1 skyrius

Rūšiavimo algoritmai

1.1. 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.

1.1.1. 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 – nemažesnius už pagrindinį elementą.

Dalinio uždavinio sprendimas. Jei poaibį sudaro vienas elementas, tai jis jau surūšiuota, priešingu atveju ir poaibį rūšiuojame sparčiuoju algorit-

mu. 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
```

Paaiškinsime 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   ( a_i < v ) i = i + 1;
                   (6)  else   j = j - 1;
            (7)  else
                   (8)  while  ( a_i < v ) i = i + 1;
                   (9)  while  ( (i < j) && a_j ⩾ v ) j = j - 1;
                   (10)  if  ( i == j ) j = j-1;
                   (11)  else  swap ( a_i, a_j );  i = i + 1;   j = j - 1;
      }
 (12)  if   ( i == l )  i = i+1;
end Partition
```

Procedūroje *Pivot* parenkame pagrindinį elementą.

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

Pateiksime tik vieno rekursijos žingsnio analizę. Pagrindiniu elementu pasirenkame pirmąjį masyvo elementą $v = 11$. Pradinės indeksų reikšmės yra $i = 1$, $j = 10$. Po 8 žingsnio ciklo indekso i reikšmė nepasikeitė $i = 1$, nes $a_1 = 11$. Tada vykdydami 9 žingsnio ciklą surandame, kad $a_7 < 11$, todėl indekso j reikšmė sumažėjo iki $j = 7$. Kadangi paieška iš kairės ir dešinės pusės surado skirtingus elementus (t.y. $i \neq j$), tai sukeičiame šiuos elementus vietomis

$$E = (9, 10, 16, 8, 19, 37, 11, 22, 19, 11)$$

ir perstumiame indeksus $i = 2$, $j = 6$. Tada vėl vykdome algoritmo 3 žingsnį. Dabar gauname, kad kritiniai elementai atitinka indeksus $i = 3$, $j = 4$. Vėl sukeičiame šiuos elementus vietomis

$$E = (9, 10, 8, 16, 19, 37, 11, 22, 19, 11)$$

ir perstumiame indeksus $i = 4$, $j = 3$. Kadangi dabar $i > j$, tai masyvas jau surūšiuotas pagrindinio elemento $v = 11$ atžvilgiu, pradedant elementu a_4 visi masyvo elementai $a_j \geq 11$, kai $j \geq 4$. Gavome du naujus uždavinius – surūšiuoti masyvus

$$E_L = (9, 10, 8), \quad E_R = (16, 19, 37, 11, 22, 19, 11).$$

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 $N > 2$:

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

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

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

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

$$\begin{aligned} L_V(N) = N + \frac{1}{N-1}\Big(&\big(L_V(1) + L_V(N-1)\big) + \big(L_V(2) + L_V(N-2)\big) \\ &+ \ldots + \big(L_V(N-1) + L_V(1)\big)\Big)\,. \end{aligned}$$

Šią lygtį galime suprastinti:

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

Imdami $N-1$ elementą turime lygtį

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

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

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

Nesunku įrodyti, kad harmoninės eilutės dalinė suma tenkina nelygybę:

$$1 + \frac{1}{2} + \ldots + \frac{1}{N} \leqslant \ln N + 1\,,$$

todėl

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

Atsižvelgę į lygybę

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

gauname sparčiojo rūšiavimo algoritmo vidutinį elementų lyginimų skaičių

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

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

1.1.2. 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 algorimų kaštus. Vėl remsimės skaldyk ir valdyk metodu, reikiamo elemento išrinkimo algoritmą sudarysime modifikuodami spartųjį rūšiavimo algoritmą.

Tarsime, kad išpildytos sąlygos $l = 1 \leqslant k \leqslant r$, apibrėžiančios pradines 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 varijantų, todėl faktiškai rekursija čia nebūtina.

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

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

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

1.1.3. 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.

Paaiškinsime, 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 $M \approx 16$.

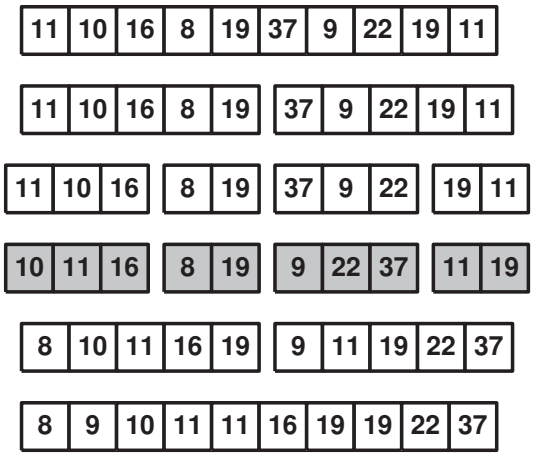
Viso uždavinio sprendinio radimas. Abiejų poaibių elementai jau surūšiuoti. Įmame pirmuosius poaibių elementus ir juos palyginame, mažesnį talpiname į surūšiuotą aibę. Tarkime šis elementas priklausė pirmajam poaibiui. Toliau procesą kartojame lygindami likusį elementą su sekančiu pirmojo poaibio elementu.

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.

1.2 pavyzdys. Skaičių masyvo rūšiavimas. Suliejimo algoritmu surūšiuokime skaičių masyvą $E = (11, 10, 16, 8, 19, 37, 9, 22, 19, 11)$. Rūšiavimo eiga pavaizduota 1.1 brėžinyje.



1.1 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 patogu saugoti tiesiniame sąraše.

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  ( a_i ⩽ a_j )
                            (5)  b_k = a_i;   i++;   k++;
                   (6)  }else {
                            (7)  b_k = a_j;   j++;   k++;
                        }
          (8)  else
                   (9)   if  ( i ⩽ m )
```

```
                    (10)  while  ( k ⩽ r ) {
                                 (11)  b_k = a_i;   i++;   k++;
                          }
               (12)  else
                          (13)  while  ( k ⩽ r ) {
                                 (14)  b_k = a_j;   j++;   k++;
                          }
     }
 (14)  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:

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

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

$$L(N) = 2L\Big(\frac{1}{2}N\Big) + cN = 4L\Big(\frac{1}{4}N\Big) + 2cN = \cdots = \frac{N}{2}L(2) + (m-1)cN$$
$$= cN\log N + (0.5-c)N\,.$$

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

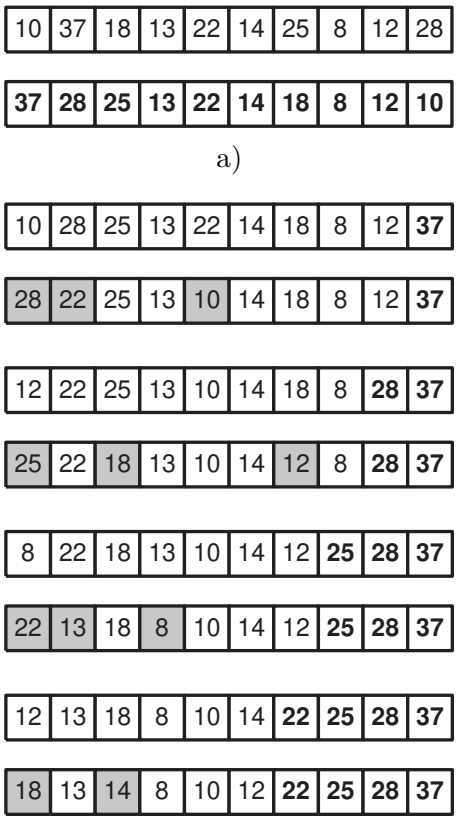
1.1.4. Piramidinis rūšiavimo algoritmas

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 (a_1, a_i);
          (4)  HeapDownOrder (1, i-1);
     }
end
```

1.3 pavyzdys. Skaičių masyvo rūšiavimas. Piramidės algoritmu surūšiuosime ankstesniame pavyzdyje pateiktą skaičių masyvą. Rūšiavimo eiga pavaizduota 1.2 brėžinyje.



1.2 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

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.

1.2. Specialieji rūšiavimo algoritmai

Šiame poskyryje susipažinsime su specialiaisiais rūšiavimo algoritmais, kurie yra greitesni už bendruosius algoritmus, kadangi naudojamės papildoma informacija apie aibės elementus arba uždavinys tenkina papildomus apribojimus.

1.2.1. Sveikų skaičių rūšiavimas

Reikia surūšiuoti masyvą A ir žinome, kad jo elementų raktai yra sveiki skaičiai $1, 2, \ldots, N$, tik pasiskirstę bet kokia tvarka. Jeigu galime naudoti papildomą masyvą B, tai sudarome tokį paprastą ir ekonomišką algoritmą.

Sveikų skaičių rūšiavimo algoritmas 1

(1) **for** (i=1; i ⩽ N; i++)
  (2) $b_{a_i.key} = a_i$;
(3) **for** (i=1; i ⩽ N; i++) $a_i = b_i$;

Jei papildomo masyvo naudoti negalime, tai modifikuojame algoritmą 1.

Sveikų skaičių rūšiavimo algoritmas 2

(1) **for** (i=1; i < N; i++)
  (2) **while** ($a_i.key$ <> i)
    (3) swap (a_i, $a_{a_i.key}$);

Tokio algoritmo vykdymo metu reikės atlikti mažiau nei N elementų sukeitimų vietomis.

1.2.2. Radix rūšiavimo algoritmas

Tarkime, kad turime A aibę, kurios elementų raktai yra sveiki k–ženkliai skaičiai:

$$0 \leqslant a_i.key < 10^{k+1}\,,$$

tačiau nebūtinai visi šio intevalo skaičiai yra panaudoti.

Rūšiavimo algoritme pasinaudojame dešimtainės aritmetikos taisyklėmis. Pirmiausia visus elementus suskirstome į dešimt poaibių pagal paskutinį rakto skaitmenį (vienetų poziciją). Po to visus poaibius jų eiliškumo tvarka sujungiame į vieną aibę. Gautąją aibę vėl skirstome į dešimt poaibių pagal priešpaskutinį rakto skaitmenį (dešimčių poziciją). Šį procesą kartojame k kartų.

1.4 pavyzdys. Sveikųjų skaičių masyvo rūšiavimas Radix algoritmu. Surūšiuosime dvejetainių sveikųjų skaičių masyvą

$A = (73, 29, 92, 14, 74, 45, 54, 18, 3, 97, 9, 61, 11, 63, 35, 37)$.

Aibę suskirstome į dešimt poaibių pagal paskutinį rakto skaitmenį

$$
\begin{aligned}
&\mathbf{0}: \\
&\mathbf{1}: \ 6\mathbf{1},\ 1\mathbf{1}, \\
&\mathbf{2}: \ 9\mathbf{2} \\
&\mathbf{3}: \ 7\mathbf{3},\ \mathbf{3},\ 6\mathbf{3}, \\
&\mathbf{4}: \ 1\mathbf{4},\ 7\mathbf{4},\ 5\mathbf{4}, \\
&\mathbf{5}: \ 4\mathbf{5},\ 3\mathbf{5}, \\
&\mathbf{6}: \\
&\mathbf{7}: \ 9\mathbf{7},\ 3\mathbf{7}, \\
&\mathbf{8}: \ 1\mathbf{8}, \\
&\mathbf{9}: \ 2\mathbf{9},\ \mathbf{9}.
\end{aligned}
$$

Visus poaibius jų eiliškumo tvarka sujungiame į vieną aibę

$A = (61, 11, 92, 73, 3, 63, 14, 74, 54, 45, 35, 97, 37, 18, 29, 9)$.

Gautąją aibę vėl skirstome į dešimt poaibių pagal pirmąjį rakto skaitmenį

$$
\begin{aligned}
&\mathbf{0}: \ \mathbf{0}3,\ \mathbf{0}9, \\
&\mathbf{1}: \ \mathbf{1}1,\ \mathbf{1}4,\ \mathbf{1}8, \\
&\mathbf{2}: \ \mathbf{2}9, \\
&\mathbf{3}: \ \mathbf{3}5,\ \mathbf{3}7, \\
&\mathbf{4}: \ \mathbf{4}5, \\
&\mathbf{5}: \ \mathbf{5}4, \\
&\mathbf{6}: \ \mathbf{6}1,\ \mathbf{6}3, \\
&\mathbf{7}: \ \mathbf{7}3,\ \mathbf{7}4, \\
&\mathbf{8}: \\
&\mathbf{9}: \ \mathbf{9}2,\ \mathbf{9}7.
\end{aligned}
$$

Sujungę poaibius, gauname surūšiuotą elementų aibę

$A = (3, 9, 11, 14, 18, 29, 35, 37, 45, 54, 61, 63, 73, 74, 92, 97)$.

Algoritmo teisingumo analizė. Įrodysime, kad įvykdžius Radix algoritmą gauname teisingai surūšiuotą aibę. Užtenka išnagrinėti dviženklių skaičių atvejį, o bendrojo atvejo teisingumas įrodomas indukcijos metodu. Imkime du skaičius:

$$X = 10\,a + b, \;\; Y = 10\,c + d, \quad 0 \leqslant a, b, c, d \leqslant 9.$$

Nelygybė $X < Y$ yra teisinga, jei

1) $a < c$,

2) $a = c,\ b < d$.

Jai $a < c$, tai antrajame Radix rūšiavimo algoritmo žingsnyje X patenka į poaibį su mažesniu numeriu nei Y. Antruoju atveju pirmajame Radix rūšiavimo algoritmo žingsnyje X pateko į poaibį su mažesniu numeriu nei Y. Po antrojo žingsnio abu elementai pateks į tą patį poaibį, bet X bus įtrauktas anksčiau. Taigi abiem atvejais elementai bus teisingai surūšiuoti.

Algoritmo sudėtingumo analizė. Įvertinsime, kiek bazinių veiksmų reikia atlikti rūšiuojant N elementų. Kiekviename Radix algoritmo žingsnyje tokių veiksmų yra N, o žingsnių skaičius yra $k = \lg N$, todėl algoritmo apimtis yra $N \lg N$.

Radix rūšiavimo algoritmą ypač patogu realizuoti panaudojant *eilės* duomenų struktūrą. Tai leidžia esminiai sumažinti elementų palyginimų ir kopijavimų skaičių.

1.2.3. Išorinio rūšiavimo uždavinys

Šiame poskyryje nagrinėsime rūšiavimo algoritmus, kai duomenų yra tiek daug, kad jie visi netelpa operatyviojoje kompiuterio atmintyje ir juos saugome talpesnėje, bet lėtesnėje išorinėje atmintyje, pvz. tiesioginės kreipties failuose. Tada lėčiausia operacija yra duomenų perrašymas iš sparčiosios atminties į išorinę atmintį ir atvirkščiai. Todėl svarbiausia yra minimizuoti tokių veiksmų apimtį.

Toks uždavinys yra vadinamas *išoriniu* rūšiavimu. Ankstesniuose poskyriuose išnagrinėjome *vidinio* rūšiavimo algoritmus.

Pateiksime *suliejimo* algoritmo modifikaciją, pritaikytą išorinio rūšiavimo uždavinio sprendimui. Tarkime, kad operatyviojoje atmintyje telpa M elementų.

- Iš failo F skaitome M dydžio duomenų blokus, juos rūšiuojame kokiu nors greituoju vidinio rūšiavimo algoritmu ir paeiliui užrašome į failus $F1, F2$. Paskutinio bloko ilgis gali būti ir mažesnis už M.

- Suliejimo algoritmu sujungiame M ilgio blokus iš failų $F1, F2$, gautuosius $2M$ ilgio blokus įrašome paeiliui į failus $F3, F4$. Algoritmą kartojame tol, kol gauname surūšiuotą N ilgio bloką, užrašytą vienam iš failų.

Išorinio rūšiavimo algoritmas

```
(1)  FILES   F, F1, F2, F3, F4;
(2)  reset (F),   rewrite(F1),   rewrite(F2),   i=0;
(3)  while  ( nonempty (F) ) {
            (4)  J = read (F, M);
            (5)  MergeSort (1, J);
            (6)  if  ( i == 0 ) write (F1, J);
            (7)  else  write (F2, J);
            (8)  i++;  i = i % 2;
     }
(9)   J = M;
(10)  FR1 = F1,   FR2 = F2,   FW1= F3,   FW2= F4;
(11)  while  ( J < N ) {
            (12)  reset (FR1),   reset (FR2),   rewrite(FW1),   rewrite(FW2);
            (13)  Merge (J, FR1, FR2, FW1, FW2);
            (14)  J = J +J;
            (15)  FT1 = FW1,   FT2 = FW2;
            (16)  FW1 = FR1,   FW2 = FR2,   FR1= FT1,   FR2= FT2;
      }
```

Pateiksime ir suliejimo algoritmą.

Suliejimo algoritmas

```
Merge ( J, FR1, FR2, FW1, FW2 )
begin
(1)  i = 0;   FW = FW1;
(2)  fin1 = 1;   fin2 = 1;
(3)  while  ( fin1 + fin2 > 0 ) {
            (4)  if  ( fin1 > 0 )
                        (5)  if  ( nonempty(FR1) )   a = read (FR1);
                        (6)  else  fin1 = 0;
            (7)  if  ( fin2 > 0 )
                        (8)  if  ( nonempty(FR2) )   b = read (FR2);
```

```
                (9)  else  fin2 = 0;
          (10)  if  ( fin1 + fin2 > 0 ) {
                  (11)  i++;
                  (12)  if  ( fin1 + fin2 == 2 )
                            (13)  if  ( a < b )  write (a, FW);
                            (14)  else  write (b, FW);
                  (15)  else
                            (16)  if  ( fin1 > 0 )  write (a, FW);
                            (17)  else  write (b, FW);
              }
          (18)  if  ( i == (2*J) ) {
                  (19)  if  ( FW == FW1 ) FW = FW2;
                  (20)  else  FW = FW1;
                  (21)  i =0;
                }
      }
end Merge
```

1.5 pavyzdys. Skaičių masyvo rūšiavimas suliejimo algoritmu. Faile F užrašytas skaičių masyvas, kurio ilgis $N = 29$:

(4, 5, 2, 8, 4, 1, 7, 9, 2, 3, 0, 3, 8, 6, 2, 4, 9, 3, 9, 5, 0, 4, 6, 2, 5, 3, 5, 1, 0).

Imkime $M = 3$, tada masyvo rūšiavimo eiga yra tokia:

$M = 3$:
F1 = (2, 4, 5 | 2, 7, 9 | 2, 6, 8 | 0, 5, 9 | 3, 5, 5)
F2 = (1, 4, 8 | 0, 3, 3 | 3, 4, 9 | 2, 4, 6 | 0, 1)

$M = 6$:
F3 = (1, 2, 4, 4, 5, 8 | 2, 3, 4, 6, 8, 9 | 0, 1, 3, 5, 5)
F4 = (0, 2, 3, 3, 7, 9 | 0, 2, 4, 5, 6, 9)

$M = 12$:
F1 = (0, 1, 2, 2, 3, 3, 4, 4, 5, 7, 8, 9 | 0, 1, 3, 5, 5)
F2 = (0, 2, 2, 3, 4, 4, 5, 6, 6, 8, 9, 9)

$M = 24$:

F3 = (0, 0, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 6, 6, 7, 8, 8, 9, 9, 9)

F4 = (0, 1, 3, 5, 5)

$M = 48$:

F1 = (0, 0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 7, 8, 8, 9, 9, 9)

Algoritmo sudėtingumo įvertinimas. Tegul $N = rM$. Analizė bus paprastesnė, jei imsime $r = 2^k$, tada viso įvykdysime k sujungimo etapų. Pirmiausia rūšiuojame M ilgio poaibius, vieną poaibį sutvarkome atlikdami $cM \log M$ veiksmų. Viso tokių poaibių yra r, todėl šiame algoritmo žingsnyje atliekame $cN \log M$ veiksmų.

Vykdydami apjungimo žingsnius, jungiame $\frac{r}{2}$ poras M ilgio poaibių, $\frac{r}{4}$ poras $2M$ ilgio poaibių ir t.t. Apjungdami l ilgio poaibius, atliekame nedaugiau kaip $2l - 1$ elementų sulyginimo veiksmus, todėl didžiausias lyginimų skaičius yra

$$\frac{r}{2}\big(2M-1\big)+\frac{r}{4}\big(4M-1\big)+\ldots+\frac{r}{2^k}\big(2^kM-1\big)\leqslant krM = N\log\Big(\frac{N}{M}\Big).$$

Taigi išorinio rūšiavimo algoritmo veiksmų skaičius yra nedidesnis už

$$N\Big(c\log M+\log\Big(\frac{N}{M}\Big)\Big).$$

Įvertinsime, kiek kartų duomenys perrašomi iš failų į greitąją operatyvinę kompiuterio atmintį. Priminsime, kad kaip tik šios operacijos vykdymas ir sudaro išorinio rūšiavimo algoritmo didžiąją kaštų dalį. Kiekvieno etapo metu nuskaitomi ir įrašomi visi N elementai, bendras etapų skaičius yra $k + 1$, todėl viso atliekame $N \log \left(\frac{N}{M}\right)$ duomenų skaitymo veiksmų.