Алгоритми сортування масивів

Мета лекції

- Ознайомитись з класифікацією алгоритмів сортувань.
- Ознайомитись з прямими (простими) та швидкими алгоритмами **внутрішнього** сортування.
- Ознакомитись з алгоритмами **зовнішнього сортування**.
- Навчитись аналізувати часову складність алгоритмів сортування.

Чому задача сортування є важливою?

Дозволяє полегшити подальшу обробку даних. Один із прикладів - задача пошуку. Так, одним з ефективних алгоритмів пошуку є бінарний пошук. Він працює швидше ніж, наприклад, лінійний пошук, але його можливо застосовувати лише за умови, що послідовність вже упорядкована, тобто відсортована.

Параметри, за якими здійснюється оцінка алгоритмів сортування

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

- **Пам’ять** - один з параметрів, який характеризується тим, що ряд алгоритмів сортування вимагають виділення додаткової пам’яті під тимчасове зберігання даних. При оцінці використовуваної пам’яті не буде враховуватися місце, яке займає вихідний масив даних і не залежать від вхідної послідовності витрат, наприклад, на зберігання коду програми.

- **Стійкість** - це параметр, який відповідає за те, що сортування не змінює взаємного розташування рівних елементів.

- **Природність поведінки** - параметр, який вказує на ефективність методу при обробці вже відсортованих, або частково впорядкованих даних. Алгоритм поводиться природно, якщо враховує цю характеристику вхідної послідовності.

Класифікація алгоритмів сортування

- **Внутрішнє сортування (сортування масивів)** - це алгоритм сортування, який в процесі упорядкування даних використовує тільки оперативну пам’ять (ОЗП). Тобто оперативної пам’яті досить для розміщення в ній вхідного масиву даних. ОЗП дозволяє довільний доступ до будь-якої частини вхідного масиву.

- **Зовнішнє сортування (сортування файлів/послідовностей)** - це алгоритм сортування, який при проведенні упорядкування даних використовує зовнішню пам’ять, як правило, жорсткі диски. Такі сортування розроблені для обробки масивів даних, що не поміщаються в оперативну пам’ять. Доступ до носія часто є послідовним, це накладає певні обмеження.

Класифікація алгоритмів сортування

- **Прямі методи** - найпростіші методи сортування, що потребують порядку O(n^2) порівнянь ключів. Такі методи зручно використовувати на так званих “коротких” масивах.

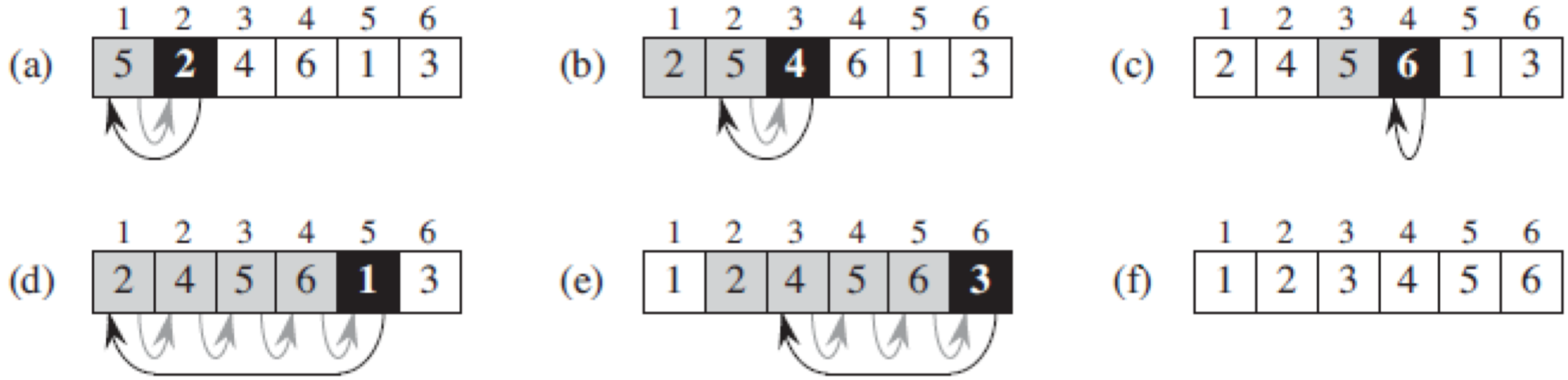
- **Швидкі методи** - методи сортування, що в найкращому випадку вимагають порядку O(n*log2n) порівнянь ключів. Виграш в ефективності для таких алгоритмів можна отримати на “довгих” масивах.

Прямі методи сортування

- Сортування включенням (Insertion sort)
- Сортування вибором (Selection sort)
- Сортування обміном (Bubble sort)

Сортування включенням (Insertion Sort)

Ідея алгоритму: на кожному кроці алгоритму ми вибираємо один з елементів вхідних даних і вставляємо його на потрібну позицію у вже відсортованій частині списку.



Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein
Introduction to Algorithms Third Edition

Сортування включенням (Insertion Sort)

КЛАС	Сортування вставками	
СТАБІЛЬНІСТЬ	Так	
ПОРІВНЯННЯ	Так	
СКЛАДНІСТЬ ЗА ЧАСОМ	НАЙГІРША	$O(n^2 / 2)$
	СЕРЕДНЯ	$O(n^2 / 4)$
	НАЙКРАЩА	$O(n)$
СКЛАДНІСТЬ ЗА ПАМ'ЯТТЮ	ЗАГАЛЬНА	$O(n)$
	ДОДАТКОВА	$O(1)$

Сортування включенням (Insertion Sort)

- **Найкращий сценарій:** якщо вхідний масив вже відсортовано, тоді алгоритм виконує O(n) порівнянь ключів і не виконує переміщень елементів. Тому часова складність алгоритму оцінюється як O(n).

- **Найгірший сценарій:** якщо вхідний масив відсортовано в зворотньому порядку, тоді необхідно (n-1) порівнянь/перестановок для останнього елемента, (n-2) операцій для наступного за останнім і т.д. Тоді маємо арифметичну прогресію $(1 + 2 + \ldots + n - 2 + n - 1)$

Порахуємо суму арифметичної прогресії використовуючи формулу: $\sum_{q=1}^{p} \frac{p(p+1)}{2}$

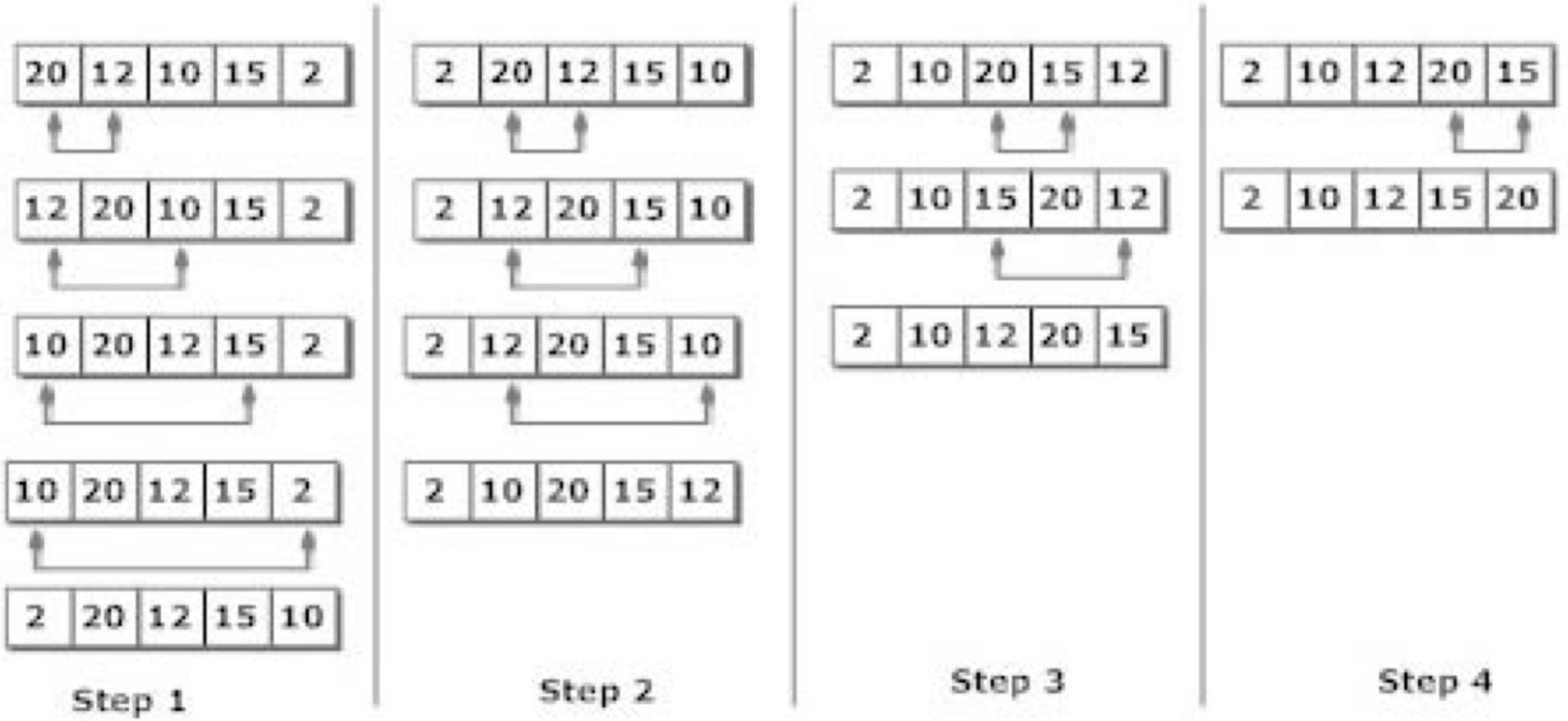
$$\frac{(n-1) * (n-1+1)}{2} = \frac{n^2 - n + n - n + 1 - 1}{2} = \frac{n^2}{2} - \frac{n}{2} \Longrightarrow O(n^2)$$

Сортування вибором (Selection Sort)

Ідея алгоритму:

- Знаходить у списку найменше значення
- Міняє його місцями із першим значеннями у списку
- Повторює два попередніх кроки, доки список не завершиться (починаючи з наступної позиції)

Сортування вибором (Selection Sort)



Сортування вибором (Selection Sort)

КЛАС	Сортування вибором	
СТАБІЛЬНІСТЬ	Ні	
ПОРІВНЯННЯ	Так	
СКЛАДНІТЬ ЗА ЧАСОМ	НАЙГІРША	$O(n^2 / 2)$
	СЕРЕДНЯ	$O(n^2 / 2)$
	НАЙКРАЩА	$O(n^2 / 2)$
СКЛАДНІТЬ ЗА ПАМ'ЯТТЮ	ЗАГАЛЬНА	$O(n)$
	ДОДАТКОВА	$O(1)$

Сортування вибором (Selection Sort)

Сортування вибором не є складним в аналізі та порівнянні його з іншими алгоритмами, оскільки жоден з циклів не залежить від даних у списку. Знаходження найменшого елементу вимагає перегляду усіх n елементів (у цьому випадку n – 1 порівняння), і після цього, перестановки його до першої позиції. Знаходження наступного найменшого елементу вимагає перегляду n – 1 елементів, і так далі, для $(n-1)+(n-2)+\ldots+2+1=\frac{n(n-1)}{2} \in$ Θ(n2) порівнянь (сума арифметичної прогресії). Кожне сканування вимагає однієї перестановки для n – 1 елементів (останній елемент знаходитиметься на своєму місці).

Сортування бульбашкою

КЛАС	Сортування бульбашкою	
СТАБІЛЬНІСТЬ	Так	
ПОРІВНЯННЯ	Так	
СКЛАДНІСТЬ ЗА ЧАСОМ	НАЙГІРША	$O(n^2)$
	СЕРЕДНЯ	$O(n^2)$
	НАЙКРАЩА	$O(n)$
СКЛАДНІСТЬ ЗА ПАМ'ЯТТЮ	ЗАГАЛЬНА	$O(n)$
	ДОДАТКОВА	$O(1)$

Швидкі методи сортування

- Сортування Шелла
- Сортування злиттям (Merge Sort)
- Алгоритм Хоара (Quick Sort)
- Пірамідальне сортування (Heap Sort)
- Сортування підрахунком (Counting Sort)

Сортування Шелла

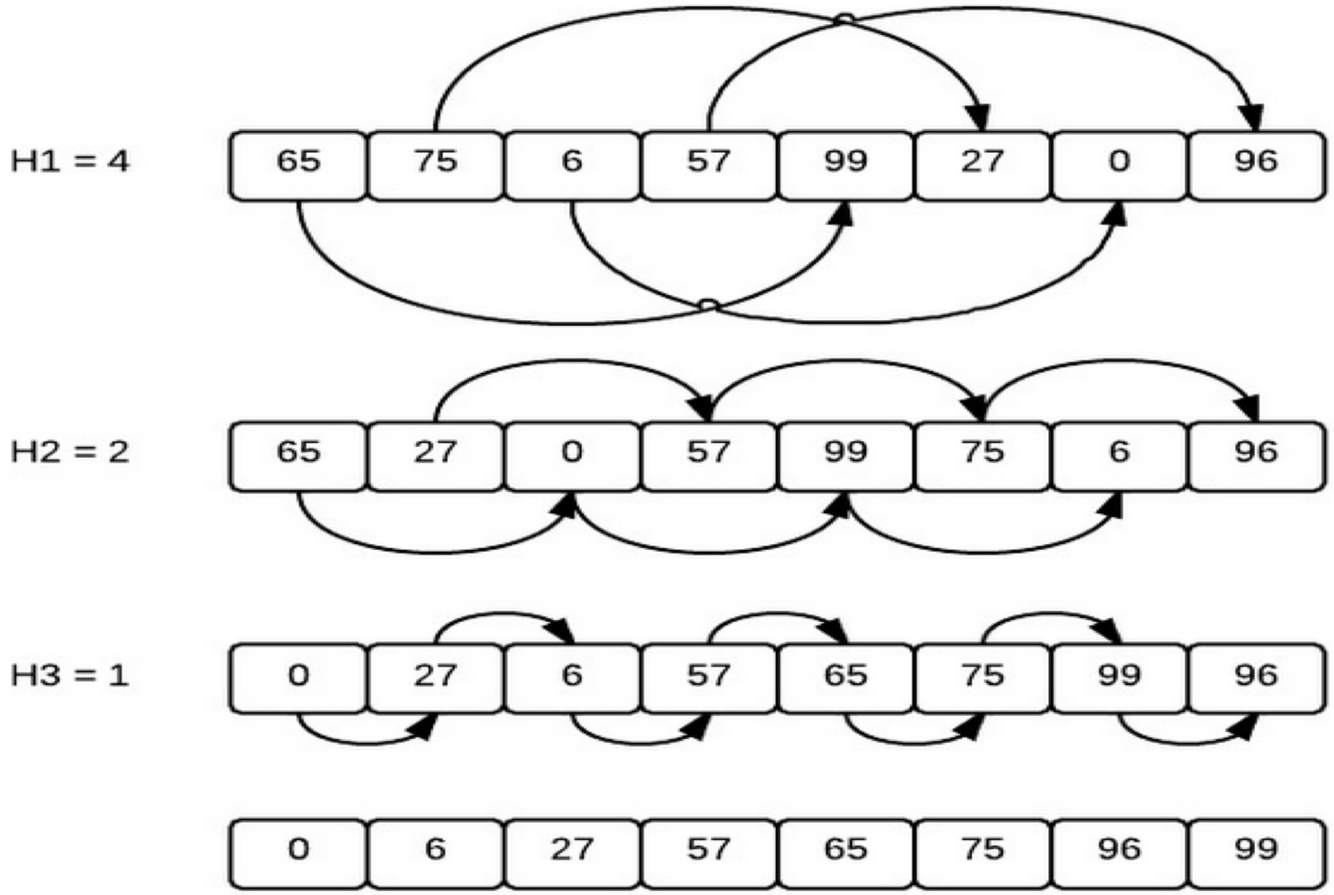
Сортування Шелла — це алгоритм сортування, що є покращеною версією сортування включенням. Алгоритм базується на двох тезах:

- Сортування вставкою ефективне для частково впорядкованих масивів.
- Сортування вставкою неефективне, тому що переміщує елемент тільки на одну позицію за один раз.

Ідея алгоритму: на початку обираються m-елементів: $d_1, d_2, \ldots, d_m$, причому $d_1 > d_2 > \ldots > d_m = 1.$

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

Сортування Шелла



Сортування Шелла

КЛАС	Сортування вставками	
СТАБІЛЬНІСТЬ	Нет	
ПОРІВНЯННЯ	Так	
СКЛАДНІСТЬ ЗА ЧАСОМ	НАЙГІРША	Залежить від кроку
	СЕРЕДНЯ	
	НАЙКРАЩА	**O(n)**
СКЛАДНІСТЬ ЗА ПАМ'ЯТТЮ	ЗАГАЛЬНА	**O(n)**
	ДОДАТКОВА	**O(1)**

Сортування Шелла

Оцінка складності алгоритму залежить від вибору кроку D:

1. Найпростіший приклад - D = n/2, D2 = D/2…,Dn = 1. В найгіршому випадку складність алгоритму $O(n) = n^2$

2. Пропозиція Хіббарда: перевірити на всіх $N^i - 1 <= N$. В такому випадку складність алгоритму $O(n) = n^{\frac{3}{2}}$

3. Числа Седжвіка $d_i = 9 * 2^i - 9 * 2^{\frac{i}{2}} + 1$ якщо і -парне,

$d_i = 8 * 2^i - 6 * 2^{\frac{i+1}{2}} + 1$ якщо і - непарне.

4. Метод Кнута $(3^k - 1)/2$.

Найкращий сценарій O(n) - коли вхідний масив вже відсортований.

Основна теорема про рекурентні співвідношення (Master theorem)

$$T(n) = a * T\left(\frac{n}{b}\right) + f(n)$$

a - кількість підзадач

b - розмір однієї підзадачі

n - розмір вхідних даних

$T(n)$ - час виконання всього алгоритму

$T\left(\frac{n}{b}\right)$ - час виконання однієї підзадачі

$f(n)$ - час об’єднання результатів

Теорема працює лише у випадку коли $f(n)$ має поліноміальну форму.

Сортування злиттям (Merge Sort)

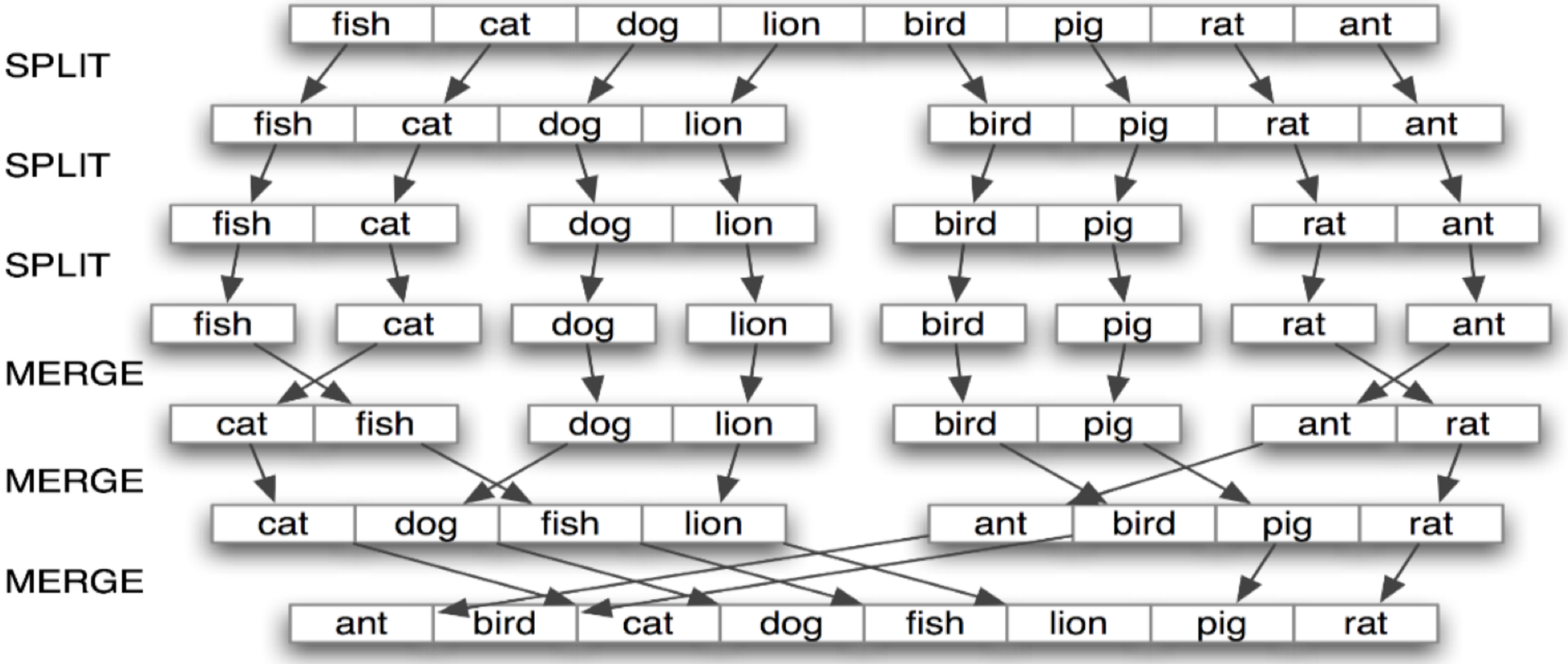
Ідея алгоритму: алгоритм сортування, в основі якого лежить принцип «Розділяй та володарюй». В основі цього способу сортування лежить злиття двох упорядкованих ділянок масиву в одну впорядковану ділянку іншого масиву.

1. Вхідний масив розбивається на дві частини приблизно однакового розміру. Розбиття відбувається доки розмір масиву не дорівнюватиме 1.

2. Кожна з отриманих частин рекурсивно сортується.

3. Два впорядкованих масива зливається в один. На кожному кроці ми беремо менший з двох елементів підмасивів і записуємо в результуючий масив. Лічильники номерів елементів результуючого масиву і підмасиву, звідки було взято елемент, збільшуємо на 1.

Сортування злиттям (Merge Sort)



Сортування злиттям (Merge Sort)

Основной недолік: потребує O(n) додаткової пам’яті.

КЛАС	Сортування злиттям	
СТАБІЛЬНІСТЬ	Так	
ПОРІВНЯННЯ	Так	
СКЛАДНІСТЬ ЗА ЧАСОМ	НАЙГІРША	**O(n log n)**
	СЕРЕДНЯ	**O(n log n)**
	НАЙКРАЩА	**O(n log n)**
СКЛАДНІСТЬ ЗА ПАМ'ЯТТЮ	ЗАГАЛЬНА	**O(n)**
	ДОДАТКОВА	**O(n)**

Сортування злиттям (Merge Sort)

Найгіршний випадок = Середній випадок = Найкращий випадок = O(nlogn)

Задача ділиться на дві підзадачі рівного розміру $\frac{n}{2}$:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \text{ (1)}$$

Час об’єднання двох підзадач лінійний - O(n).

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2} \text{ (2)}$$

Підставимо (2) в (1):

$$T(n) = 2 * \left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \text{ (3)}$$

$$T(n) = 2^2T\left(\frac{n}{2^2}\right) + 2n \text{ (4)}$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4} \text{ (5)}$$

Підставимо (5) в (4):

$$T(n) = 2^3T\left(\frac{n}{2^3}\right) + 3n \text{ (6)}$$

Отримали загальний вираз:

$$T(n) = 2^iT\left(\frac{n}{2^i}\right) + in$$

Вирішити задачу з одним елементом можливо за константу (масив з одного елементу вже відсортований): T(1) = 1

$$\frac{n}{2^i} = 1 \longrightarrow n = 2^i$$

$$log_2n = log_22^i = ilog_22 = i$$

$$T(n) = nT(1) + nlog_2n = n + nlog_2n = O(nlog_2n)$$

Алгоритм Гоара (Quick Sort)

Ідея алгоритму: полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин відбувається рекурсивно.

Метод відноситься до **групи методів обміну**, тобто до тієї ж, що і метод сортування бульбашкою.

Алгоритм Гоара (Quick Sort)

Клас	Сортування на основі обміну
Найгірша швидкодія	$O(n^2)$
Найкраща швидкодія	$O(nlogn)$
Середня швидкодія	$O(nlogn)$
Стабільність	Ні

Алгоритм Гоара (Quick Sort)

Використано розбиття Ломуто:

```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p - 1)
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo
    for j := lo to hi do
        if A[j] < pivot then
            swap A[i] with A[j]
            i := i + 1
    swap A[i] with A[hi]
    return i
```

Алгоритм Гоара (Quick Sort)

Час роботи алгоритму сортування залежить від збалансованості, що характеризує розбиття. Збалансованість, у свою чергу залежить від того, який елемент обрано як опорний (відносно якого елемента виконується розбиття).

Найкращий сценарій: В найкращому випадку процедура Partition ділить задачу на дві підзадачі, розмір кожної не перевищує n/2. Маємо наступне рекуренте співвідношення:

$$T(n) = 2T(n/2) + O(n)$$

Дане співвідношення було вирішене для Merge Sort, $T(n) = O(n * logn)$ - асимптотично найкращий час.

Алгоритм Гоара (Quick Sort)

Найгірший сценарій: найгірша поведінка має місце у тому випадку, коли процедура, що виконує розбиття, породжує одну підзадачу з n-1 елементом, 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:

$$T(n) = T(n-1) + T(1) + O(n) = T(n-1) + O(n) \text{ (1)}$$

$$T(n-1) = T(n-2) + (n-1) \text{ (2)}$$

Підставимо вираз (2) в (1):

$$T(n) = T(n-2) + (n-1) + n \text{ (3)}$$

З виразу (1): $T(n-2) = T(n-3) + (n-2)$ (4)

Підставимо (4) в (1): $T(n) = T(n-3) + (n-2) + (n-1) + n$

З цього отримуємо загальний вираз: $T(n) = T(1) + 2 + 3 + 4 + \ldots + (n-1) + n$

$$T(n) = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2} = O(n^2)$$

Пірамідальне сортування (Heap Sort)

Ідея алгоритму: сортування пірамідою використовує бінарне сортувальне дерево. Сортувальне дерево — це таке дерево, у якого виконані умови:

- Кожен лист має глибину d або $d-1$, d — максимальна глибина дерева.
- Значення в будь-якій вершині не менші (інший варіант — не більші) за значення їх нащадків.

Пірамідальне сортування (Heap Sort)

```
Input data: 4, 10, 3, 5, 1
         4(0)
        /   \
     10(1)   3(2)
    /   \
 5(3)    1(4)

The numbers in bracket represent the indices in the array
representation of data.

Applying heapify procedure to index 1:
         4(0)
        /   \
    10(1)    3(2)
    /   \
5(3)    1(4)

Applying heapify procedure to index 0:
        10(0)
        /  \
     5(1)  3(2)
    /   \
 4(3)    1(4)
The heapify procedure calls itself recursively to build heap
 in top down manner.
```

Пірамідальне сортування (Heap Sort)

Кроки алгоритму:

- Побудова max_heap з вхідного масиву даних. Виконується крок для $[0,n/2-1]$ елементів масиву. В найгіршому випадку кожен елемент буде переміщатись $log(n)$ разів (висота дерева). Найгірший випадок на цьому кроці оцінюється як $O(n/2 * log(n))$

- Крок сортування з видаленням кореневого вузла і виклик heapify процедури виконується для n елементів. Складність O(n*log(n))

$$O(n/2 * log(n)) + O(n * log(n)) \longrightarrow O(n * logn)$$

Перевага алгоритму: оцінка в найгіршому випадку, середньому і найкращому - однакова.

Недоліки алгоритму: нестійкий, на майже відсортованих даних працює так само довго, як і на хаотичних даних, в середньому працює повільніше за Quick Sort.

Цікавий факт: алгоритм використовується в Linux Kernel тому що була потрібна гарантія O(n*logn) складності у всіх випадках.

Пірамідальне сортування (Heap Sort)

КЛАС	Сортування вибором	
СТАБІЛЬНІСТЬ	Ні	
ПОРІВНЯННЯ	Так	
СКЛАДНІТЬ ЗА ЧАСОМ	НАЙКРАЩА	**O** (**n** log **n**)
	СЕРЕДНЯ	
	НАЙГІРША	
СКЛАДНІТЬ ЗА ПАМ'ЯТТЮ	ЗАГАЛЬНА	**O**(**n**)
	ДОДАТКОВА	**O**(1)

Сортування підрахунком (Counting Sort)

Алгоритм впорядкування, що застосовується при малій кількості різних елементів (ключів) у масиві даних. Час його роботи лінійно залежить як від загальної кількості елементів у масиві так і від кількості різних елементів.

Ідея алгоритму: спочатку підрахувати скільки разів кожен елемент (ключ) зустрічається в вихідному масиві. Спираючись на ці дані можна одразу вирахувати на якому місці має стояти кожен елемент, а потім за один прохід поставити всі елементи на свої місця.

Сортування підрахунком (Counting Sort)

For simplicity, consider the data in the range 0 to 9.
Input data: 1, 4, 1, 2, 7, 5, 2
1) Take a count array to store the count of each unique object.

Index:	0	1	2	3	4	5	6	7	8	9
Count:	0	2	2	0	1	1	0	1	0	0

2) Modify the count array such that each element at each index stores the sum of previous counts.

Index:	0	1	2	3	4	5	6	7	8	9
Count:	0	2	4	4	5	6	6	7	7	7

The modified count array indicates the position of each object in the output sequence.

3) Output each object from the input sequence followed by decreasing its count by 1.
Process the input data: 1, 4, 1, 2, 7, 5, 2. Position of 1 is 2.
Put data 1 at index 2 in output. Decrease count by 1 to place next data 1 at an index 1 smaller than this index.

Сортування підрахунком (Counting Sort)

- Складність алгоритму лінійно залежить від кількості вхідних елементів n і кількості унікальних елементів даного масиву k => $O(n + k)$ у всіх випадках.

- Просторова складність $O(n + k)$ тому що потребує створення додаткового масиву розміром k.

- Стабільність: так.

Зовнішні методи сортування

- Сортування злиттям (просте злиття і природне злиття)
- Поліпшені сортування (багатофазне сортування і каскадне сортування).

Зовнішні методи сортування
Основні поняття

- **Зовнішнє сортування** - це сортування даних, які розташовані на зовнішніх пристроях і не вміщаються в оперативну пам’ять.
- **Двоколійне злиття** - це сортування, в якій дані розподіляються на два допоміжних файла.
- **Двофазне сортування** - це сортування, в якій окремо реалізуються дві фази: розподіл і злиття.
- **Довжина серії** - це кількість елементів в серії.
- **Природне злиття** - це сортування, при якій завжди зливаються дві найдовші з можливих серій.
- **Багатоколійне злиття** - це сортування, в якій дані розподіляються на **N** (N> 2) допоміжних файлів.
- **Незбалансоване злиття** - це природне злиття, у якого після фази розподілу кількість серій у допоміжних файлах відрізняється один від одного більш ніж на одиницю.
- **Збалансоване злиття** - це природнє злиття, у якого після фази розподілу кількість серій у допоміжних файлах відрізняється один від одного не більше ніж на одиницю.
- **Однофазне сортування** - це сортування, в якій об'єднані фази розподілу і злиття в одну.
- **Просте злиття** - це один з видів сортувань на основі злиття, в якій довжина серій фіксується на кожному кроці.
- **Розподіл** - це процес поділу упорядкованих серій на два і кілька допоміжних файлів.
- **Серія** (упорядкований відрізок) - це послідовність елементів, яка впорядкована по ключу.
- **Злиття** - це процес об'єднання двох (або більше) упорядкованих серій в одну впорядковану послідовність за допомогою циклічного вибору елементів доступних в даний момент.
- **Фаза** - це дії по одноразовій обробці всієї послідовності елементів.

Сортування злиттям

- Спочатку серії розподіляються на два або більше допоміжних файла.
- Даний розподіл йде по черзі: перша серія записується в перший допоміжний файл, друга - в другий і так далі до останнього допоміжного файлу. Потім знову запис серії починається в перший допоміжний файл. Після розподілу всіх серій, вони об’єднуються в довші впорядковані відрізки, тобто з кожного допоміжного файлу береться по одній серії, які зливаються.
- Якщо в якомусь файлі серія закінчується, то перехід до наступної серії не провадиться. Залежно від виду сортування формована довша впорядкована серія записується або в вихідний файл, або в один з допоміжних файлів.
- Після того як всі серії з усіх допоміжних файлів об’єднані в нові серії, потім знову починається їх розподіл. І так до тих пір, поки всі дані не будуть відсортовані.

Основні характеристики

- Кількість фаз в реалізації сортування.
- Кількість допоміжних файлів, на які розподіляються серії.

Сортування простим злиттям (двофазне)

Алгоритм сортування простим злиття є найпростішим алгоритмом зовнішнього сортування, заснований на процедурі злиття серією.

В даному алгоритмі довжина серій фіксується на кожному кроці. У вихідному файлі всі серії мають довжину 1, після першого кроку вона дорівнює 2, після другого - 4, після третього - 8, після k-го кроку - 2^k.

Отже процес сортування простим злиттям вимагає порядку $O(logn)$ проходів за даними і $O(n)$ операцій для злиття на кожному кроці.

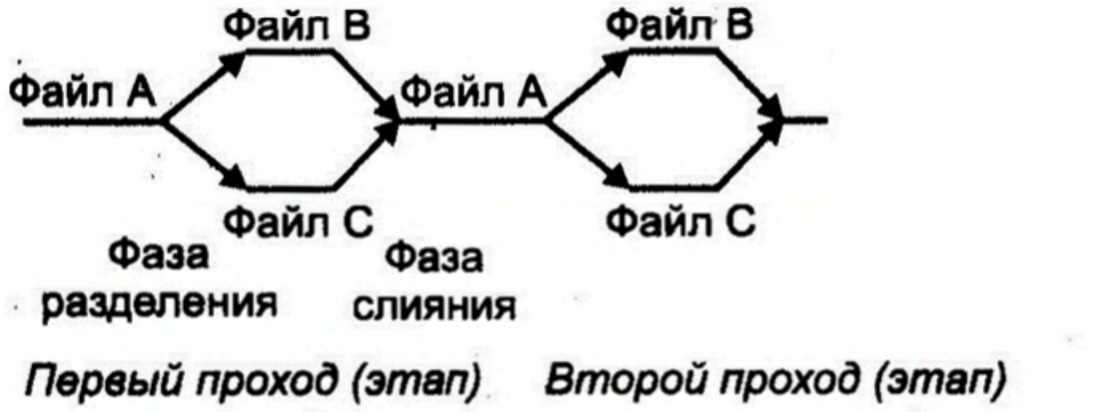


Рис. 5.4. Двухфазная сортировка прямым слиянием

Сортування простим злиттям (однофазне)

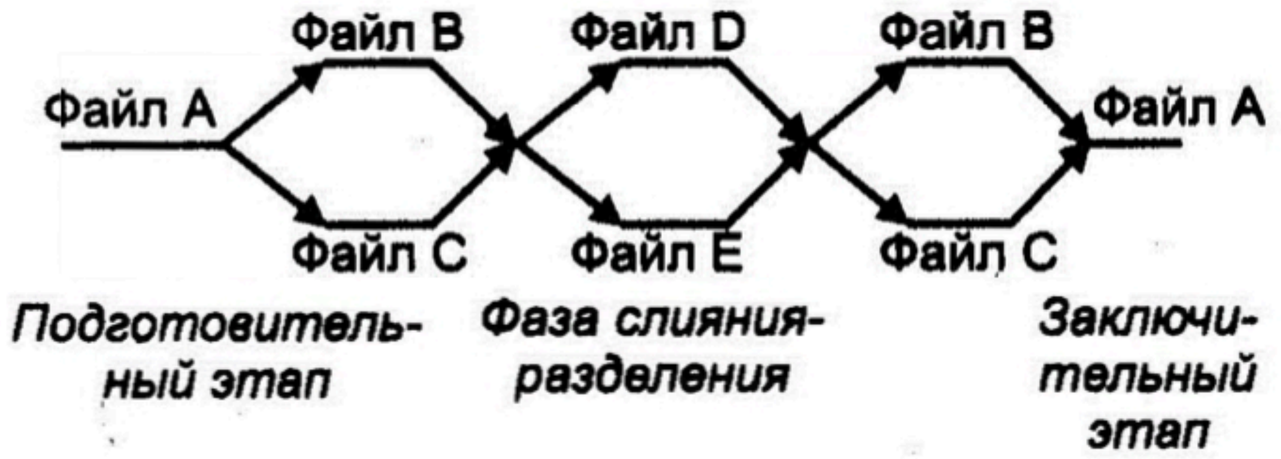


Рис. 5.5. Однофазная сортировка прямым слиянием

Сортування простим злиттям

Исходный файл f: **2 3 17 7 8 9 1 4 6 9 2 3 1 18**

	Распределение	***Слияние***
1 проход	***f1*****: 2 3 17 ` 1 4 6 9 ` 1 18** ***f2*****: 7 8 9 ` 2 3**	***f*****: 2 3 7 8 9 17 1 2 3 4 6 9 1 18**
2 проход	***f1*****: 2 3 7 8 9 17 ` 1 18** ***f2*****: 1 2 3 4 6 9 `**	***f*****: 1 2 2 3 3 4 6 7 8 9 9 17 1 18**
3 проход	***f1*****: 1 2 2 3 3 4 6 7 8 9 9 17** ***f2*****: 1 18**	***f*****: 1 1 2 2 3 3 4 6 7 8 9 9 17 18**

Сортування природним злиттям

Сортування прямим злиттям не враховує, чи були дані вже частково впорядковані чи ні.

Сортування, при якій зливаються дві чергові впорядковані під послідовності (серії), називаються природним злиттям.

Збалансоване багатоколійне злиття

Один із способів скоротити число пересилань це розподілити серії в досить велику кількість файлів. Якщо у вихідному файлі є **n** серій і **N** файлів.Тоді перший прохід дасть n / N серій, другий прохід - n / N^2, третій - n / N^3 і так далі…

Загальна кількість проходів, необхідне для сортування N елементів дорівнює

$k = \log_N n$, а число пересилань при обєднанні фаз злиття і розділення в найгіршому випадку буде

$M = n * \log_N n$

Домашнє завдання

- Яке сортування використовується в вашій улюбленій мові програмування?
- Аналіз складності сортування Шелла для різних алгоритмів вибору інтервалів.
- В ряді алгоритмів сортування пропущено аналіз середнього випадку, проаналізувати самостійно.