taigi naujasis medianos paieškos algoritmas $\frac{1}{2} \log N$ kartų spartesnis už rūšiavimo algoritmą.

3.3.3. Suliejimo rūšiavimo algoritmas

Spartusis rūšiavimo algoritmas yra labai efektyvus vidutiniu atveju, tačiau blogiausiu atveju jo skaičiavimo apimtis yra $\mathcal{O}(N^2)$, t.y. jo greitis toks pat kaip ir paprastų rūšiavimo algoritmų. Šiame poskyryje susipažinsime su dar vienu greituoju algoritmu, kurio skaičiavimo apimtis net ir blogiausiu atveju yra $\mathcal{O}(N \log N)$.

Šis algoritmas remiasi pastebėjimu, kad galima efektyviai sujungti du surūšiuotus poaibius į vieną surūšiuotą aibę. Todėl metodas ir vadinamas *suliejimo* algoritmu (angl. *merge sort*). Pastebėsime, kad ir suliejimo rūšiavimo algoritmas yra sukonstruotas remiantis skaldyk ir valdyk metodu. Taigi naudodami vieną bendrą metodą galime gauti keletą skirtingų to pačio uždavinio sprendimo algoritmų.

Paaškinsime, kaip realizuojami trys pagrindiniai jo etapai.

Uždavinio skaidymas. Visą rūšiuojamą aibę dalijame į du poaibius. Pirmajam poaibiui priskiriame elementus nuo pirmojo iki viduriniojo, o antrajam poaibiui – visus likusius elementus. Taigi šiame etape nereikia atlikti jokių skaičiavimo veiksmų.

Dalinio uždavinio sprendimas. Jei poaibį sudaro vienas elementas, tai jis jau surūšiuotas, priešingu atveju ir poaibį rūšiuojame suliejimo algoritmu. Dažnai rekursiją užbaigiame anksčiau, pasirenkame nedidelį skaičių M ir jei poaibio elementų skaičius yra nedidesnis už M , tai poaibį surūšiuojame kuriuo nors paprastu algoritmu. Rekomenduojama imti $M \approx 16$.

Viso uždavinio sprendinio radimas. Abiejų poaibių elementai jau surūšiuoti. Įmame pirmuosius poaibių elementus ir juos palyginame, mažesnę talpiname į surūšiuotą aibę. Tarkime šis elementas priklausė pirmajam

poaibiui. Toliau procesą kartojame lygindami likusį elementą su sekančiu pirmojo poaibio elementu.

Suliejimo rūšiavimo algoritmas

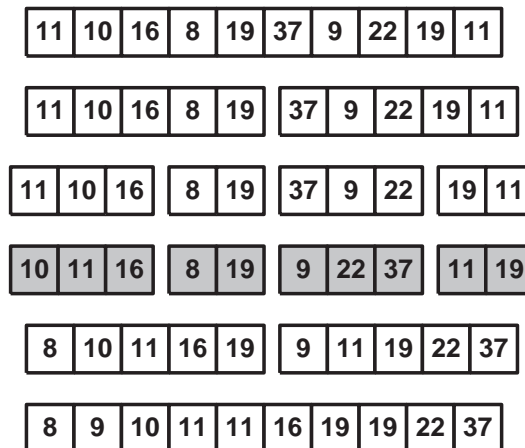
```

MergeSort (l, r)
begin
  (1) if ( l < (r - M) ) {
    (2) m = (l + r)/2 ;
    (3) MergeSort ( l, m );
    (4) MergeSort ( m+1, r );
    (5) Merge ( l, m, r );
  } else
    (6) if ( l < r ) SelectionSort ( l, r );
end MergeSort

```

Beveik visi veiksmai yra atliekami vykdant dviejų poaibių sujungimą į vieną surūšiuotą aibę, todėl reikia stengtis, kad šios funkcijos realizavimas būtų efektyvus.

3.7 pavyzdys. Skaičių masyvo rūšiavimas. Suliejimo algoritmu surūšiuokime skaičių masyvą $E = (11, 10, 16, 8, 19, 37, 9, 22, 19, 11)$. Rūšiavimo eiga pavaizduota 3.5 brėžinyje.



3.5 pav. Skaičių masyvo rūšiavimas suliejimo algoritmu, pilka spalva pavaizduotas elementų rūšiavimas išrinkimo algoritmu

Pateiksime suliejimo algoritmą, kai duomenys saugomi masyve a . Realizuodami algoritmą naudojame pagalbinį masyvą b , o suliejimo procedūros pabaigoje kopijuojame visus masyvo b elementus į pradinį masyvą a (tai atlieka procedūra *Copy*). Jei papildomo masyvo išskirti negalime, tai duomenis patogiau saugoti tiesiniame sąrašė.

Suliejimo algoritmas

```

Merge ( l, m, r )
begin
(1) i = l;    j = m+1;    k = l;
(2) while ( k ≤ r ) {
(3)   if ( ( i ≤ m ) && ( j ≤ r ) ) {
(4)     if ( ai ≤ aj )
(5)       bk = ai;    i++;    k++;
(6)   } else {
(7)     bk = aj;    j++;    k++;
(8)   }
(9)   if ( i ≤ m )
(10)    while ( k ≤ r ) {
(11)     bk = ai;    i++;    k++;
(12)   }
(13)  else
(14)   while ( k ≤ r ) {
(15)    bk = aj;    j++;    k++;
(16)   }
(17) }
(18) Copy ( b, a, l, r );
end Merge

```

Algoritmo sudėtingumo įvertinimas. Suliejimo rūšiavimo algoritme elementų lyginimų skaičius mažai priklauso nuo rūšiuojamų duomenų pradinio pasiskirstymo. Tarsime, kad dviejų surūšiuotų N_1 ir N_2 ilgio poaibių sujungimo metu atliekame $c(N_1 + N_2)$ elementų lyginimų. Kad analizė būtų paprastesnė, imkime $N = 2^m$. Tada gauname tokį sąryšį, susiejantį elementų lyginimų skaičius:

$$L(N) = \begin{cases} 2L(\frac{1}{2}N) + cN, & \text{kai } N > 2, \\ 1, & \text{kai } N = 2. \end{cases}$$

Pritaikę šią lygybę $m - 1$ kartą, apskaičiuojame elementų lyginimų skaičių

$$\begin{aligned} L(N) &= 2L\left(\frac{1}{2}N\right) + cN = 4L\left(\frac{1}{4}N\right) + 2cN = \dots = \frac{N}{2}L(2) + (m - 1)cN \\ &= cN \log N + (0.5 - c)N. \end{aligned}$$

Taigi suliejimo rūšiavimo algoritmo skaičiavimų apimtis yra optimali ir blogiausiu atveju.

3.3.4. Piramidinis rūšiavimo algoritmas

Šiame poskyryje susipažinsime su dar vienu greituoju rūšiavimo algoritmu, kuris naudoja vieną svarbią duomenų struktūrą – *piramidę*. Pasinaudosime svarbia piramidės savybe, kad didžiausias jos elementas yra medžio šaknyje.

```

HeapSort ()
begin
  (1) MakeHeap ();
  (2) for ( i=N; i > 1; i = i-1 ) {
        (3) swap (a1, ai);
        (4) HeapDownOrder (1, i-1);
      }
end

```

3.8 pavyzdys. Skaičių masyvo rūšiavimas. Piramidės algoritmu surūšiuosime ankstesniame pavyzdyje pateiktą skaičių masyvą. Rūšiavimo eiga pavaizduota 3.6 brėžinyje.

Algoritmo sudėtingumo įvertinimas. Jau įvertinome aibės elementų pertvarkymo į piramidę skaičiavimo apimtį. Atlikdami (2) ciklo visus žingsnius blogiausiu atveju papildomai $2N \log N$ kartų lyginsime elementus ir $N \log N$ kartų juos sukeisime vietomis, todėl pirmadės rūšiavimo algoritmo sudėtingumas yra

$$L(N) = 3N \log N, \quad S(N) = \frac{3}{2}N \log N.$$

Taigi sudarėme dar vieną greitojo rūšiavimo algoritmą.

| | | | | | | | | | |
|----|----|----|----|----|----|----|---|----|----|
| 10 | 37 | 18 | 13 | 22 | 14 | 25 | 8 | 12 | 28 |
|----|----|----|----|----|----|----|---|----|----|

| | | | | | | | | | |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|----------|-----------|-----------|
| 37 | 28 | 25 | 13 | 22 | 14 | 18 | 8 | 12 | 10 |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|----------|-----------|-----------|

a)

| | | | | | | | | | |
|----|----|----|----|----|----|----|---|----|-----------|
| 10 | 28 | 25 | 13 | 22 | 14 | 18 | 8 | 12 | 37 |
|----|----|----|----|----|----|----|---|----|-----------|

| | | | | | | | | | |
|-----------|-----------|----|----|-----------|----|----|---|----|-----------|
| 28 | 22 | 25 | 13 | 10 | 14 | 18 | 8 | 12 | 37 |
|-----------|-----------|----|----|-----------|----|----|---|----|-----------|

| | | | | | | | | | |
|----|----|----|----|----|----|----|---|-----------|-----------|
| 12 | 22 | 25 | 13 | 10 | 14 | 18 | 8 | 28 | 37 |
|----|----|----|----|----|----|----|---|-----------|-----------|

| | | | | | | | | | |
|-----------|-----------|-----------|----|----|----|-----------|---|-----------|-----------|
| 25 | 22 | 18 | 13 | 10 | 14 | 12 | 8 | 28 | 37 |
|-----------|-----------|-----------|----|----|----|-----------|---|-----------|-----------|

| | | | | | | | | | |
|---|----|----|----|----|----|----|-----------|-----------|-----------|
| 8 | 22 | 18 | 13 | 10 | 14 | 12 | 25 | 28 | 37 |
|---|----|----|----|----|----|----|-----------|-----------|-----------|

| | | | | | | | | | |
|-----------|-----------|-----------|----------|----|----|----|-----------|-----------|-----------|
| 22 | 13 | 18 | 8 | 10 | 14 | 12 | 25 | 28 | 37 |
|-----------|-----------|-----------|----------|----|----|----|-----------|-----------|-----------|

| | | | | | | | | | |
|----|----|----|---|----|----|-----------|-----------|-----------|-----------|
| 12 | 13 | 18 | 8 | 10 | 14 | 22 | 25 | 28 | 37 |
|----|----|----|---|----|----|-----------|-----------|-----------|-----------|

| | | | | | | | | | |
|-----------|-----------|-----------|---|----|----|-----------|-----------|-----------|-----------|
| 18 | 13 | 14 | 8 | 10 | 12 | 22 | 25 | 28 | 37 |
|-----------|-----------|-----------|---|----|----|-----------|-----------|-----------|-----------|

b)

3.6 pav. Skaičių masyvo rūšiavimas piramidės algoritmu: a) pradinis masyvas ir suformuota piramidė, b) pirmieji keturi rūšiavimo žingsniai, elementai a_7, a_8, a_9, a_{10} jau surūšiuoti