# Тема 6. Сортування за лінійний час

## 6.1. Нижня оцінка алгоритмів сортування

Алгоритми сортування, які були розглянуті у попередніх темах, мають одну спільну властивість: при сортуванні використовується тільки порівняння вхідних елементів. Такі алгоритми сортування можна назвати сортуванням на основі порівнянь (або просто – сортування порівняннями). Для деяких з цих алгоритмів – сортування методом злиття – ми отримали час роботи в найгіршому випадку рівним $\Theta(n \lg n)$, а для деяких – швидке сортування – оцінка $\Theta(n \lg n)$ отримується в середньому випадку. В цьому розділі ми покажемо, що алгоритми, які засновані на порівнянні окремих елементів вхідного масиву, в найгіршому випадку повинні зробити мінімум $\Omega(n \lg n)$ порівнянь.

В алгоритмах сортування порівняннями для отримання інформації про розташування елементів вхідної послідовності $\langle a_1, a_2, \ldots, a_n \rangle$ використовуються тільки попарні порівняння елементів. Іншими словами, для визначення взаємного порядку двох елементів $a_i$ та $a_j$ виконується одна з перевірок $a_i < a_j$, $a_i \le a_j$, $a_i = a_j$, $a_i \ge a_j$ або $a_i > a_j$. Значення самих елементів або інша інформація про них відсутня.

При знаходженні нижньої оцінки без втрати загальності ми будемо вважати, що всі вхідні елементи різні, а тому порівняння $a_i = a_j$ стає зайвим. А всі інші варіанти перевірок можна звести до однієї: $a_i < a_j$, на основі якої ми й будемо вести свої наступні міркування.

Вивчення алгоритмів сортування порівняннями можна проводити за допомогою дерев рішень. Дерево рішень – це повне бінарне дерево, в якому представлені операції порівняння елементів, які виконуються тим або іншим алгоритмом сортування. Загальні операції по керуванню процесом, операції переміщення даних та всі інші аспекти алгоритму ігноруються. На рис. 6.1 показане дерево рішень, яке відповідає сортуванню за методом включення і яке опрацьовує вхідну послідовність з трьох елементів.

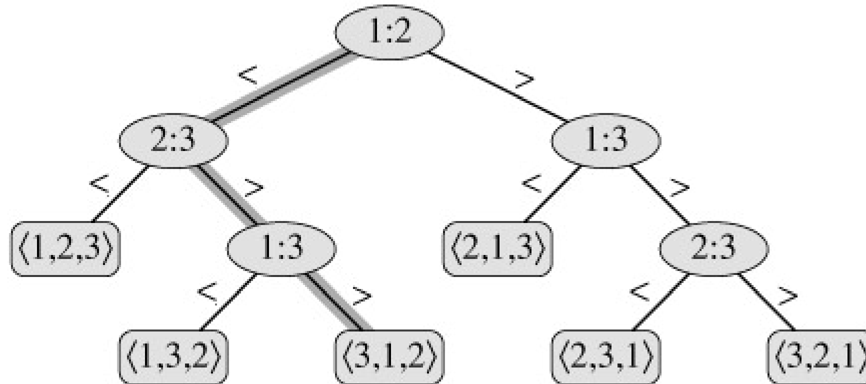


Рис. 6.1. Дерево рішень для алгоритму сортування включенням для трьох елементів.

В дереві рішень кожний внутрішній вузол позначений двома індексами: $i$ та $j$, які належать інтервалу 1, .., $n$, де $n$ – кількість елементів у вхідній послідовності. Мітка вузла вказує на те, що порівнюються два елементи $a_i$ та $a_j$. Кожний лист позначений перестановкою $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$, яка представляє кінцеве впорядкування елементів $\langle a_{\pi(1)} \le a_{\pi(2)} \le \ldots \le a_{\pi(n)} \rangle$. Виконання алгоритму сортування відповідає проходженню шляху від кореня дерева до одного з його листків. Потовщеною сірою лінією на рис. 6.1. позначена послідовність рішень, які приймає алгоритм при сортуванні вхідної послідовності $\langle a_1 = 6, a_2 = 8, a_3 = 5 \rangle$; вказана в листку дерева перестановка $\langle 3, 1, 2 \rangle$ вказує на те, що порядок сортування визначається співвідношеннями $a_3 = 5 \le a_1 = 6 \le a_2 = 8$.

В даному випадку взагалі є 3! = 6 можливих перестановок вхідних елементів, тому дерево рішень повинно мати не менше 6 листків. В загальному випадку в кожному

внутрішньому вузлі відбувається порівняння $a_i < a_j$. Лівим піддеревам відповідають подальші порівняння, які необхідно виконати при $a_i < a_j$, а правим – порівняння, які потрібно виконати при $a_i > a_j$. Дійшовши до якого-небудь листка, алгоритм сортування встановлює відповідне цьому листку впорядкування елементів $\left\langle a_{\pi(1)} \le a_{\pi(2)} \le \ldots \le a_{\pi(n)} \right\rangle$. Оскільки кожний коректний алгоритм сортування повинен бути здатним провести будь-яку перестановку вхідних елементів, необхідна умова коректного сортування порівняннями полягає в тому, що в листках дерева повинні розміститись всі $n!$ перестановок $n$ елементів, і що з кореня дерева до кожного його листка можна прокласти шлях, який відповідає одному з реальних варіантів роботи сортування порівняннями.

Довжина найдовшого шляху від кореня дерева рішень до будь-якого листка відповідає кількості порівнянь які виконуються в алгоритмі сортування в найгіршому випадку. Відповідно, кількість порівнянь, які виконуються в найгіршому випадку в алгоритмі сортування, дорівнює висоті цього дерева. Тож, нижня оцінка висоти дерева є нижньою оцінкою часу роботи для будь-якого алгоритму сортування порівняннями. Цю оцінку надає наступна теорема.

Теорема 6.1. В найгіршому випадку під час виконання будь-якого алгоритму сортування порівняннями виконується $\Omega(n\lg n)$ порівнянь.

*Доведення*. З наведених вище міркувань стає зрозуміло, що для доведення теореми достатньо визначити висоту дерева, в якому кожна перестановка представлена листком. Розглянемо дерево рішень висотою $h$ з $l$ листками, яке відповідає сортуванню $n$ елементів. Оскільки кожна з $n!$ перестановок вхідних елементів зіставляється з одним із листків, $n! \le l$. Бінарне дерево висотою $h$ має не більше $2^h$ листків, а тому отримуємо:

$$n! \le l \le 2^h .$$

Після логарифмування попереднього виразу, отримуємо:

$$h \ge \log_2 n!. \tag{6.1}$$

Скористаємось формулою Стірлінґа:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \text{ або } n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right).$$

Вираз (6.1) можна перетворити наступним чином:

$$h \ge \log_2 n = \log_2\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)\right) = \log_2\sqrt{2\pi n} + \log_2\left(\frac{n}{e}\right)^n + \log_2\left(1+\Theta\left(\frac{1}{n}\right)\right) =$$

$$= \frac{1}{2}\log_2 2\pi n + n\log_2\frac{n}{e} + \log_2\left(1+\Theta\left(\frac{1}{n}\right)\right) = \frac{1}{2}\log_2 2\pi n + n\log_2 n - n\log_2 e + \log_2\left(1+\Theta\left(\frac{1}{n}\right)\right) =$$

$$= \Theta(n\lg n).$$

Звідки отримуємо, що $h = \Omega(n\lg n)$. ■

### 6.2. Сортування підрахунком

Далі ми розглянемо два алгоритми сортування, час роботи яких є лінійним по відношенню до розмірності вхідного масиву. Зрозуміло, що результат теореми 6.1 не дозволяє, щоб дані алгоритми працювали за рахунок порівняння елементів вхідного масиву. Власне, вони цього й не роблять і, таким чином, відсортовують вхідний масив не порівнюючи його елементи один з одним.

В сортуванні підрахунком (*counting sort*) передбачається, що всі $n$ елементів – цілі числа, які лежать в інтервалі від 0 до $k$, де $k$ – деяка ціла стала. Якщо $k = O(n)$, то час роботи алгоритму сортування підрахунком дорівнює $\Theta(n)$.

Основна ідея сортування підрахунком полягає в тому, що для кожного вхідного елементу $x$ визначити кількість елементів, які менші за $x$. За допомогою цієї інформації елемент $x$ можна розташувати на тій позиції вихідного масиву, де він повинен

знаходитись. Наприклад, якщо всього є 17 елементів, які менші за *x*, то у вихідній послідовності елемент *x* повинен займати 18-у позицію. Якщо дозволяється ситуація, коли декілька елементів мають одне й те саме значення, цю схему доведеться дещо модифікувати, оскільки ми не можемо розмістити всі такі елементи в одній і тій самій позиції.

На вхід алгоритму сортування методом підрахунку подається масив *A* довжиною *n*. Відсортований масив записується у вихідний масив *B*, який має також розмірність *n*. Під час роботи алгоритму створюється допоміжний масив *C* довжиною *k*, який слугує тимчасовим робочим сховищем.

```
CountingSort(A, B, k)
1  for i ← 0 to k
2     do C[i] ← 0
3  for j ← 1 to length[A]
4     do C[A[j]] ← C[A[j]] + 1
5  // в C[i] зберігається кількість елементів, які дорівнюють i
6  for i ← 1 to k
7     do C[i] ← C[i] + C[i-1]
8  // в C[i] зберігається кількість елементів, які не більші за i
9  for j ← length[A] downto 1
10    do B[C[A[j]]] ← A[j]
11       C[A[j]] ← C[A[j]] - 1
```

Лістинг 6.1. Процедура сортування `CountingSort`.

Робота алгоритму сортування підрахунком наведена на рис. 6.2. В частині *а* рисунку показаний масив *A* та допоміжний масив *C* після виконання рядку 4. В частині *б* рисунку масив *C* наведений після виконання рядку 7. В частинах *в-д* показаний стан вихідного масиву *B* та допоміжного масиву *C* після, відповідно, однієї, двох та трьох ітерацій циклу в рядках 9-11. Заповненими є тільки світло-сірі елементи масиву *B*. Кінцевий відсортований масив *B* відображений в частині *е* рисунку.

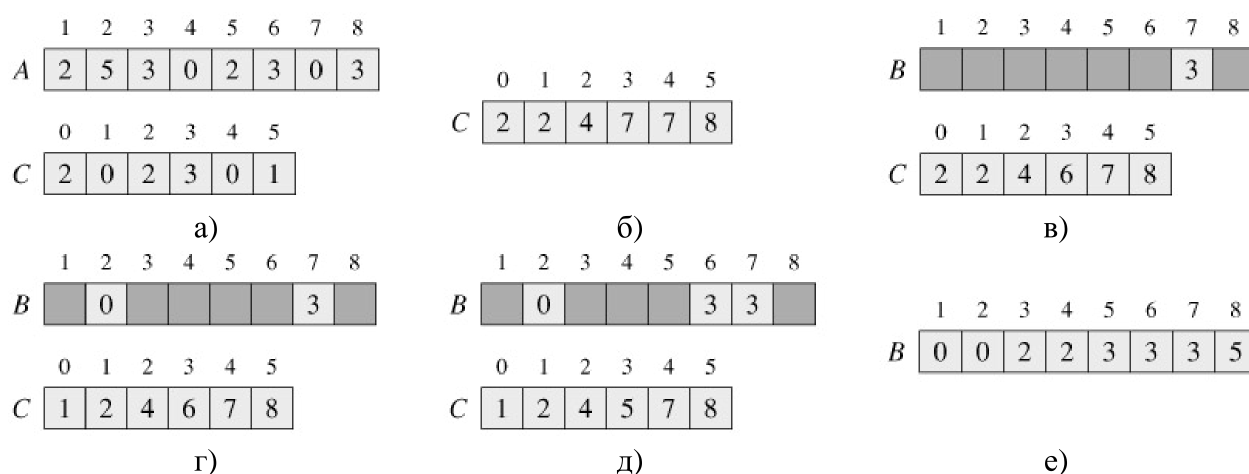


Рис. 6.2. Обробка алгоритмом `CountingSort` вхідного масиву *A*[1…8]

Після ініціалізації в циклі **for** в рядках 1-2, в циклі **for** в рядка 3-4 виконується перевірка кожного вхідного елементу. Якщо його значення дорівнює *i*, то до величини *C*[*i*] додається одиниця. Таким чином, після виконання рядку 4 для кожного $i = 0, 1, \ldots, k$ у комірці *C*[*i*] зберігається кількість вхідних елементів, які дорівнюють *i*. В рядках 6-7 для кожного $i = 0, 1, \ldots, k$ визначається число елементів, які не більші за *i*.

Нарешті, в циклі **for** в рядках 9-11 кожний елемент *A*[*j*] розміщується у відповідну позицію вихідного масиву *B*. Якщо всі *n* елементів різні, то при першому переході до

рядку 9 для кожного елементу $A[j]$ у комірці $C[A[j]]$ зберігається коректний індекс кінцевого положення цього елементу у вихідному масиві, оскільки є $C[A[j]]$, які менші за $A[j]$. Оскільки різні елементи можуть мати одні й ті самі значення, розміщуючи значення $A[j]$ в масив $B$, ми кожний раз зменшуємо $C[A[j]]$ на одиницю. Завдяки цьому наступний вхідний елемент, значення якого дорівнює $A[j]$ (якщо такий є), у вихідному масиву розміщується безпосередньо перед елементом $A[j]$.

Оцінимо кількість часу, необхідного на сортування методом підрахунку. На виконання циклу **for** у рядках 1-2 витрачається час $\Theta(k)$, цикл в рядках 3-4 потребує час $\Theta(n)$, у рядках 6-7 – $\Theta(k)$, а цикл у рядках 9-11 – $\Theta(n)$. Таким чином, повний час можна записати як $\Theta(k + n)$. На практиці сортування підрахунком використовується, коли $k = O(n)$, а в цьому випадку час роботи алгоритму дорівнює $\Theta(n)$.

В алгоритмі сортування підрахунком нижня границя $\Omega(n \lg n)$, про яку йшлась мова раніше, виявляється перевершеною, оскільки описаний алгоритм не заснований на порівняннях. Фактично ніде в коді не виконується порівняння вхідних елементів – замість цього безпосередньо використовуються їх значення, за допомогою яких елементам зіставляться конкретні індекси.

Важлива властивість алгоритму сортування підрахунком полягає в тому, що він **стійкий**: елементи з одним й тим самим значенням знаходяться у вихідному масиві в тому самому порядку, що й у вхідному. Зазвичай властивість стійкості важлива тільки для ситуації, коли разом з елементами для сортування є додаткові дані. Стійкість, яка властива сортуванню підрахунком, важлива також ще й з іншої причини: цей алгоритм часто використовується в якості підпрограми для сортування за розрядами.

### 6.3. Сортування за розрядами

Припустимо, що нам потрібно відсортувати числа, які записані у десятковій системі та які мають $d$ розрядів. Причому процес сортування природно виконувати за розрядами цих чисел. У нас є два варіанти як проходити ці розряди: або від молодших до старших, або від старших до молодших. І, якщо останній спосіб більше підходить для сортування слів над деяким алфавітом у лексикографічному порядку, то для сортування чисел краще використовувати якраз сортування від молодших розрядів до старших.

Даний процес сортування зручно представити наступним чином. Нехай всі числа записані на окремі картки. Спочатку всі картки розсортовані у довільному порядку і задача полягає в тому, щоб після процедури сортування ці картки знаходились у відсортованому порядку: від менших чисел до більших.

Уявимо, що ми склали всі картки в одну колоду. Процедура сортування починається з того, що ми сортуємо цю колоду карток за молодшим розрядом кожного числа так, що отримуємо до десяти окремих стопок, кожна з яких містить картки з числами, що мають спільний молодший розряд. Після цього ми об'єднуємо ці стопки у нову колоду так, що спочатку в ній йдуть картки з молодшим розрядом 0, потім – 1, 2, і т.д. Потім вся колода сортується схожим чином, але вже по передостанній цифрі (другому молодшому розряду), і картки знову збираються в одну колоду. Процес продовжується до тих пір, поки картки не виявляться відсортованими по всіх $d$ розрядах, що автоматично означатиме, що картки в результуючій колоді відсортовані у зростанні $d$-значнх чисел. Отже, для сортування таких чисел потрібно лише $d$ проходів колоди карток.

На рис. 6.3 показано, як сортування за розрядами опрацьовує «колоду» з семи трьохзначних чисел. У крайньому лівому стовпчику показані вхідні числа, а у наступних стовпчиках – послідовні стани списку після його сортування за цифрами, починаючі з молодшої. Сірим кольором виділений поточний розряд, за яким виконується сортування, в результаті чого отримується наступний (розташований справа) стовпець.

Важливо, щоб сортування за цифрами того або іншого розряду була стійким.

Нижче наводиться псевдокод процедури сортування за розрядами. Передбачається, що кожний з $n$ елементів масиву $A$ – це число, в якого всього $d$ цифр, причому перша цифра стоїть в самому молодшому розряді, а цифра під номером $d$ – в самому старшому.

| | | | | | | |
|---|---|---|---|---|---|---|
| 329 | | 720 | | 720 | | 329 |
| 457 | | 355 | | 329 | | 355 |
| 657 | | 436 | | 436 | | 436 |
| 839 | → | 457 | → | 839 | → | 457 |
| 436 | | 657 | | 355 | | 657 |
| 720 | | 329 | | 457 | | 720 |
| 355 | | 839 | | 657 | | 839 |

Рис. 6.3. Приклад роботи сортування за розрядами.

```
RadixSort(A, d)
1  for i ← 1 to d
2      do Стійке сортування масиву A за i-м розрядом
```

Лістинг 6.2. Процедура сортування `RadixSort`.

Теорема 6.2. Нехай є $n$ $d$-значних чисел, в яких кожна цифра приймає одне з $k$ можливих значень. Тоді алгоритм `RadixSort` дозволяє виконати коректне сортування цих чисел за час $\Theta(d(n+k))$, якщо стійке сортування, яке використовуються цим алгоритмом, має час роботи $\Theta(n+k)$.

*Доведення.* Коректність сортування за розрядами доводиться методом математичної індукції по стовпцях, за якими відбувається сортування. Аналіз часу роботи алгоритму залежить від того, який з методів стійкого сортування використовується в якості допоміжного алгоритму. Якщо кожна цифра належить інтервалу від 0 до $k-1$, і $k$ не надто велике, то сортування за розрядами – оптимальній вибір. Для оброки кожної з $d$ цифр $n$ чисел потрібний час $\Theta(n+k)$, а всі цифри будуть опрацьовані алгоритмом сортування за розрядами за час $\Theta(d(n+k))$. ■

Якщо відомо, що $k = O(n)$ і $d$ – стала, то час роботи сортування за розрядами лінійно залежить від кількості вхідних елементів.

Чи є сортування за розрядами кращим за, наприклад, швидке сортування, адже час $\Theta(n)$ виглядає більш прийнятним, ніж $\Theta(n \lg n)$? Проте слід пам'ятати, що в цих виразах приховані сталі множники. Не дивлячись на те, що для сортування за розрядами можливо буде потрібна менша кількість проходів, ніж для швидкого сортування, але кожний прохід сортування за розрядами може виявитись достатньо довгим у порівнянні з проходами для швидкого сортування. Вибір адекватного алгоритму сортування залежить від характеристик реалізації алгоритму, машини, на якій відбувається сортування (наприклад, при швидкому сортуванні апаратний кеш часто використовується ефективніше, ніж при сортуванні за розрядами), а також від вхідних даних. Окрім того, в тій версії сортування за розрядами, де використовується стійке сортування підрахунком в якості допоміжного етапу, обробка елементів відбувається із залученням додаткової пам'яті, яка не потрібна тому ж таки алгоритму швидкого сортування.

**Література**

Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford (2001) [1990]. Introduction to Algorithms (2nd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03293-7. Глава 8, розділи 8.1 – 8.3.