Розділ 3. Прості алгоритми внутрішнього сортування

Алгоритм сортування – це алгоритм для впорядкування елементів у деякій структурі даних за зростанням чи спаданням. У випадку наявності елементів з однаковими значеннями, в упорядкованій послідовності вони розташовуються поруч один за одним у будь-якому порядку. Однак іноді буває корисно дотримуватися початкового порядку елементів з однаковими значеннями.

В залежності від того, над якою структурою даних здійснюється сортування, буває сортування масиву, зв'язаного списку, дерева, графа, таблиці.

У випадку, коли елемент структури даних має декілька полів, вводиться поняття ключ сортування.

Ключ сортування – це поле структури даних, за значенням якого визначається порядок елементів (тобто за яким відбувається сортування). На практиці як ключ часто виступає число, а в інших полях зберігаються які-небудь дані, що ніяк не впливають на роботу алгоритму.

Простий алгоритм сортування можна розбити на 3 частини:

- порівняння, що визначає впорядкованість пари елементів;
- перестановка, що міняє місцями цю пару елементів;
- повтор перших двох дій доти, поки всі елементи структури даних не будуть впорядковані.

Алгоритми сортування мають велике практичне значення. Їх можна зустріти там, де відбувається обробка й зберігання великих обсягів інформації. Вони використовуються також у криптографії, кодуванні, вирішенні математичних задач тощо.

Жодна інша проблема не породила такої кількості різних рішень, як задача сортування. Універсального, найкращого

алгоритму сортування не існує. Однак, маючи приблизні характеристики вхідних даних, можна підібрати метод, що працює оптимальним чином. Для цього необхідно знати параметри, за якими буде здійснюватися оцінка алгоритмів.

Параметри алгоритмів сортування:

– *Час сортування* – основний параметр, що характеризує швидкодію алгоритму.

– *Пам'ять* – один з параметрів, що характеризується тим, що ряд алгоритмів сортування вимагають виділення додаткової пам'яті під тимчасове зберігання даних. При оцінці використаної пам'яті не буде враховуватися місце, що займає вихідний масив даних і незалежні від вхідної послідовності витрати, наприклад, на зберігання коду програми.

– *Стійкість* – це параметр, що відповідає за те, що сортування не змінює взаємного розташування рівних елементів. Наприклад, якщо алфавітний список групи сортується за оцінками, то стійкий метод створює список у якому прізвища студентів з однаковими оцінками будуть впорядковані за алфавітом, а нестійкий метод створить список у якому, можливо, вихідний порядок буде порушений.

– *Природність поводження* – параметр, який вказує на ефективність методу при обробці вже відсортованих, або частково відсортованих даних. Алгоритм поводиться природно, якщо враховує цю характеристику вхідної послідовності й працює краще.

– *Використання операції порівняння.* Алгоритми, що використовують для сортування порівняння елементів між собою, називаються *заснованими на порівняннях*. Мінімальна трудомісткість гіршого випадку для цих алгоритмів становить $O(n \log n)$, але вони відрізняються гнучкістю застосування. Для спеціальних випадків (типів даних) існують більш ефективні алгоритми.

Класифікація алгоритмів сортування

Всі алгоритми сортування можна класифікувати за різними ознаками, наприклад, за стійкістю, за особливостями функціонування, за використанням операцій порівняння, за потребою в додатковій пам'яті, за структурою даних, за шириною області застосування, за потребою в знаннях про структуру даних, що виходять за рамки операцій порівняння ключів тощо.

Розглянемо більш детально класифікацію алгоритмів сортування за використанням пам'яті. У цьому випадку основні типи сортування діляться на:

– *Внутрішнє сортування* – це алгоритм сортування, що у процесі впорядкування даних використовує тільки оперативну пам'ять (ОЗП) комп'ютера. Тобто оперативної пам'яті досить для розміщення в ній масиву даних, що сортується з довільним доступом до будь-якої комірки й власне для виконання алгоритму. Внутрішнє сортування застосовується у всіх випадках, за винятком однопрохідного зчитування даних і однопрохідного запису відсортованих даних. Залежно від конкретного алгоритму і його реалізації дані можуть сортуватися в тій же області пам'яті, або використовувати додаткову оперативну пам'ять.

– *Зовнішнє сортування* – це алгоритм сортування, що при проведенні впорядкування даних використовує зовнішню пам'ять, як правило, жорсткі диски. Зовнішнє сортування розроблене для обробки великих структур даних, які не поміщаються в оперативну пам'ять. Звертання до різних носіїв накладає деякі додаткові обмеження на даний алгоритм: доступ до носія здійснюється послідовним чином, тобто в кожний момент часу можна зчитати або записати тільки елемент наступний за поточним; обсяг даних не дозволяє їм розміститися в ОЗП.

Внутрішнє сортування є базовим для будь-якого

алгоритму зовнішнього сортування – окремі частини масиву даних сортуються в оперативній пам'яті й за допомогою спеціального алгоритму з’єднуються в один масив, упорядкований за ключем.

Слід зазначити, що внутрішнє сортування значно ефективніше зовнішнього, тому що на звертання до оперативної пам'яті затрачується набагато менше часу, ніж до носіїв.

Одна з найбільш розповсюджених класифікацій методів сортування наведена на рисунку 3.1.

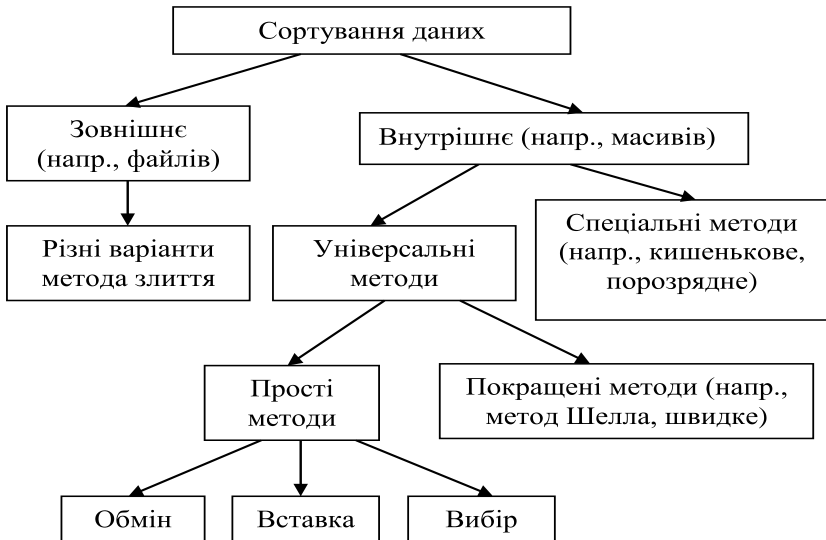

Рисунок 3.1 – Загальна класифікація методів сортування

Класифікація за часом виконання:

За час $O(n^2)$:

- сортування вибором;
- сортування вставкою;
- сортування обміном.

За час $O(n \log n)$:

- пірамідальне сортування;
- швидке сортування;
- сортування злиттям.

За час *O(n)* з використанням додаткової інформації про елементи :

- сортування підрахунком;
- сортування за розрядами;
- сортування комірками.

За час $O(n \log^2 n)$:

- сортування злиттям модифіковане;
- сортування Шелла.

Алгоритми стійкого впорядкування:

- сортування вибором;
- сортування вставкою;
- сортування обміном;
- сортування злиттям;
- сортування підрахунком;
- сортування за розрядами;
- сортування комірками;
- сортування злиттям модифіковане.

Прості алгоритми внутрішнього сортування

Щоб сконцентруватися на алгоритмічних питаннях, ми будемо працювати з алгоритмами, які сортують масиви цілих чисел. Ці алгоритми легко адаптувати для сортування записів.

В основному програми сортування працюють із записами двома способами: або вони порівнюють і сортують тільки ключі, або пересувають записи повністю. Більшість алгоритмів, які ми вивчимо можна застосовувати, використовуючи їхнє переформулювання в термінах цих двох операцій, для довільних записів.

Якщо записи досить великі, то звичайно намагаються

уникнути їхнього пересування за допомогою "непрямого сортування": при цьому самі записи не сортуються, а замість цього сортується масив вказівників (індексів), так, що перший вказівник вказує на самий маленький елемент і так далі. Ключі можуть зберігатися або із записами (якщо вони великі), або з вказівниками (якщо вони маленькі).

Сортування вибором

Один з найпростіших методів сортування працює в такий спосіб:

1) знаходимо найменший елемент у масиві;

2) міняємо його місцями з елементом, що знаходиться на першому місці;

3) повторюємо процес із другої позиції у файлі й знайдений найменший елемент обмінюємо із другим елементом і так далі поки весь масив не буде відсортований.

У міру просування зліва направо через масив, елементи ліворуч від вказівника перебувають уже в своїй кінцевій позиції (і їх вже не будуть більше пересувати), тому масив стає повністю відсортованим до того моменту, коли вказівник досягає правого краю.

Цей метод називається *сортуванням вибором* оскільки він працює циклічно вибираючи найменший з елементів, що залишилися.

Псевдокод процедури сортування вибором:

```
Процедура СортуванняВибором(масив a, цілочисельна N)
 Початок
     змінні цілочисельні i, j, min, t
     для i від 1 до N-1 //N-розмір масиву
     початок
          min = i
          //цикл знаходження мінімального елементу
          для j від i + 1 до N
               якщо a[j] < a[min] тоді
                   min = j
```

```
          t = a[min] //заміна елементів
          a[min] =a[i]
          a[i] = t
    кінець
 Кінець
```

Приклад сортування масиву з п'яти елементів за даним алгоритмом:

7 5 2 3 1
1 5 2 3 **7**
1 **2** **5** 3 7
1 2 **3** **5** 7

Цей метод – один з найпростіших, і він працює дуже добре для невеликих структур даних. Його "внутрішній цикл" складається з порівняння a[i]<a[min] (плюс код необхідний для збільшення j та перевірки того, щоб він не перевищив N), що навряд чи можна ще спростити.

Більш того, незважаючи на те, що цей метод очевидно є методом "грубої сили", він має дуже важливе застосування: оскільки кожний елемент пересувається не більше ніж раз, то він дуже гарний для великих записів з маленькими ключами.

Характеристика алгоритму:

- Структура даних: Масив.
- Швидкодія: $O(n^2)$.
- Простір: $O(n)$, $O(1)$.
- Оптимальність: Не практичний.

Сортування вставкою

Сортування вставкою – це метод який майже настільки ж простий, що й сортування вибором, але набагато більш гнучкий. Більшість людей при сортуванні колоди гральних карт, використовують метод, схожий на алгоритм сортування вставкою. Суть алгоритму: беремо один елемент і вставляємо

його в потрібне місце серед тих, що ми вже обробили (тим самим залишаючи їх відсортованими). Алгоритм:

1) зліва направо проходимо масив, порівнюючи сусідні елементи, поки не знайдемо елемент, що розташований не в порядку сортування;

2) обмінюємо цей елемент з елементами розташованими ліворуч від нього, поки він не займе потрібну позицію;

3) повторюємо перші дві дії, поки масив не буде відсортовано.

Приклад сортування вставкою масиву з семи елементів за даним алгоритмом зображено на рис. 3.2.

3	8	9	10	6	7	11	j=5, i=4	6<10
3	8	9	6	10	7	11	i=3	6<9
3	8	6	9	10	7	11	i=2	6<8
3	6	8	9	10	7	11	i=1	6>3
3	6	8	9	10	7	11	j=6, i=5	7<10
3	6	8	9	7	10	11	i=4	7<9
3	6	8	7	9	10	11	i=3	7<8
3	6	7	8	9	10	11	i=2	7>6

Рисунок 3.2 – Приклад сортування вставкою

Псевдокод процедури сортування вставкою:

```
Процедура СортуванняВставкою(масив a, цілочисельна N)
 Початок
    Змінні цілочисельні i, j, t
    для i від 2 до N
    початок
        t = a[i]
        j = i-1
        поки j > 0 та t < a[j]
            початок
                 a[j+1] = a[j]
                 j = j-1
```

```
            кінець
        a[j+1] = t
    кінець
 Кінець
```

Також як і в сортуванні вибором, у процесі сортування вставкою елементи ліворуч від вказівника перебувають уже в відсортованому порядку, але вони не обов'язково перебувають у своїй остаточній позиції, оскільки їх ще можуть пересунути праворуч, щоб вставити більш маленькі елементи, які зустрічаються пізніше. Однак масив стає повністю відсортованим, коли вказівник досягає правого краю.

Даний алгоритм простий в реалізації та ефективний для невеликих масивів і є стійким. Він ефективний при сортуванні масивів, дані в яких вже непогано відсортовані: продуктивність рівна $O(n + d)$, де d – кількість інверсій.

Характеристика алгоритму:

- Структура даних: Масив.
- Швидкодія: $O(n^2)$, для найкращого випадку $O(n + d)$.
- Простір: $O(n)$, $O(1)$.
- Оптимальний: Переважно ні.

Бульбашкове сортування (сортування простими обмінами)

Алгоритм працює таким чином – у масиві порівнюються два сусідні елементи. Якщо один з елементів, не відповідає критерію сортування (є більшим, або ж, навпаки, меншим за свого сусіда), то ці два елементи міняються місцями. Прохід по списку продовжується до тих пір, доки дані не будуть відсортованими. Алгоритм отримав свою назву від того, що процес сортування за ним нагадує поведінку бульбашок повітря у резервуарі з водою. Оскільки для роботи з елементами масиву він використовує лише порівняння, це сортування на основі порівнянь.

```
Процедура БульбашковеСортування(масив a,
Цілочисельна N)
 Початок
     Змінні цілочисельні i, j, t
     для i від 1 до N
         для j від 1  до N-1
           якщо a[j]>a[j+1] тоді
                початок
                     t = a[j]
                     a[j] = a[j+1]
                     a[j+1] = t
                кінець
 Кінець
```

Приклад реалізації крок за кроком

Візьмемо масив чисел "5 1 4 2 8", і за допомогою даного алгоритму, відсортуємо його від найменшого до найбільшого елементу. На кожному кроці, елементи, виділені **жирним** шрифтом будуть порівнюватись.

Перший прохід:

(**5** **1** 4 2 8) ⟶(**1** **5** 4 2 8)
(1 **5** **4** 2 8) ⟶(1 **4** **5** 2 8)
(1 4 **5** **2** 8) ⟶(1 4 **2** **5** 8)
(1 4 2 **5** **8**) ⟶(1 4 2 **5** **8**)

Другий прохід:

(**1** **4** 2 5 8) ⟶(**1** **4** 2 5 8)
(1 **4** **2** 5 8) ⟶(1 **2** **4** 5 8)
(1 2 **4** **5** 8) ⟶(1 2 **4** **5** 8)
(1 2 4 **5** **8**) ⟶(1 2 4 **5** **8**)

Тепер наш масив повністю відсортований, однак, алгоритм цього ще не знає. Йому потрібен ще один "пустий" прохід, під час якого він не поміняє місцями жодного елементу

(якщо реалізувати відповідний прапор, інакше проходів буде стільки скільки елементів у масиві – за класичною реалізацією).

Третій прохід:

```
( 1 2 4 5 8 ) → ( 1 2 4 5 8 )
( 1 2 4 5 8 ) → ( 1 2 4 5 8 )
( 1 2 4 5 8 ) → ( 1 2 4 5 8 )
( 1 2 4 5 8 ) → ( 1 2 4 5 8 )
```

Нарешті, масив відсортовано, і алгоритм може припинити свою роботу.

Одним з найбільш частих застосувань алгоритмів сортування є сортування рядків (напр., списку назв, списку слів, тощо). Зазвичай воно проводиться так: спочатку множина рядків сортується за першим символом кожного рядка, потім кожна підмножина рядків, що мають однаковий перший символ, сортується по другому символу, і так до тих пір, поки всі рядки не будуть впорядковані. При цьому відсутній символ (при порівнянні рядка довжини N з рядком довжини N + 1) вважається меншим будь-якого символу.

Застосування даного методу до рядків, які представляють собою числа у текстовому записі, видає невірні результати: наприклад, «9» виявляється більше, ніж «11», так як перший символ першого рядка має більше значення, ніж перший символ другого. Для виправлення цієї проблеми алгоритм сортування може перетворювати сортовані рядки в числа і сортувати їх як числа. Такий алгоритм називається «числовим сортуванням», а описаний раніше – «строковом сортуванням».

Питання до розділу:

1. Дайте означення алгоритму сортування.
2. Поясніть, що таке ключ сортування.

3. Де застосовуються алгоритми сортування?
4. Охарактеризуйте параметри алгоритмів сортування.
5. Вкажіть ознаки, за яким можна класифікувати алгоритми сортування.
6. Чим відрізняється внутрішнє сортування від зовнішнього, і як вони пов’язані?
7. Що таке непряме сортування і коли воно використовується?
8. Графічно проілюструйте принцип сортування масиву чисел (2 10 5 3 8) в порядку зростання за методом сортування вибором.
9. Графічно проілюструйте принцип сортування масиву чисел (2 10 5 3 8) в порядку зростання за методом сортування вставкою.
10. Використовуючи бульбашкове сортування, графічно проілюструйте принцип сортування масиву чисел (2 10 5 3 8) за зростанням.
11. Порівняйте використанні в пп. 8-10 методи.

Розділ 4. Покращені алгоритми внутрішнього сортування

Покращення алгоритму бульбашкового сортування

Кролики і черепахи. Позиція елементів, що підлягають сортуванню відіграє велику роль у питанні продуктивності даного алгоритму. Великі елементи на початку списку не викликають проблеми, оскільки вони досить швидко переміщуються на свої місця. Однак, малі елементи у кінці списку переміщуються на його початок дуже повільно. Це призвело до того, що ці типи елементів було названо *кроликами* і *черепахами*, відповідно.

З метою підвищення швидкодії алгоритму, у свій час було здійснено чимало зусиль для зменшення кількості "черепах". Сортування змішуванням є порівняно непоганим, однак, усе ще у своєму найгіршому випадку має складність $O(n^2)$. Сортування гребінцем спершу порівнює великі елементи один з одним, а вже тоді поступово переходить до все менших і менших. Його середньостатистична швидкість приблизно рівна такій, як в алгоритмі Швидке сортування.

Сортування змішуванням – один із різновидів алгоритму сортування бульбашкою. Відрізняється від сортування бульбашкою тим, що сортування відбувається в обох напрямках, міняючи напрямок при кожному проході. Даний алгоритм лише трохи складніший за сортування бульбашкою, однак, вирішує так звану проблему "черепах".

Швидкодія

Ефективність алгоритму змішування рівна $O(n^2)$ для середньостатистичного та найгіршого випадку, водночас, вона прямує до $O(n)$ якщо список вже не погано відсортований, наприклад, якщо кожен елемент знаходиться у позиції, яка

відрізняється від кінцевої більше, ніж на k ($k \geq 1$), то його швидкодія рівна $O(k \cdot n)$.

Відмінності від сортування бульбашкою

Сортування змішуванням мало чим відрізняється від сортування бульбашкою. Єдина його відмінність у тому, що замість багаторазового проходження через список знизу вгору, він проходить по черзі знизу вгору і згори вниз. Він може досягати трохи вищої ефективності, ніж алгоритм сортування бульбашкою. Причиною цьому є те, що алгоритм сортування бульбашкою проходить по списку лише в одному напрямі, а тому за одну ітерацію елементи списку можна перемістити лише на один крок.

Наприклад, для того, щоб відсортувати список (2, 3, 4, 5, 1), алгоритму сортування змішуванням достатньо лише одного проходу, у той час, як алгоритму сортування бульбашкою знадобиться чотири проходи. Однак, один прохід сортування змішуванням слід рахувати за два проходи сортування бульбашкою. Зазвичай, сортування змішуванням удвічі швидше за сортування бульбашкою.

Іншою можливою оптимізацією є запам'ятовування попередніх перестановок. У наступній ітерації, перестановки не повторюватимуться, а тому алгоритм матиме коротші проходи по списку.

Псевдокод сортування бульбашкою:

```
procedure bubble()
 begin
    var byte i, byte j, byte t
    for i = 2 to N do
        for j = N down to i do
            if x[i-1] > x[j] then begin
                t = x[j - 1]; x[j - 1] = x[j]; x[j] = t
            end
 end
```

Сортування гребінцем – спрощений алгоритм сортування, розроблений Влодзімежом Добосєвічем (Wlodzimierz Dobosiewicz) у 1980 році, і пізніше заново досліджений та популяризований Стефаном Лейсі (Stephen Lacey) та Річардом Боксом (Richard Box), котрі написали про нього в журналі Byte Magazine у квітні 1991 р. Сортування гребінцем є поліпшенням алгоритму сортування бульбашкою, і конкурує у швидкодії з алгоритмом швидкого сортування. Основна його ідея полягає в тому, щоб усунути так званих "черепах" (малі значення) ближче до кінця списку, оскільки у сортуванні бульбашкою вони сильно уповільнюють процес сортування. ("Кролики" або великі значення на початку списку у сортуванні бульбашкою не викликають проблеми).

У сортуванні бульбашкою, коли два елементи порівнюються, вони завжди мають розрив (відстань один від одного) рівну 1. Основна ідея сортування гребінцем полягає у тому, що цей розрив може бути більший одиниці. (Алгоритм сортування Шелла також базується на даній ідеї, однак, він є модифікацією алгоритму сортування вставками, а не сортування бульбашкою).

Розрив починається зі значення, що рівне довжині списку, поділеного на фактор зменшення (зазвичай, 1.3; див. далі), і список сортується з урахуванням цього значення (при необхідності, воно заокруглюється до цілого). Потім розрив знову ділиться на фактор розриву, і список продовжує сортуватись з новим значенням, процес продовжується до тих пір, поки розрив рівний 1. Далі список сортується з розривом рівним 1 до тих пір, доки не буде повністю відсортований. Таким чином, фінальний етап сортування аналогічний такому ж, як у сортуванні бульбашкою, однак, до цього "черепахи" усуваються.

Фактор зменшення

Фактор зменшення справляє великий ефект на швидкість

алгоритму сортування гребінцем. В оригінальній статті, автор пропонує значення 1.3 після багатьох експериментів з іншими значеннями.

Текст описує процес вдосконалення алгоритму використовуючи значення $1/(1-\frac{1}{e^{\varphi}}) \approx 1.247330950103979$ в якості фактора зменшення. Стаття також містить приклад використання алгоритму на псевдокоді.

Псевдокод сортування гребінцем

```
function combSort(array input)
    gap = input.size

    loop until gap <= 1 and swaps == 0
        if gap > 1
            gap = gap / 1.3
            if gap == 10 or gap = 9
                gap = 11
            end
        end
        i = 0
        swaps = 0
        loop until i + gap <= input.size
            if input[i] > input[i+gap]
             swap(input[i], input[i + gap])
                swaps = 1
            end
            i = i + 1
        end
    end
end
```

Приклад 4.1. Сортування гребінцем на мові C++

```
void sort(data *array,dword size )
{
  if(!array||!size)
    return;
```

```
  dword jump=size;
  bool swapped=true;

  while(jump>1||swapped)
  {
    if(jump>1)
      jump=(dword)(jump/1.25);
    swapped=false;
    for(dword i=0; i+jump<size; i+=jump)
      if(array[i]>array[i+jump])
      swap(array,i,i+jump),swapped =true;
  }
}
```

Алгоритм сортування Шелла

Сортування Шелла – це алгоритм сортування, що є узагальненням сортування вставкою.

Алгоритм базується на двох тезах:

– Сортування включенням ефективне для майже впорядкованих масивів.

– Сортування включенням неефективне, тому що переміщує елемент тільки на одну позицію за раз.

Тому сортування Шелла виконує декілька впорядкувань включенням, кожен раз порівнюючи і переставляючи елементи, що знаходяться на різній відстані один від одного.

Сортування Шелла не є стійким.

Сортування Шелла названо на честь автора – Дональда Шелла, який опублікував цей алгоритм у 1959 році. В деяких пізніших друкованих виданнях алгоритм називають *сортуванням Шелла-Мацнера*, за ім'ям Нортона Мацнера. Але сам Мацнер стверджував: «Мені не довелось нічого робити з цим алгоритмом, і моє ім'я не має пов'язуватись з ним».

Ідея алгоритму

На початку обираються *m*-елементів: d_1, d_2, … , d_m, причому, $d_1 > d_2 > \ldots > d_m = 1$.

Потім виконується *m* впорядкувань методом включення, спочатку для елементів, що стоять через d_1, потім для елементів через d_2 і т. д. до $d_m = 1$.

Ефективність досягається тим, що кожне наступне впорядкування вимагає меншої кількості перестановок, оскільки деякі елементи вже стали на свої місця.

Псевдокод алгоритму сортування Шелла

Сам алгоритм не залежить від вибору *m* і *d*, тому будемо вважати, що вони задані.

```
Shell_Sort(A,N)
 begin
   for k = 1 to m do
       for i = d[k]+1 to N do
        begin
            a = A[i]
            j = i
            while  j-d[k] >= 1 i A[j-d[k]] > a do
             begin
                A[j] = A[j-d[k]]
                j = j – d[k]
                A[j] = a
             end
        end
 end
```

Коректність алгоритму

Оскільки $d_m = 1$, то на останньому кроці виконується звичайне впорядкування включенням всього масиву, а отже кінцевий масив буде впорядкованим.

Час роботи

Час роботи залежить від вибору значень елементів масиву *d*. Існує декілька підходів вибору цих значень:

- При $d_1 = \left\lfloor \frac{N}{2} \right\rfloor, d_2 = \left\lfloor \frac{d_1}{2} \right\rfloor, d_3 = \left\lfloor \frac{d_2}{2} \right\rfloor, \dots, d_m = 1$ виборі час роботи алгоритму, в найгіршому випадку, є $O(N^2)$.

- Якщо d – впорядкований за спаданням набір чисел виду $\frac{3^j-1}{2}$, $j \in N$, $d_i < N$, то час роботи є $O(N^{\frac{3}{2}})$.
- Якщо d – впорядкований за спаданням набір чисел виду $2^i 3^j$, $i, j \in N$, $d_k < N$, то час роботи є $O(N \log^2 N)$.

Приклад роботи

Проілюструємо роботу алгоритму на вхідному масиві *A* = (5, 16, 1, 32, 44, 3, 16, 7), *d* = (5,3,1).

1. Масив після впорядкування з кроком в 5: (*3*, **16**, 1, 32, 44, *5*, **16**, 7) – зроблено 1 обмін.
2. Масив після впорядкування з кроком 3: (*3*, **7**, 1, *16*, **16**, 5, *32*, **44**) – зроблено 3 обміну.
3. Масив після впорядкування з кроком 1: (1, 3, 5, 7, 16, 16, 32, 44) – зроблено 5 обмінів.

Отже, весь масив впорядковано за 9 операцій обміну.

Питання до розділу:

1. Поясніть терміни "кролики" і "черепахи".
2. Чим відрізняється алгоритм сортування змішуванням від алгоритму сортування бульбашками?
3. Чому сортування змішуванням удвічі швидше за сортування бульбашками?
4. Поясніть алгоритм сортування гребінцем.
5. Що таке фактор зменшення і яким чином він використовується в алгоритмі сортування Шелла?
6. Графічно проілюструйте принцип сортування масиву чисел *A* = (7,58,10,18,96,5,13,8), *d* = (5,3,1) в порядку зростання за методом Шелла.

Розділ 5. Швидке сортування

Швидке сортування Хоара – удосконалений метод сортування, що базується на обміні. К.Хоар запропонував алгоритм QuickSort сортування масивів, що дає на практиці відмінні результати і дуже просто програмується. Це сортування називають *швидким*, тому що на практиці воно виявляється найшвидшим методом сортування з тих, що оперують порівняннями.

Основна стратегія прискорення алгоритмів сортування – обмін між якомога більш віддаленими елементами вихідного файлу.

Ідея К. Хоара полягає в наступному: на кожному кроці методу ми спочатку вибираємо "середній" елемент, потім переставляємо елементи масиву так, що він поділяється на три частини: спочатку ідуть елементи, менші "середнього", потім рівні йому, а в третій частині – більші. Після такого розподілу масиву залишається тільки відсортувати першу і третю його частини, з якими ми зробимо аналогічно (розділимо на три частини). І так доти, поки ці частини не будуть складатися з одного елементу, а масив з одного елементу завжди відсортований.

Дано	**17**	35	48	52	27	9	15	**13**	89
1-й обмін	13	**35**	48	52	27	9	15	**17**	89
2-й обмін	13	**17**	18	52	27	9	**15**	35	89
3-й обмін	13	15	**18**	52	27	9	**17**	35	89
4-й обмін	13	15	**17**	52	27	**9**	18	35	89
5-й обмін	13	15	9	**52**	27	**17**	18	35	89
6-й обмін	13	15	9	**17**	27	52	18	35	89

Рисунок 5.1 – Ілюстрація роботи швидкого сортування

Вибір "середнього" – задача непроста, тому що потрібно, не виконуючи сортування, знайти елемент зі значенням максимально близьким до середнього. Тут, звичайно, можна просто вибрати довільний елемент (звичайно вибирають елемент, що стоїть посередині підмасиву, що сортується), але можемо вибирати з трьох елементів самого лівого, самого правого і того, що стоїть посередині.

Складність:

Аналіз складності алгоритму в середньому, що використовує гіпотезу про рівну імовірність усіх входів, показує, що:

$$C(n) = O(n \log^2 n),\ M(n) = O(n \log^2 n).$$

У гіршому випадку, коли в якості базового вибирається, наприклад, максимальний елемент підмасиву, складність алгоритму квадратична.

Швидке сортування є алгоритмом на основі порівнянь, і не є стійким.

Класична реалізація

В класичному варіанті, запропонованому Хоаром, з масиву обирався один елемент, і весь масив розбивався на дві частини по принципу: в першій частині – ті що не більші даного елементу, в другій частині – ті що не менші даного елементу. Процедура *Quicksort*(*A*,*p*,*q*) здійснює часткове впорядкування масиву *A* з *p*-го по *q*-й індекс, псевдокод:

```
Quicksort(A, p, q)
 begin
    if    p >= q   return
    r = A[p]
    i = p - 1
    j = q + 1
    while i < j  do
        repeat
             i = i + 1
        until A[i] >= r
```

```
        repeat
            j = j - 1
        until A[i] <= r
        if i < j then
            // Поміняти місцями значення A[i]та A[j]
        Quicksort(A, p, j)
        Quicksort(A, j + 1, q)
 end
```

Сучасна реалізація

На сьогодні в стандартних бібліотеках використовують таку реалізацію алгоритму, псевдокод:

```
Partition(A, p, q)
 begin
    x  =  A[q]
    i  =  p - 1
    for  i  =  p   to   q - 1  do
    if  A[j] <= x
        then  i = i + 1
            // Поміняти місцями значення A[i] та A[j]
    i = i + 1
    // Поміняти місцями значення A[i]та A[q]
    return i
end

Quicksort(A, p, q)
begin
    if p >= q return
    i = Partition(A, p, q)
    Quicksort(A, p, i  -1)
    Quicksort(A, i + 1, q)
end
```

Аналіз

Час роботи алгоритму сортування залежить від збалансованості, що характеризує розбиття. Збалансованість, у свою чергу залежить від того, який елемент обрано як опорний (відносно якого елементу виконується розбиття). Якщо розбиття збалансоване, то асимптотично алгоритм

працює так само швидко як і алгоритм сортування злиттям. У найгіршому випадку, асимптотична поведінка алгоритму настільки ж погана, як і в алгоритмі сортування включенням.

Найгірше розбиття

Найгірша поведінка має місце у тому випадку, коли процедура, що виконує розбиття, породжує одну підзадачу з n-1 елементами, а другу – з 0 елементами. Нехай таке незбалансоване розбиття виникає при кожному рекурсивному виклику. Для самого розбиття потрібен час *O(n)*. Тоді, рекурентне співвідношення для часу роботи, можна записати так:

$$T(n) = T(n - 1) + T(0) + O(n) = T(n - 1) + O(n)$$

Розв'язком такого співвідношення є $T(n) = O(n^2)$.

Найкраще розбиття

В найкращому випадку процедура *Partition* ділить задачу на дві підзадачі, розмір кожної не перевищує *n/2*. Час роботи, описується нерівністю:

$$T(n) \leq 2T(n / 2) + O(n)$$

Тоді:

$T(n) = O(n \log n)$ – асимптотично найкращий час.

Середній випадок

Математичне очікування часу роботи алгоритму для всіх можливих вхідних масивів є $O(n \log n)$, тобто середній випадок ближчий до найкращого.

Модифікації

В середньому алгоритм працює дуже швидко, але на практиці, всі можливі вхідні масиви мають не однакову імовірність. Тоді, шляхом додання рандомізації вдається отримати середній час роботи в будь-якому випадку.

Рандомізований алгоритм

В рандомізованному алгоритмі, при кожному розбитті випадковий елемент обирається в якості опорного, псевдокод:

```
Randomized_Partition(A, p, q)
 begin
    i = Random(p, q)
    // Поміняти місцями значення A[i]та A[q]
    return Partition(A, p, q)
 end

Randomized_Partition(A, p, q)
 begin
    if p >= q return
    i = Randomized_Partition(A, p, q)
    Randomized_Quicksort(A, p, i - 1)
    Randomized_Quicksort(A, i + 1, q)
 end
```

Питання до розділу:

1. Чому сортування Хоара називають швидким?
2. Поясніть основну ідею Хоара.
3. Чим сучасна реалізація сортування Хоара відрізняється від класичної?
4. Яку складність має алгоритм сортування Хоара і від чого вона залежить?
5. Що дає рандомізація?