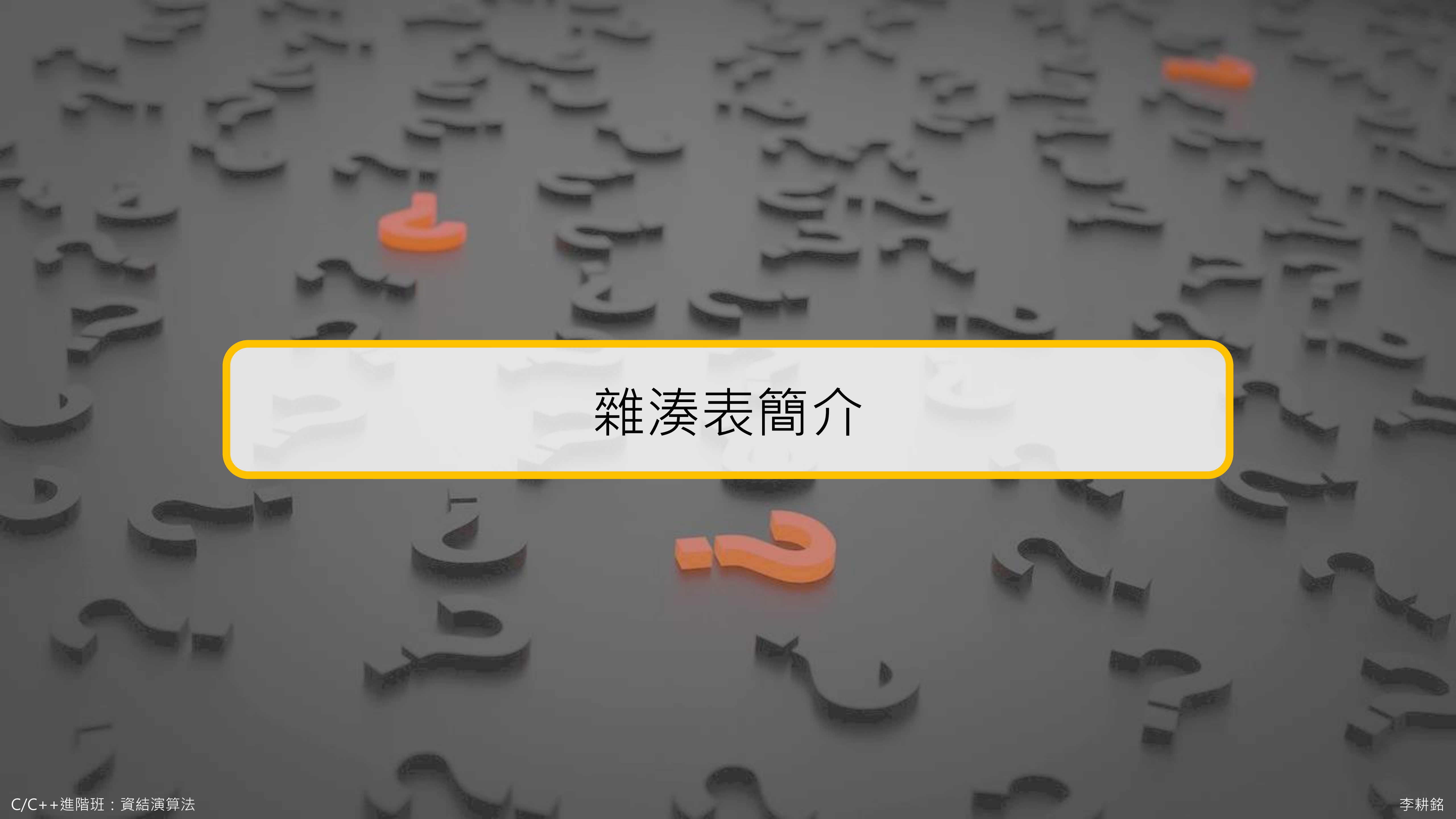
C/C++ 進階班
資料結構

雜湊表
(Hash table)

李耕銘

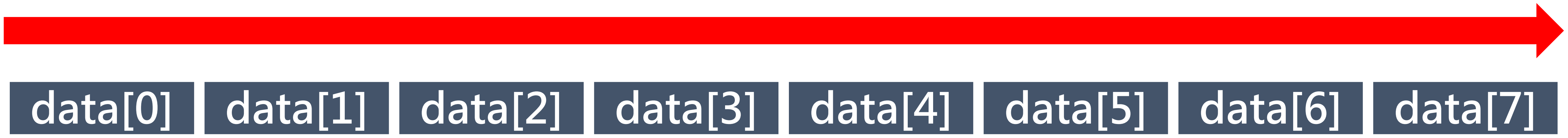
課程大綱

- 雜湊表簡介
    - 雜湊函式(Hash Function)
- 實作雜湊表
- 碰撞(Collision)處理
- 雜湊表的挑戰
- Map 與 Dict



雜湊表簡介

雜湊表簡介



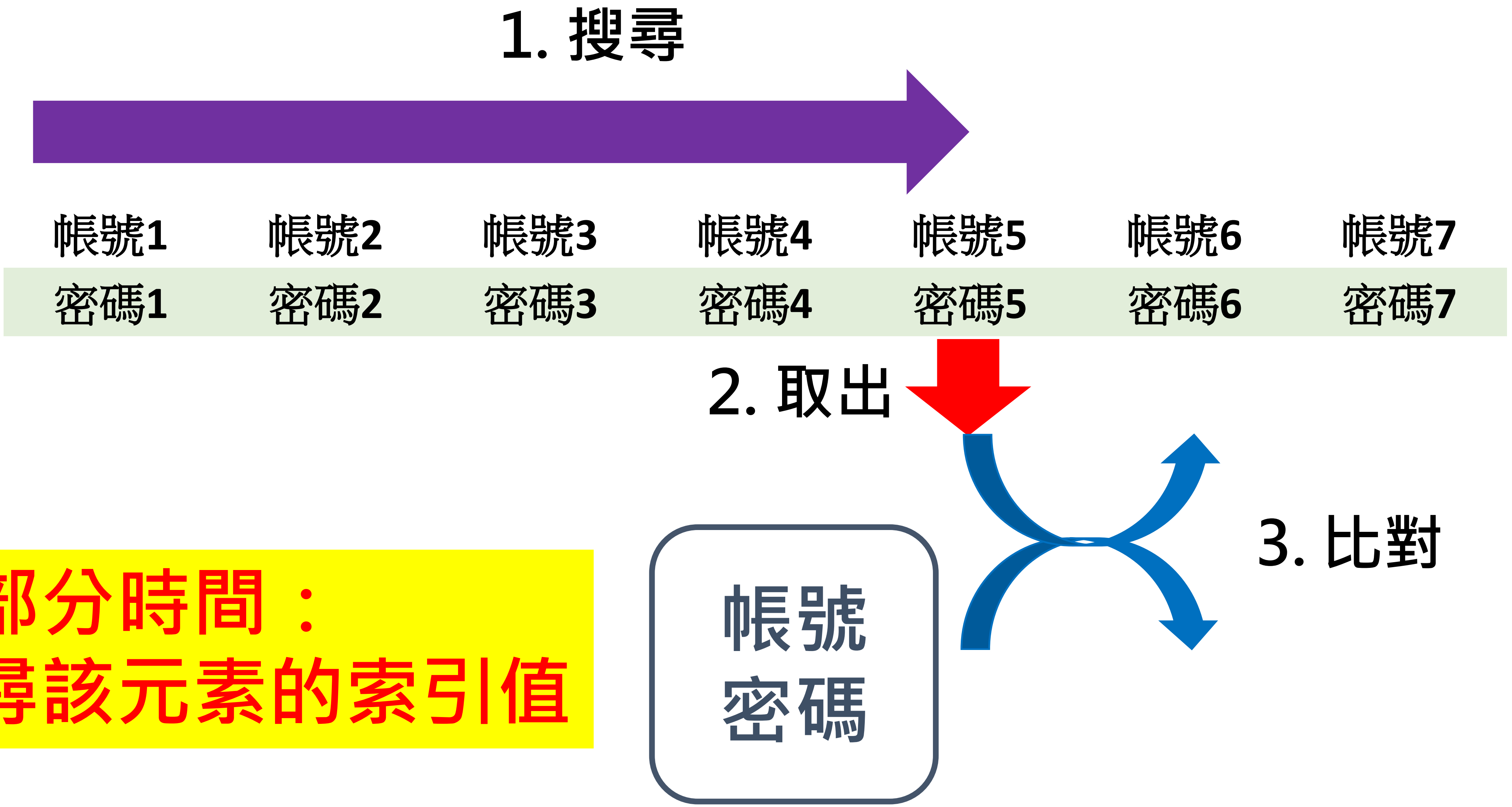
陣列

- 搜尋特定元素
  - O(N)

```
int search(int *p, int len, int value)
{
    for(int i=0;i<len;i++){
        if(*(p+i)==value)
            return i;
    }
    return -1;
}
```

大部分時間在搜尋該元素的索引值

雜湊表簡介



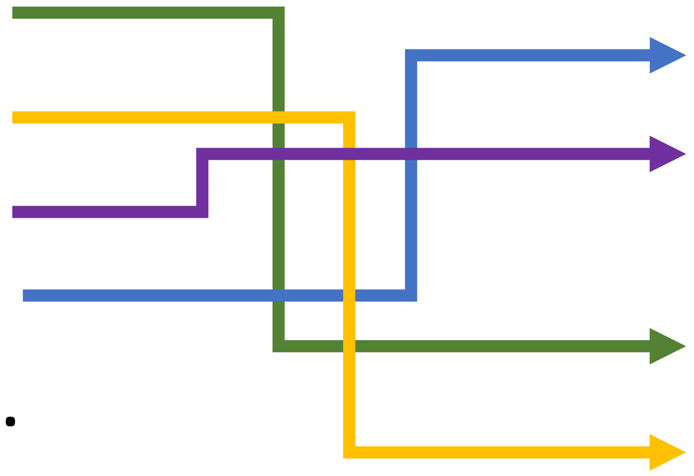
大部分時間：
搜尋該元素的索引值

雜湊表簡介

如果可以直接轉換：

查詢的值(Value) →索引值(Index)

Rallod	→	4
Andrew	→	5
Tom	→	2
John	→	1
……	→	……

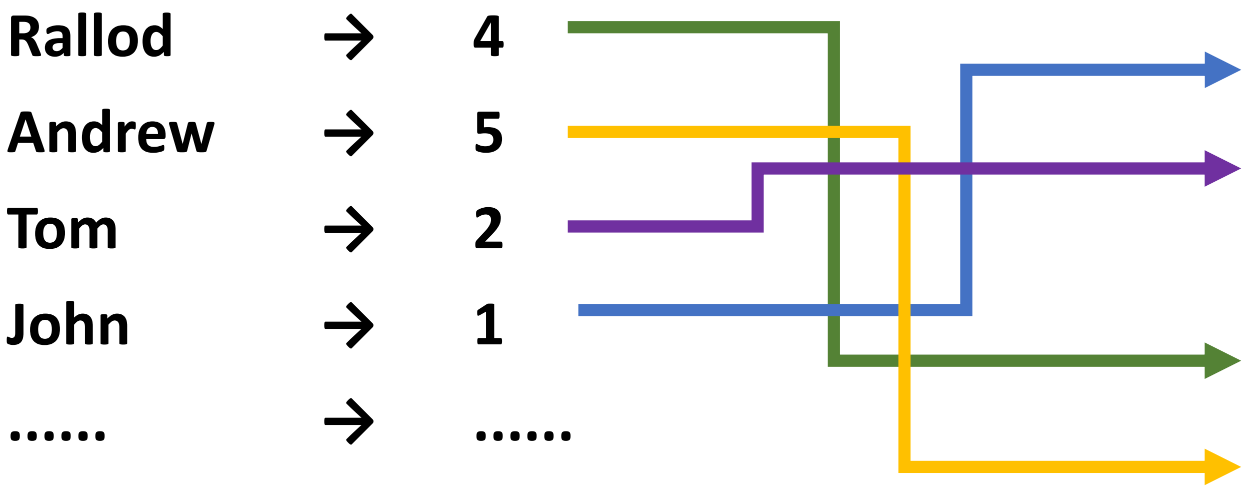


Index	Name	Score
0	David	83
1	John	95
2	Tom	82
3	Sherry	61
4	Rallod	78
5	Andrew	85
6	Helen	96
…	…	…

所有的搜尋、新增、刪除只需要O(1)

雜湊表簡介

$$index = hash(Name)$$



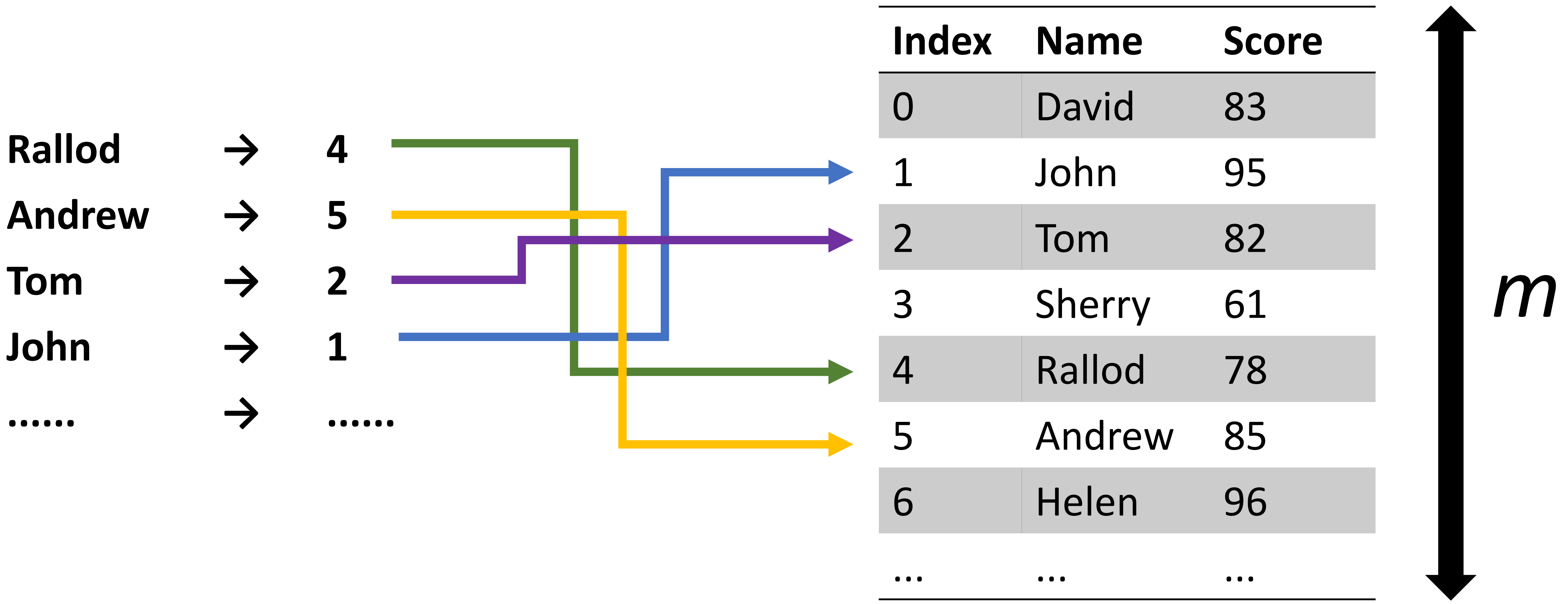
Index	Name	Score
0	David	83
1	John	95
2	Tom	82
3	Sherry	61
4	Rallod	78
5	Andrew	85
6	Helen	96
…	…	…

雜湊函式(Hash Function)

雜湊函式(Hash Function)

給定任意 input，output 必須介於 0~m-1

$$hash: U \rightarrow \{0, 1, \ldots, m-1\}$$



Rallod	→	4
Andrew	→	5
Tom	→	2
John	→	1
……	→	……

Index	Name	Score
0	David	83
1	John	95
2	Tom	82
3	Sherry	61
4	Rallod	78
5	Andrew	85
6	Helen	96
…	…	…

雜湊函式(Hash Function)

- hash function
  - ➢ 多對一函式
- 需求：
  - ➢ 計算簡單→O(1)
  - ➢ 平均分布

- 常見的雜湊函式
  1. Division
  2. Multiplication
  3. Mid-square
  4. Folding addition
  5. Digit analysis

雜湊函式(Hash Function)

Division

- $hash(key) = key \% m$
  - ➢ 透過取餘數壓回 0~m-1
  - ➢ Pros
    - ✓ Fast
  - ➢ Cons
    - ✓ m = ?

Index	Name	Score
0	David	83
1	John	95
2	Tom	82
3	Sherry	61
4	Rallod	78
5	Andrew	85
6	Helen	96
…	…	…

m

雜湊函式(Hash Function)

For example:

Character	D	a	v	i	d
Ascii code	68	97	118	105	100
Total	488				

$$if\ m = 8$$

$$hash(488\,) = 488\,\%\,8 = 0$$

$$hash(David\,) = 0$$

David → 0

雜湊函式(Hash Function)

For example:

Character	d	D	a	v	i
Ascii code	100	68	97	118	105
Total	488				

字母的順序不影響生成的 hash
直接加總是 Bad idea :(

雜湊函式(Hash Function)

For example:

Character	D	a	v	i	d
Ascii code	68	97	118	105	100
Total	293692926308				

$$if\ m = 8$$

$$68 \times 256^4 + 97 \times 256^3 + 118 \times 256^2 + 105 \times 256^1 + 100 \times 256^0 = 293692926308$$

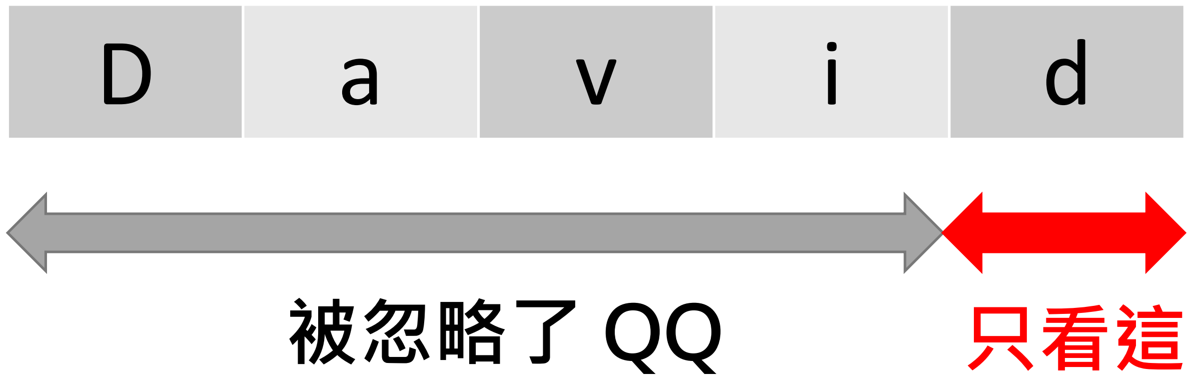
$$hash(293692926308) = 293692926308\%\,8 = 4$$

David $\rightarrow$ ***4***

雜湊函式(Hash Function)

$$hash(key) = key \,\%\, m$$

- 通常表格長度 $m = 2^n$
  - ➢ $hash(key) = key \,\%\, 2^n$
- 假設 n=3
  - ➢ $hash(key) = key \,\%\, 2^3$
  - ➢ $hash(key) = key \,\%\, 8$



m 盡量不要等於 $2^n, n \in N$

雜湊函式(Hash Function)

Multiplication：Key is unknown?

Steps (Given m = 2^n)
- Choose contant C
  - 0 < C < 1
- Multiply C by Key
  - Key × C
- Get the fraction part of Key × C
  - frac = Key × C - ⌊Key × C⌋
- Multiply frac by m
  - m × frac
- Hash = ⌊m × frac⌋

Steps (Given m = $2^3 = 8$, key = 488)
- Choose contant C = 7/16
  - 0 < C < 1
- Multiply C by Key
  - 488 × 7/16
- Get the fraction part of 488 ×7/16
  - frac = $8/16$
- Multiply frac by m
  - 8 × (8/16)
- Hash = ⌊8 × (8/16)⌋ = 5

David → 5

雜湊函式(Hash Function)

Multiplication：

$$\boldsymbol{hash(key) = \lfloor m((key \times C)\,\%1)\rfloor}$$

$\boldsymbol{(key \times C)\%1}$ ： $\boldsymbol{fraction\ part\ of\ key \times C}$

Use every bit in Key！

- Knuth
  - ➢ contant $\mathbf{C} = \frac{\sqrt{5}-1}{2} \sim 0.618$
- but Bit shifting is more efficiency

雜湊函式(Hash Function)

- Mid-square
  1. 把 Key 平方，取中間幾位數當 index
  2. Key = 488, $\mathbf{488^2 = 238144}$
  3. 00000000 00000011 10100010 01000000
  4. 取 15 ~ 17 個 bit → 6

David → ***6***

雜湊函式(Hash Function)

- Folding addition
  1. 切割後再相加
  2. Key = 488
  3. 4 + 8 + 8 = 20
  4. 20 % 8 = 4

也可以只取某幾位數

David $\rightarrow$ ***6***

Example Code

Mission

利用除法 (Division) 設計雜湊函式

- 輸入：字串
- 輸出：整數

Practice

Mission

利用乘法 (Multiplication) 設計雜湊函式

- 輸入：字串
- 輸出：整數

實作雜湊表

實作雜湊表

- Key→Value
  - ➢ Key 跟 Value 的資料型態可以不同！
- 常見的操作
  1. 查詢
  2. 新增
  3. 刪除

實作雜湊表

1. 宣告結構存放 Key 與 Value
2. 以 Array 存放結構
3. 使用雜湊函式對應到索引值
4. 把資料存到相對應的索引值中

陣列
結構
結構
結構
結構
結構
結構
結構
結構

實作雜湊表

結構

1. Key
   - ➢ 客戶姓名
2. Value
   - ➢ 客戶存款

類別

1. 建構式
2. 查詢
3. 新增
4. 刪除

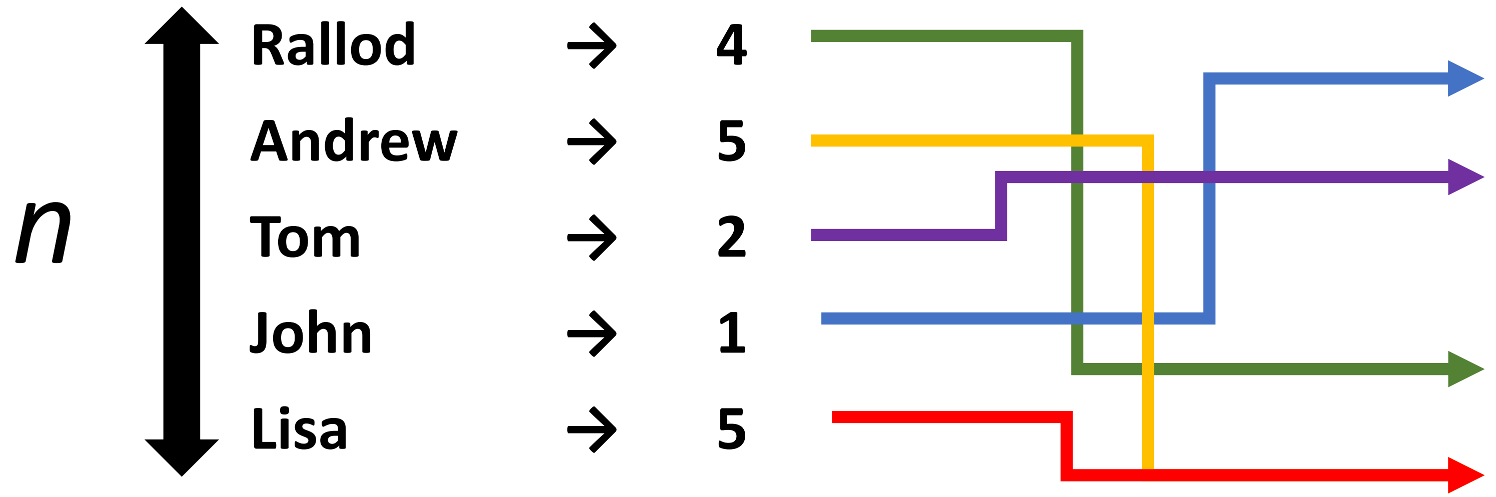
Example Code

Mission

建立銀行系統的雜湊表，使其可以把字串對應到存簿餘額，並撰寫雜湊表內的**搜尋**函式

碰撞(Collision)處理

碰撞(Collision)處理



Index	Name	Score
0	David	83
1	John	95
2	Tom	82
3	Sherry	61
4	Rallod	78
5	Andrew	85
6	Helen	96
…	…	…

m

擠在同一個地方怎麼辦？

Load factor(α) ： $\frac{n}{m}$

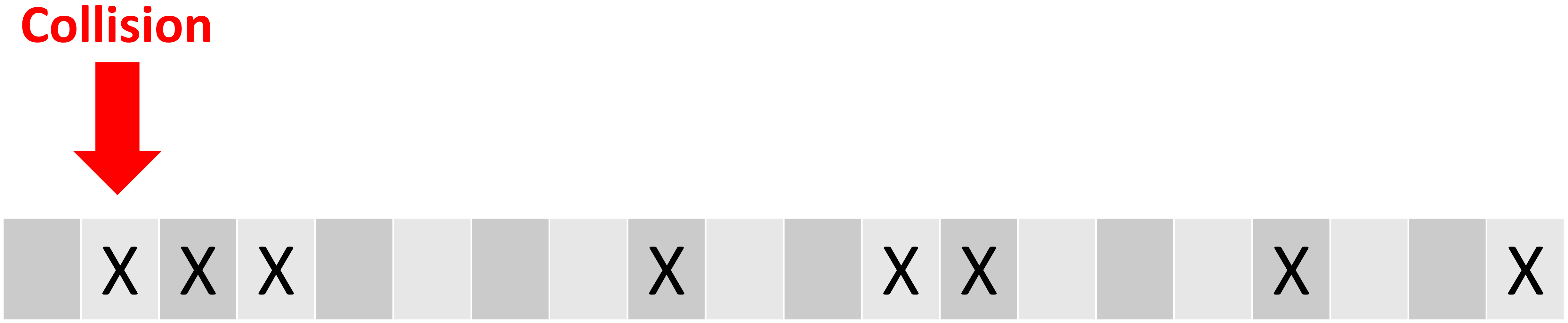
碰撞(Collision)處理

- hash function
  - ➢ 多對一函式
  - ➢ 當 n ≫ m, 極容易發生碰撞
- Open Addressing
  1. Linear Probing
  2. Quadratic Probing
  3. Double Hashing
- Perfect hashing
- Chaining

Open Addressing

Open Addressing : Linear Probing

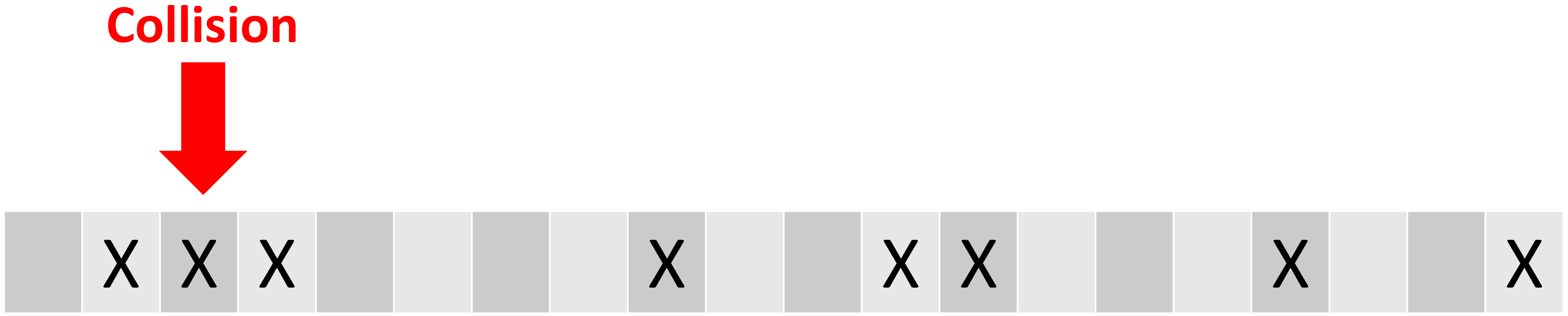
$$\boldsymbol{hash(key, i) = (hash(key, 0) + i)\%\, m}$$
$$i = collision\ times = 0$$



Open Addressing

Open Addressing : Linear Probing

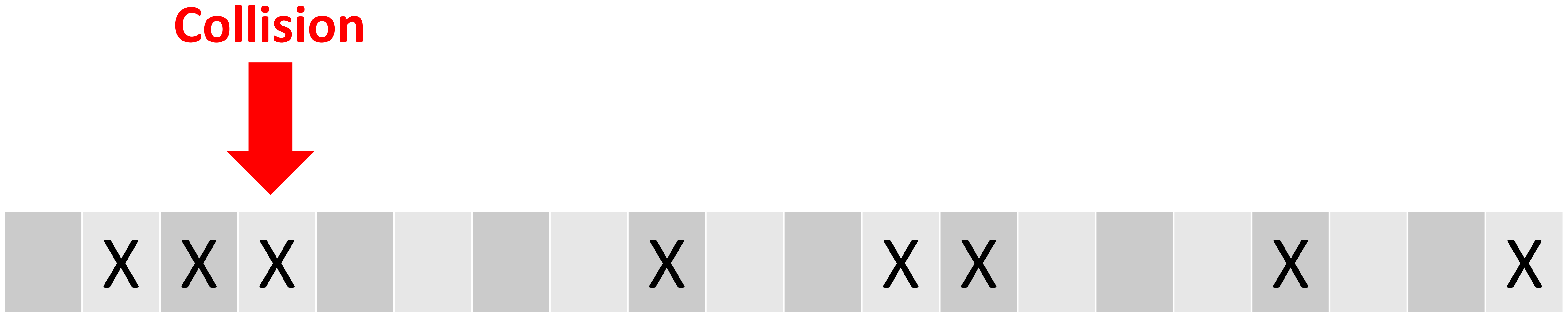
$$\boldsymbol{hash(key, i) = (hash(key, 0) + i)\%\, m}$$
$$i = collision\ times = 1$$



Open Addressing

Open Addressing : Linear Probing

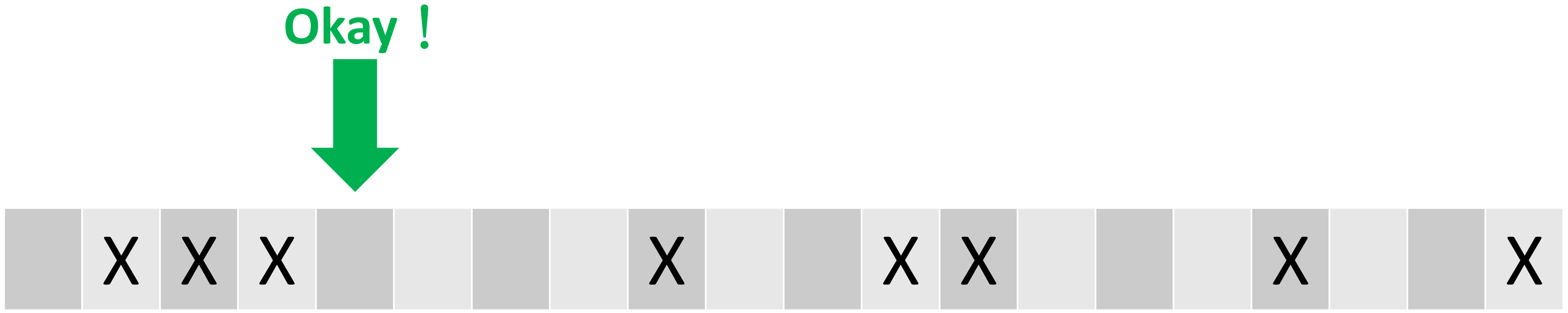
$$\boldsymbol{hash(key, i) = (hash(key, 0) + i)\% \, m}$$
$$i = collision\ times = 2$$



Open Addressing

Open Addressing : Linear Probing

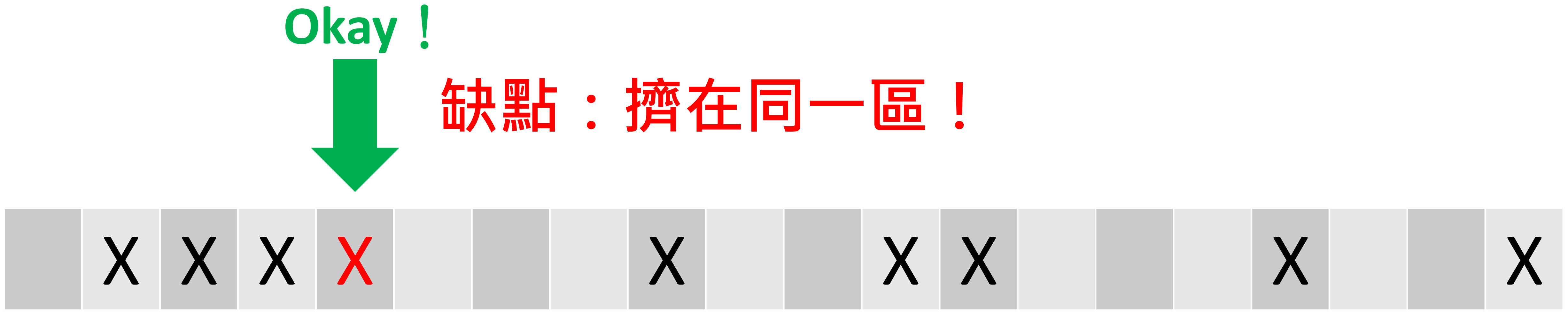
$$\boldsymbol{hash(key, i) = (hash(key, 0) + i)\% \, m}$$
$$i = collision\ times = 3$$



Open Addressing

Open Addressing : Linear Probing

$$hash(key, i) = (hash(key, 0) + i)\% \, m$$
$$i = collision\ times = 3$$



缺點：擠在同一區！

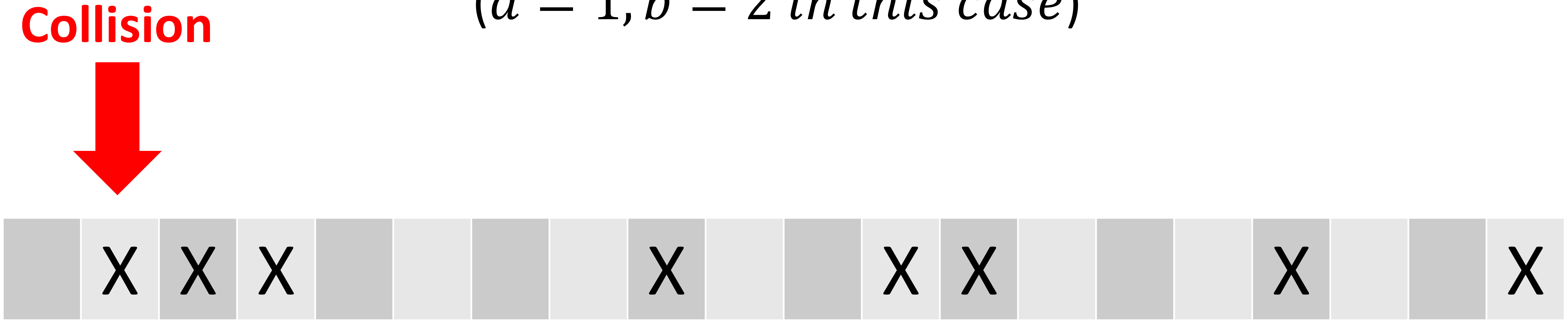
Open Addressing

Open Addressing : Quadratic Probing

$$\boldsymbol{hash(key, i) = (hash(key, 0) + ai^2 + bi)\% \, m}$$

$$i = collision\ times = 0$$

$$(a = 1, b = 2\ in\ this\ case)$$



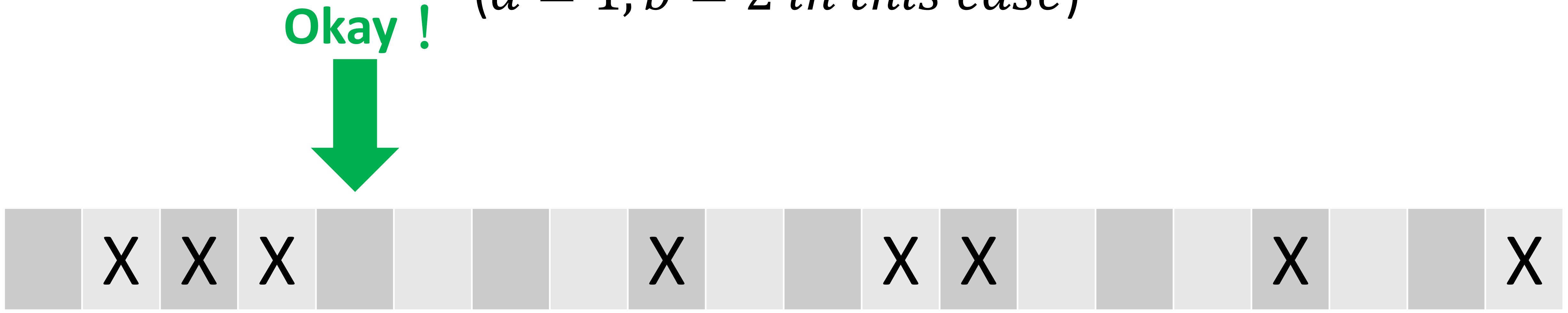
Open Addressing

Open Addressing : Quadratic Probing

$$\boldsymbol{hash(key, i) = (hash(key, 0) + ai^2 + bi)\% \, m}$$

$$i = collision\ times = 1$$

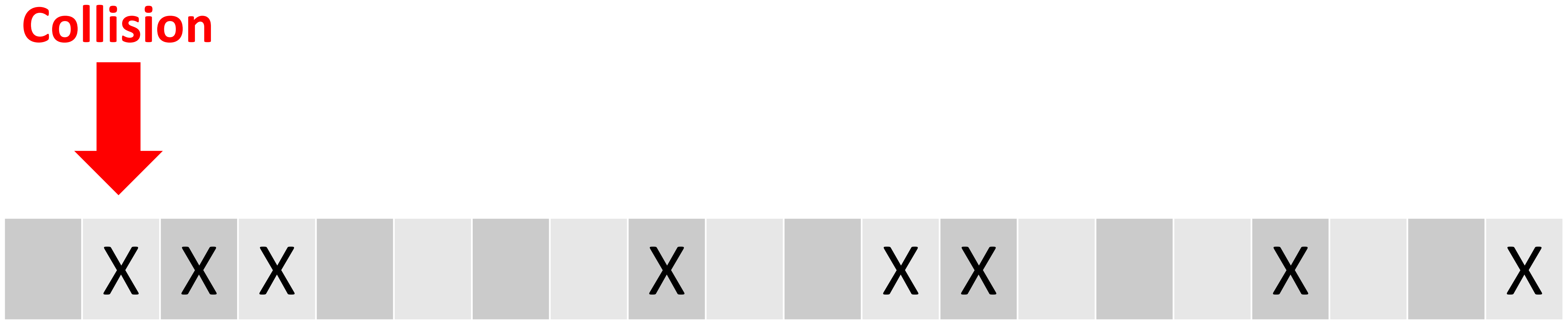
$$(a = 1, b = 2\ in\ this\ case)$$



Open Addressing

Open Addressing：Double Hashing

$$hash(key, i) = (hash_1(key) + i\,hash_2(key))\%\,m$$
$$i = collision\ times$$



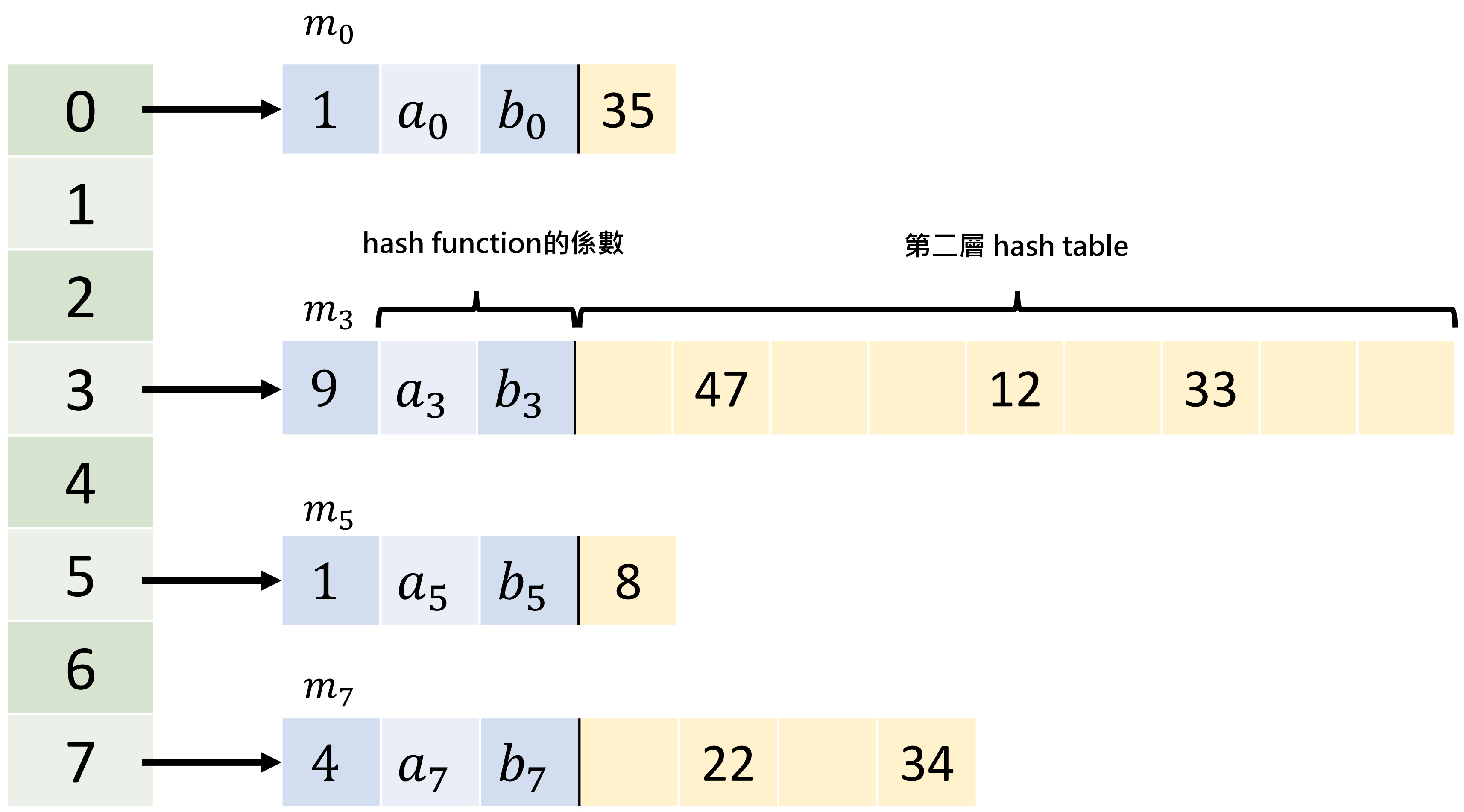
Universal Hashing

- 避免所有鍵值被指派到同一索引值
  - O(n)
- Universal Hashing 解決方式
  - 隨機選擇哈希函式
  - 同樣的鍵值可能對應到不同索引值
  - 讓碰撞的發生機率固定在 $\frac{n}{m}$

Perfect Hashing

- 利用兩次 Universal Hashing
  - ➢ 第一次 hash 到的是另一個 hash table
  - ➢ 第二次 hash 到的才是真正的 value
- Worst case下的運算複雜度仍為 O(1)
- 透過挑選適合的 hash function
  - ➢ 空間複雜度仍為 O(1)

Perfect Hashing



兩次 hash table = O(1+1) = O(1)

Perfect Hashing

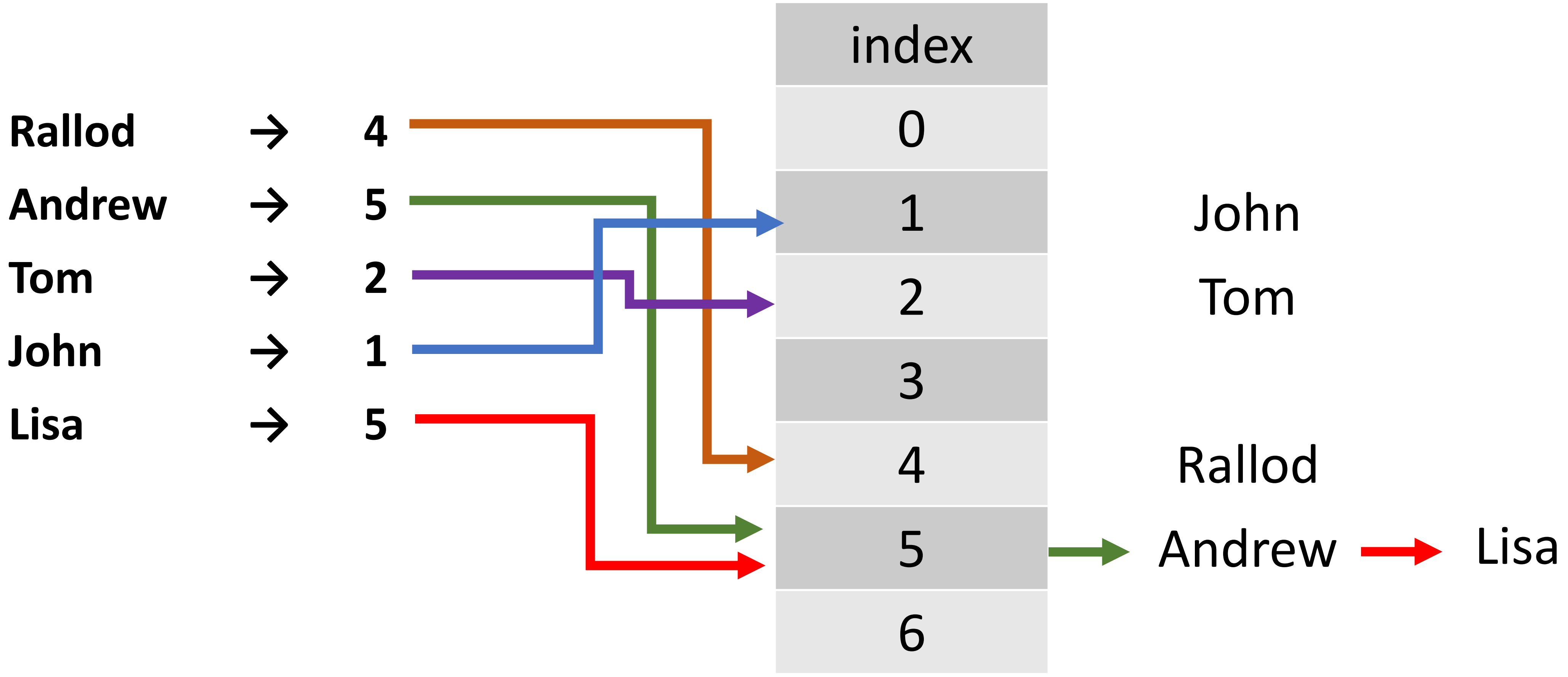
- **核心概念**
  - **讓 $m = n^2$，可以確保碰撞機率$< \frac{1}{2}$**
  - Proof：

    There are C_2^n pairs of keys that may collide;
    each pair collides with probability 1/m.

$$P(collision) = \frac{C_2^n}{n^2} = \frac{n^2 - n}{2} \times \frac{1}{n^2} < \frac{1}{2}$$

Chaining

Chaining：使用 Linked List 連結碰撞



Chaining

- Worst case：O(n)
  - ➢ 所有 Key 都被 Hash function 分到同一個索引值
  - ➢ 等同於 Linked list
- Average case：O(1+α)
  - ➢ Load factor(α) = $\frac{n}{m}$
  - ➢ 資料數量 (n) 與 slot 數量 (m) 的比例
  - ➢ 同時是 每個 slot 內 linked list 的平均長度
  - ➢ If α is small, O(1+α)=O(1)

Collision Avoidance

- 避免碰撞
  - Collision is unavoidable！
  - 使用 multiplication 而非 division
  - 減少碰撞次數來增加效能
  - 但因鍵值未知，碰撞必定無法避免！

雜湊表效能

α：load factor

	Open Addressing Uniform Probing	Open Addressing Linear Probing	Chaining
Successful Search	$\frac{1}{\alpha} \times \ln(\frac{1}{1-\alpha})$	$\frac{1}{2}(1+\left(\frac{1}{1-\alpha}\right)^2)$	$1+\alpha$
Unsuccessful Search	$\frac{1}{1-\alpha}$	$\frac{1}{2}(1+\frac{1}{1-\alpha})$	$1+\alpha$

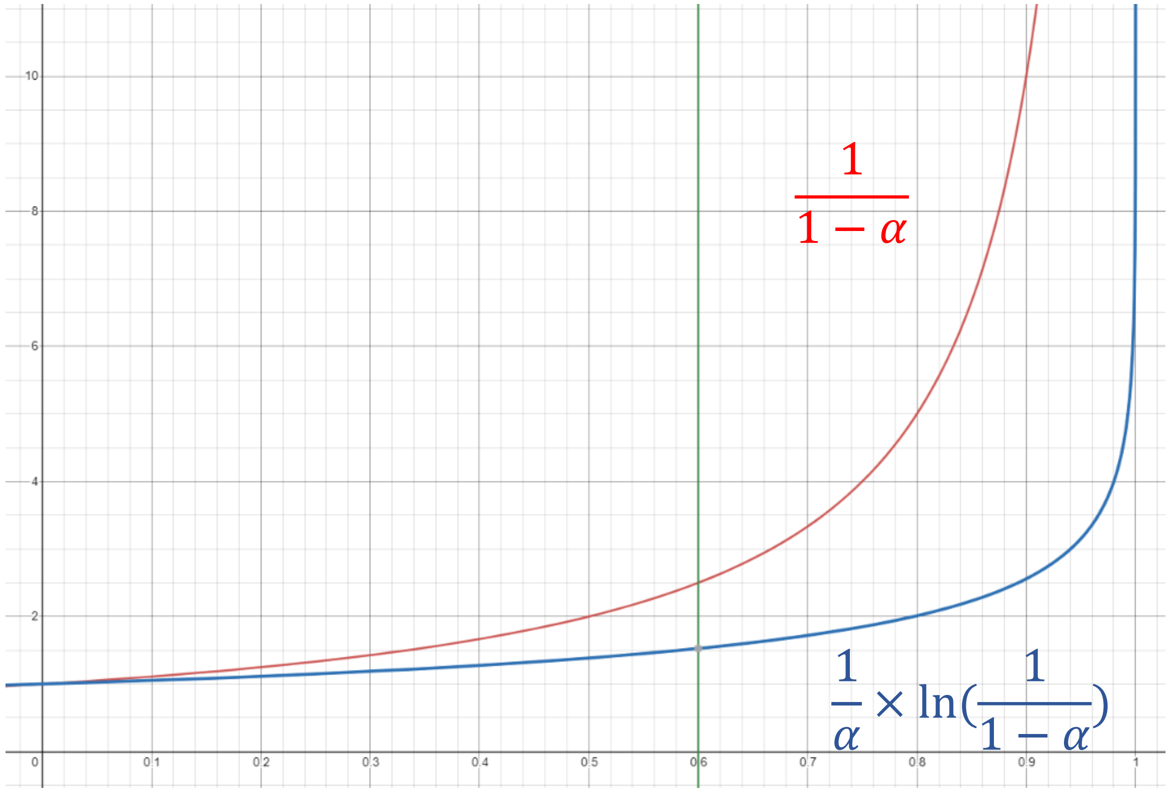
- Successful Search
  - ➢ 刪除時找到有值的位置
- Unsuccessful Search
  - ➢ 插入時找到空的值以插入

- Open Addressing
  - ➢ 不需要頻繁更改記憶體
  - ➢ 但當$\alpha = \frac{n}{m} \sim 1$時，$\frac{1}{1-\alpha} \sim \infty$

雜湊表效能

Open Addressing (Uniform probing)

- for every $k \in U$, $h(k, 0), \dots, h(k, m-1)$
  - Random permutation
  - ✓ Independent of other permutations
  - ✓ Universal Hashing
- The probability of a random cell
  - ✓ Occupied：α
  - ✓ Not occupied ：$1-\alpha$
- Unsuccessful Search：$\frac{1}{1-\alpha}$
  - ✓ $1+\alpha+\alpha^2+\alpha^3+\cdots=\frac{1}{1-\alpha}$
- $\alpha < 0.6$, all small constants

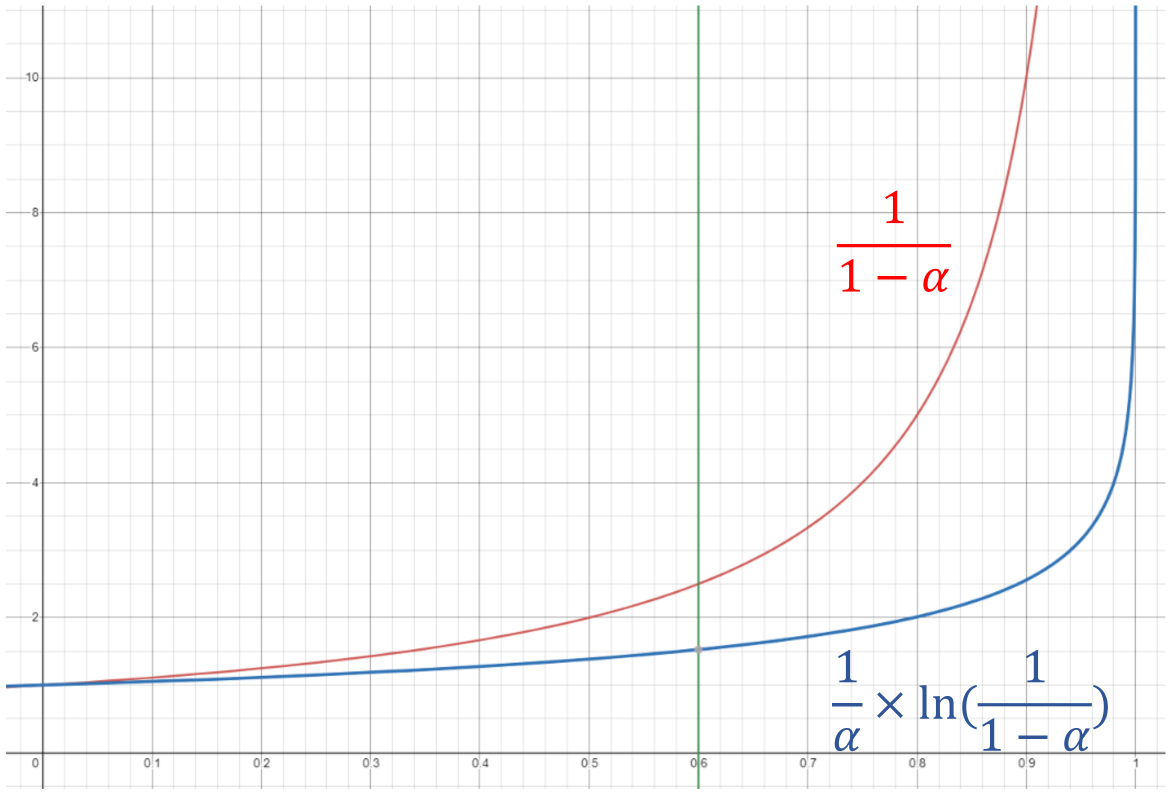


雜湊表效能

Open Addressing (Uniform probing)

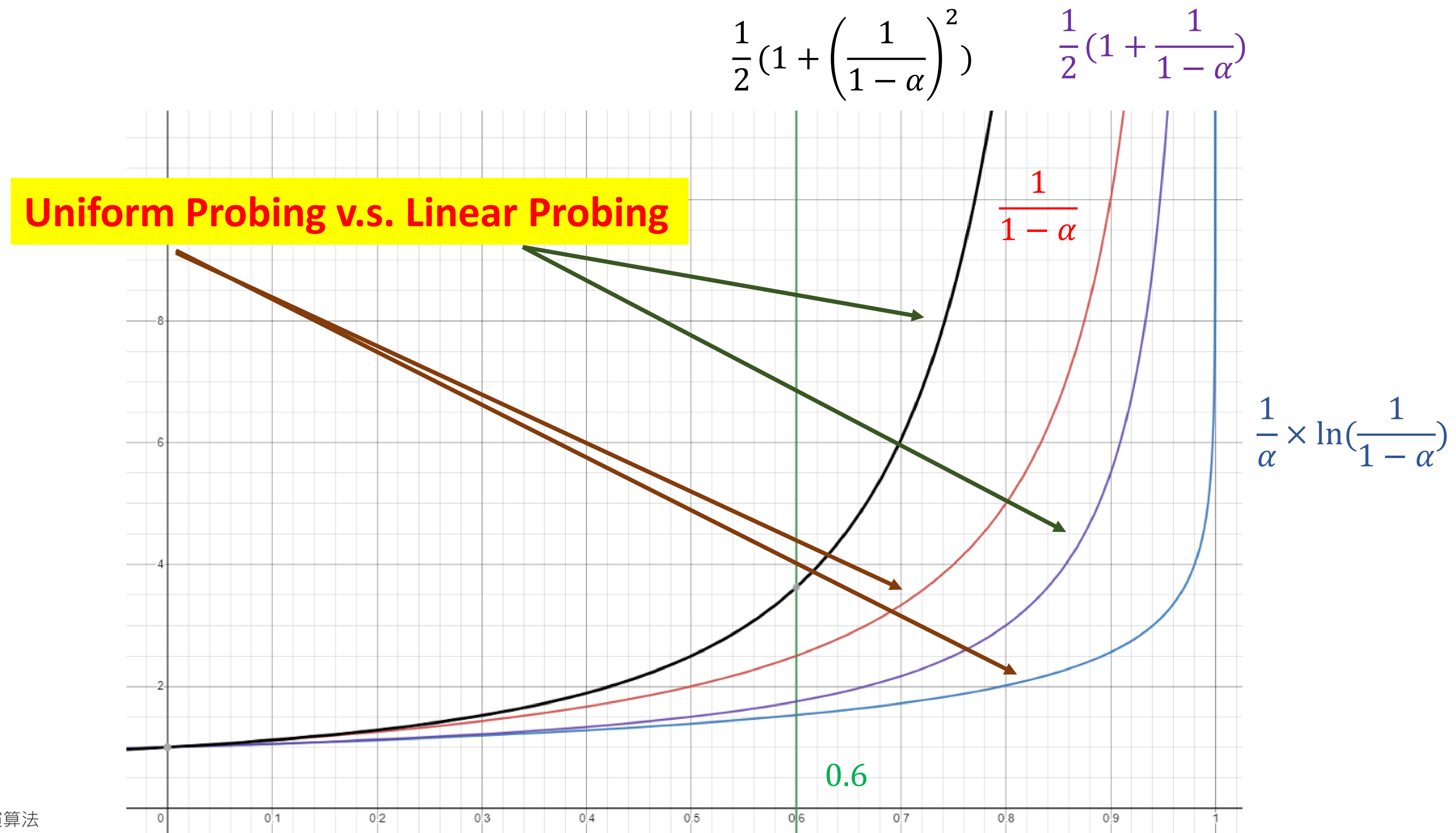
- ➢ Successful Search
  - ✓ (i + 1)st key inserted into the hash table

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{1}{1-\frac{i}{m}}$$
$$=\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i}$$
$$=\frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i}$$
$$=\frac{1}{\alpha}\sum_{k=m-n+1}^{m}\frac{1}{k}$$
$$\leq\frac{1}{\alpha}\int_{m-n}^{m}\frac{1}{x}dx$$
$$=\frac{1}{\alpha}\ln\frac{m}{m-n}$$
$$=\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$$



- ➢ $\alpha < 0.6$, all small constants

雜湊表效能



$\frac{1}{2}(1+\left(\frac{1}{1-\alpha}\right)^2)$
$\frac{1}{2}(1+\frac{1}{1-\alpha})$
$\frac{1}{1-\alpha}$
$\frac{1}{\alpha}\times\ln(\frac{1}{1-\alpha})$
Uniform Probing v.s. Linear Probing
0.6
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
2
4
6
8

Example Code

Mission

以 Chaining 避免碰撞 (Collision)

雜湊表的挑戰

雜湊表的挑戰

- 計算簡單
- 避免碰撞 (Collision Avoidance)
- 解決碰撞 (Collision Resolution)
- Universal Hashing
- Perfect Hashing
- Rehashing：長度 m 不夠用

Rehashing

- 當不斷塞入資料導致原有的 Hash table 過小
  - ➢ 碰撞不停發生
  - ➢ 每個 slot 裏頭被塞入許多資料，導致效率不佳
- 重新配置 Hash table 大小
  - ➢ 把 hash table 的大小擴增為原有兩倍
  - ➢ 再把資料重新 hash 後移入

Rehashing

0	A→B→C→D→E→F→G→
1	H→I→J→K→L→M→N→
2	O→P→Q→R→S→T→U→
3	V→W→X→Y→Z→

Rehashing

0	A→B→C→
1	D→E→F→
2	G→H→I→J→
3	K→L→M→N→
4	O→P→Q→
5	R→S→T→U→
6	V→W→X→
7	Y→Z→

Example Code

Mission

實作Rehashing函式，當 Load factor > 5 時自動呼叫

C++ STL 中的雜湊表

Map 與 Set 的差異

Map

1. Key→Value
2. 資料結構：雜湊表或紅黑樹
3. 常用來儲存對應關係

Set

1. Value
2. 資料結構：雜湊表或紅黑樹
3. 常用來分群、紀錄出現與否

Map 與 Dict

- C++
  - ➢ unordered_map 是 hash table
    - ✓ 插入、搜尋、刪除：$O(1)$
  - ➢ map 不是 hash table
    - ✓ 紅黑樹(Red-Black Tree)
    - ✓ 插入、搜尋、刪除：$O(log_2 N)$
- Python
  - ➢ dict() 是 hash table

Map 與 Dict

unordered_map

- #include <unordered_map>
- 原理：**雜湊表**(Hash table)
- 優：速度快
- 缺：沒有次序資料、空間需求更大
- 速度：較快($O(1)$)
- 適用：沒有次序的資料

map

- #include <map>
- 原理：**紅黑樹**(Red-Black Tree)
- 優：有次序
- 缺：占用多的空間
- 速度：較慢($O(log_2 N)$)
- 適用：有順序要求的資料

Map 與 Dict

- map的操作
  - 函式庫

```
#include <map>
#include <unordered_map>
```

  - 宣告

```
map<datatype_1, datatype_2> map_name;
unordered_map<datatype_1, datatype_2> map_name;
```

  - 迭代器

```
map<datatype_1, datatype_2>::iterator iter;
unordered_map<datatype_1, datatype_2>::iterator iter;
```

Map 與 Dict

- map的操作
  - ➢ 新增

```
map_name.insert(pair<datatype_1, datatype_2>(Key,  Value));
map_name[Key] = Value;
```

Key：first
Value：second

Example Code

Mission

比較 map 與 unordered_map 的新增速度

Map 與 Dict

- map的操作
  - 搜尋

```
iter = map_name.find(Key);

if(iter != map_name.end())
      cout << "Value: " << iter->second << endl;
else
      cout << "Not found in this map! " << endl;
```

Key：first
Value：second

Map 與 Dict

- map的操作
  - ➢ 刪除特定項

```
iter = map_name.find(Key);
map_name.erase(iter);
bool flag = map_name.erase(Key);
```

  - ➢ 全部清空

```
bool flag = map_name.erase(map_name.begin(), map_name.end());
map_name.clear()
```

Map 與 Dict

- map的操作
  - ➢ 判斷是否為空： empty

```
flag = map_name.empty();
```

  - ➢ 取出所有資料

```
for (auto& element : map_name) {
    cout<<"Key: "<<element.first<<" ,Value: "<<element.second<<"\n";
}
```

Map 與 Dict

- map的操作
  - 設定雜湊表的槽數並搬遷資料

    ```
    map_name.rehash(Length);
    ```

  - 擴充容量，確保在該容量前不須搬遷資料

    ```
    map_name.reserve(Length);
    ```

  - 取出雜湊函式的函式指標

    ```
    auto hash_func = map_name.hash_function();
    ```

  - 取出比較 key 的函式指標

    ```
    auto key_eq_func = map_name.key_eq();
    ```

map 的操作

```
#include <iostream>
#include <unordered_map>
using namespace std;

int main()
{
    unordered_map<string, int> balance;
    balance["Mick"] = 100;
    balance.insert(pair<string, int>("Rallod",  101));
    for (auto& element : balance)
        cout << "Key: " << element.first << " ,Value: " << element.second << "\n";

    auto iter = balance.find("Mick");
    if(iter != balance.end()) cout << "Value: " << iter->second << endl;
    else cout << "Not found in this map! " << endl;

    balance.erase("Mick");
    iter = balance.find("Mick");
    if(iter != balance.end()) cout << "Value: " << iter->second << endl;
    else cout << "Not found in this map! " << endl;
    for (iter = balance.begin(); iter!=balance.end();iter++)
        cout << "Key: " << iter->first << " ,Value: " << iter->second << "\n";

    return 0;
}
```

```
Key: Rallod ,Value: 101
Key: Mick ,Value: 100
Value: 100
Not found in this map!
Key: Rallod ,Value: 101
```

Example Code

Mission

Google Code Jam

You receive a credit C at a local store and would like to buy two items. You first walk through the store and create a list L of all available items. From this list you would like to buy two items that add up to the entire value of the credit. The solution you provide will consist of the two integers indicating the positions of the items in your list (smaller number first).

Example Code

Mission

Please download the test data (Small and Big)

- https://goo.gl/Wyjc8f
- https://goo.gl/kyoWgD

The first row of input gives the number of cases, N. N test cases follow. For each test case there will be:

- One row containing the value C, the amount of credit you have at the store.
- One row containing the value I, the number of items in the store.
- One row containing a space separated list of I integers. Each integer P indicates the price of an item in the store.

Each test case will have exactly one solution.

Input
```
3
100
3
5 75 25
200
7
150 24 79 50 88 345 3
8
8
2 1 9 4 4 56 90 3
```

Example Code

Mission

For each test case, output one row containing "Case #**x**: " followed by the indices of the two items whose price adds up to the store credit. The lower index should be output first.

Limits

5 ≤ **C** ≤ 1000

1 ≤ **P** ≤ 1000

Small dataset

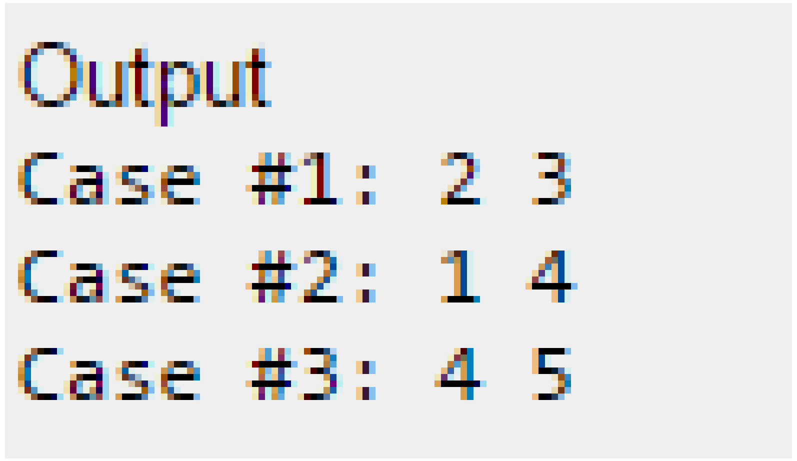
N = 10

3 ≤ **I** ≤ 100

Large dataset

N = 50

3 ≤ **I** ≤ 2000



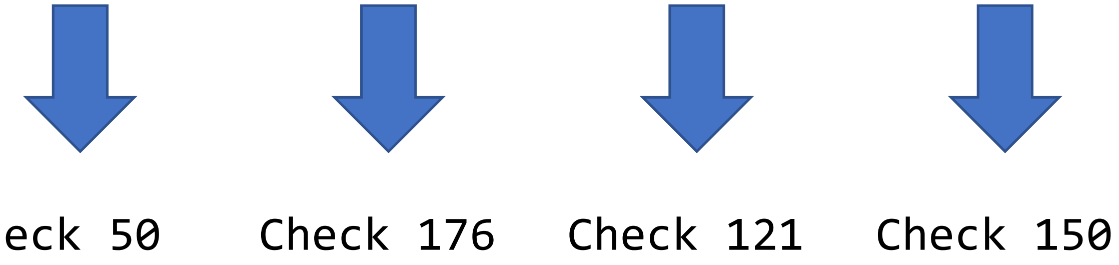
Output

```
Case #1: 2 3
Case #2: 1 4
Case #3: 4 5
```

Example Code

```
Sum: 200
```

150 -> 0	24 -> 1	79 -> 2	50 -> 3			
150	24	79	50	88	345	3



O(n)

Example Code

Mission

LeetCode #217. Contains Duplicate

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Ref: https://leetcode.com/problems/contains-duplicate/

Practice

Mission

LeetCode #219. Contains Duplicate II

Given an integer array nums and an integer k, return true if there are two distinct indices i and j in the array such that nums[i] == nums[j] and abs(i - j) <= k.

Ref: https://leetcode.com/problems/contains-duplicate-ii/

Practice

Mission

LeetCode #242 Valid Anagram

Given two strings s and t, return true if t is an anagram of s, and false otherwise.

Ref: https://leetcode.com/problems/valid-anagram/

Practice

Mission

LeetCode #525. Contiguous Array

Given a binary array nums, return the maximum length of a contiguous subarray with an equal number of 0 and 1.

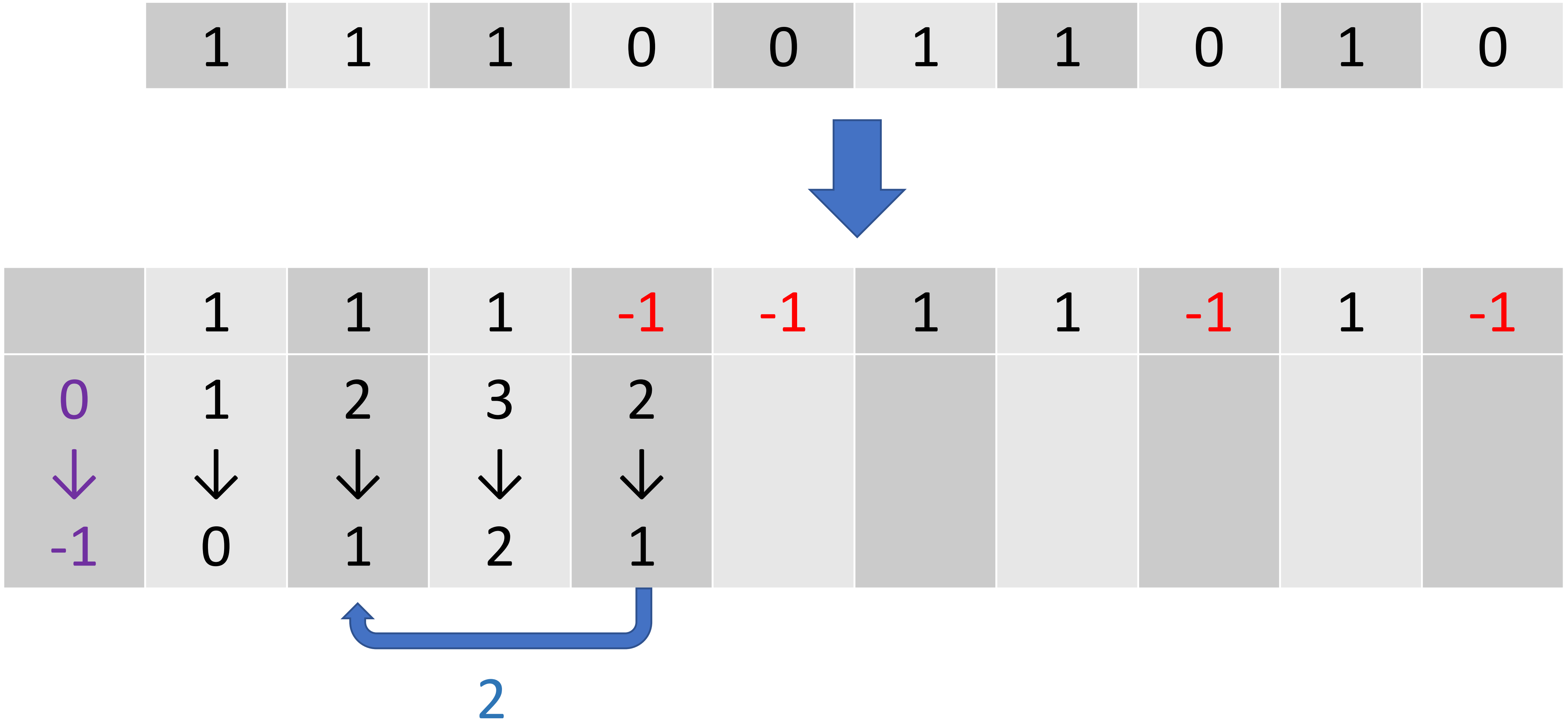
Ref: https://leetcode.com/problems/contiguous-array/

Practice

1	1	1	0	0	1	1	0	1	0

	1	1	1	-1	-1	1	1	-1	1	-1
0 ↓ -1	1 ↓ 0	2 ↓ 1	3 ↓ 2							

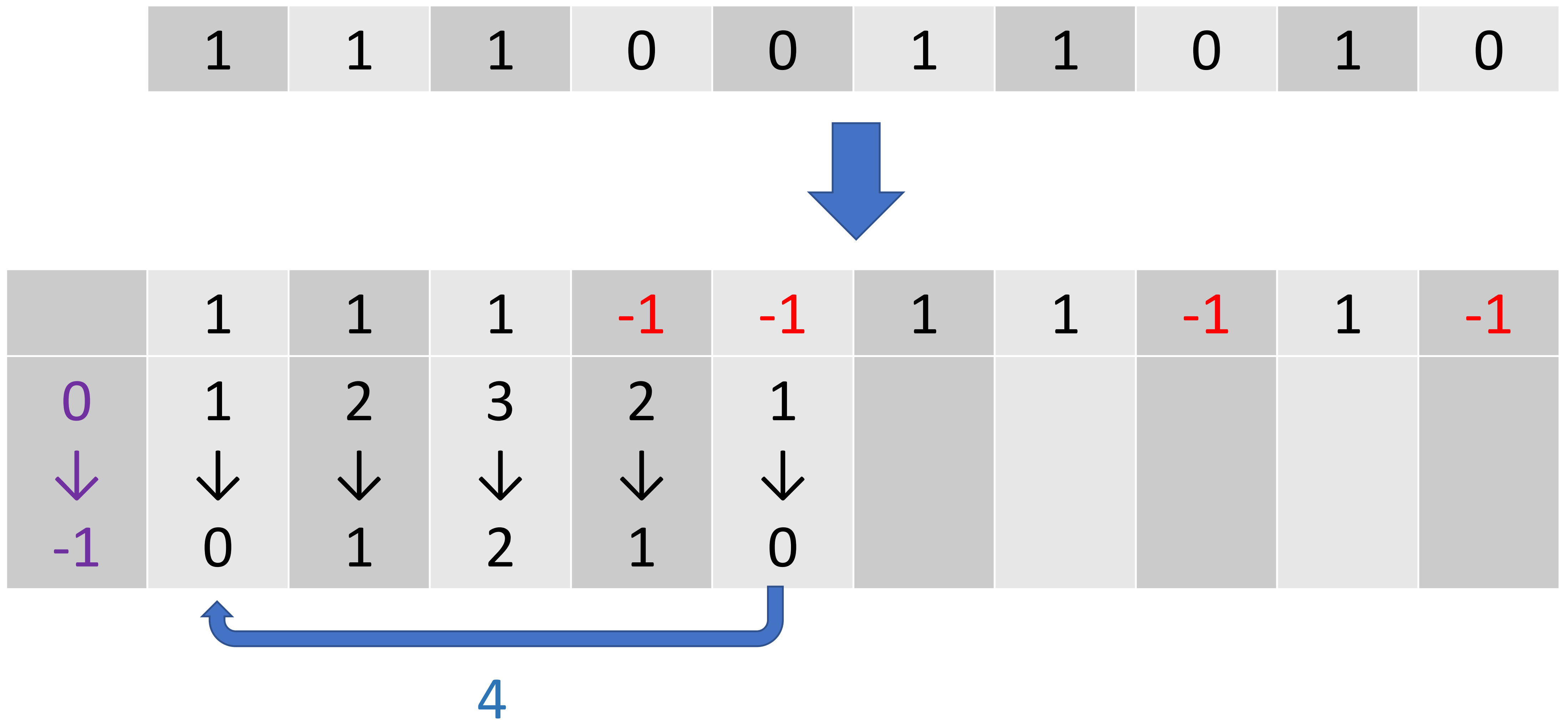
Practice



1	1	1	0	0	1	1	0	1	0

	1	1	1	-1	-1	1	1	-1	1	-1
0 ↓ -1	1 ↓ 0	2 ↓ 1	3 ↓ 2	2 ↓ 1						

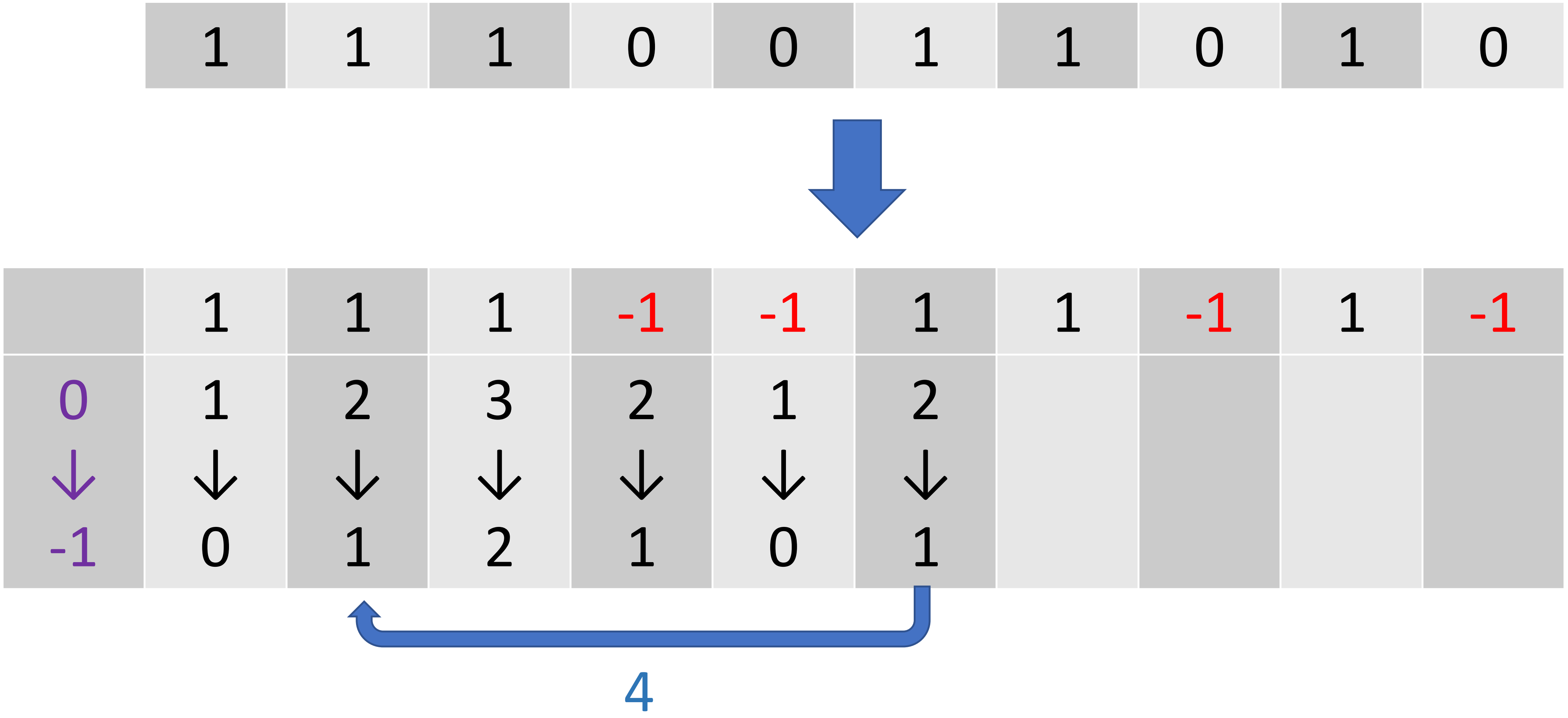
Practice



1	1	1	0	0	1	1	0	1	0

	1	1	1	-1	-1	1	1	-1	1	-1
0 ↓ -1	1 ↓ 0	2 ↓ 1	3 ↓ 2	2 ↓ 1	1 ↓ 0					

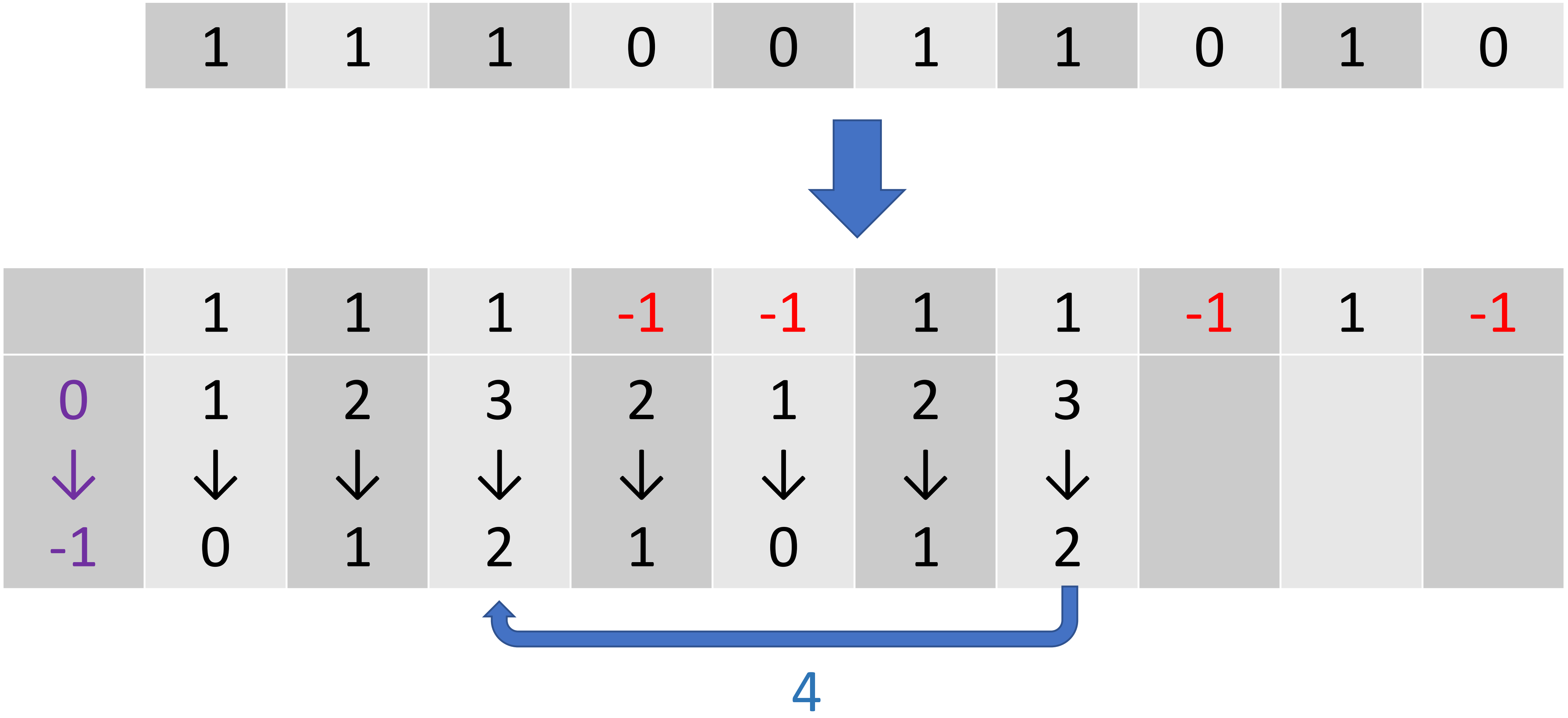
Practice



1	1	1	0	0	1	1	0	1	0

	1	1	1	-1	-1	1	1	-1	1	-1
0 ↓ -1	1 ↓ 0	2 ↓ 1	3 ↓ 2	2 ↓ 1	1 ↓ 0	2 ↓ 1				

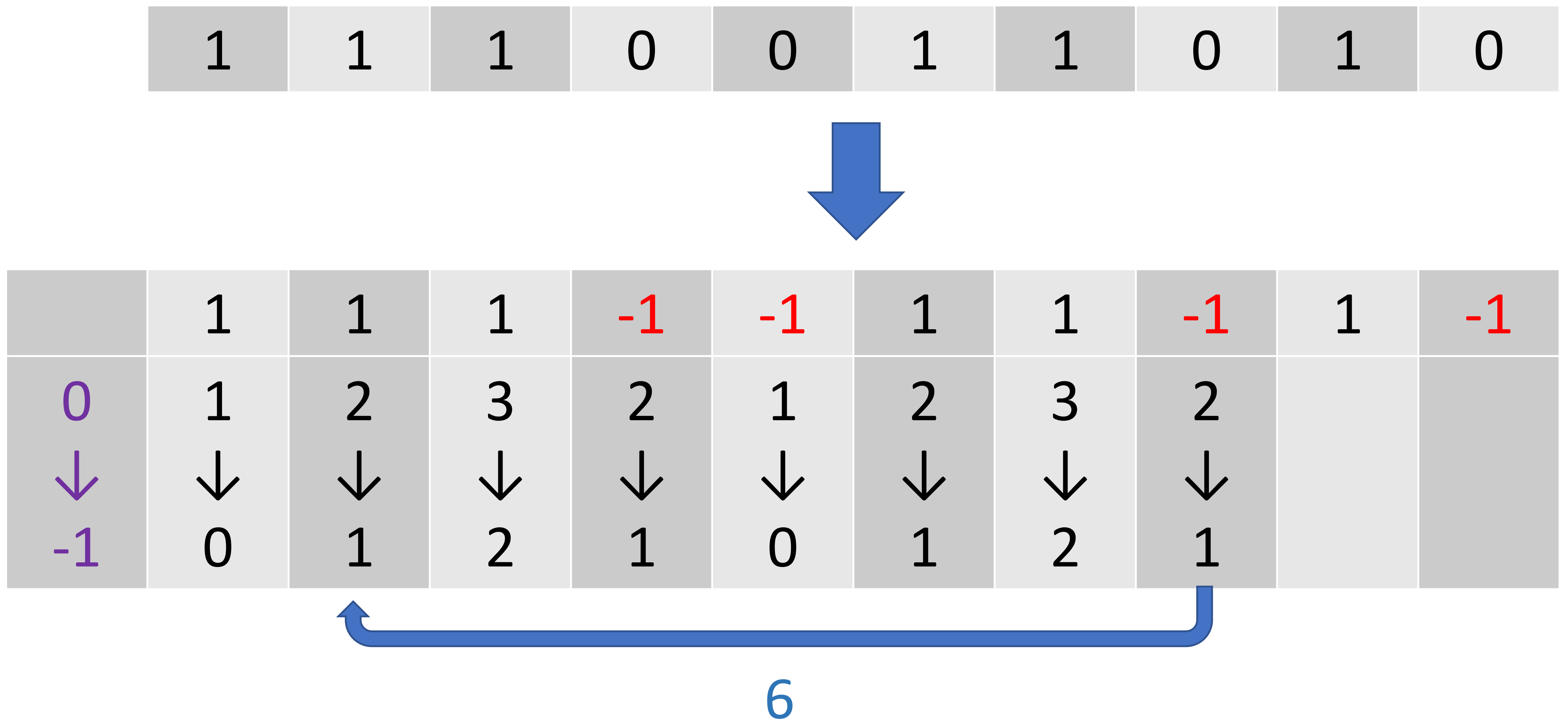
Practice



1	1	1	0	0	1	1	0	1	0

	1	1	1	-1	-1	1	1	-1	1	-1
0 ↓ -1	1 ↓ 0	2 ↓ 1	3 ↓ 2	2 ↓ 1	1 ↓ 0	2 ↓ 1	3 ↓ 2			

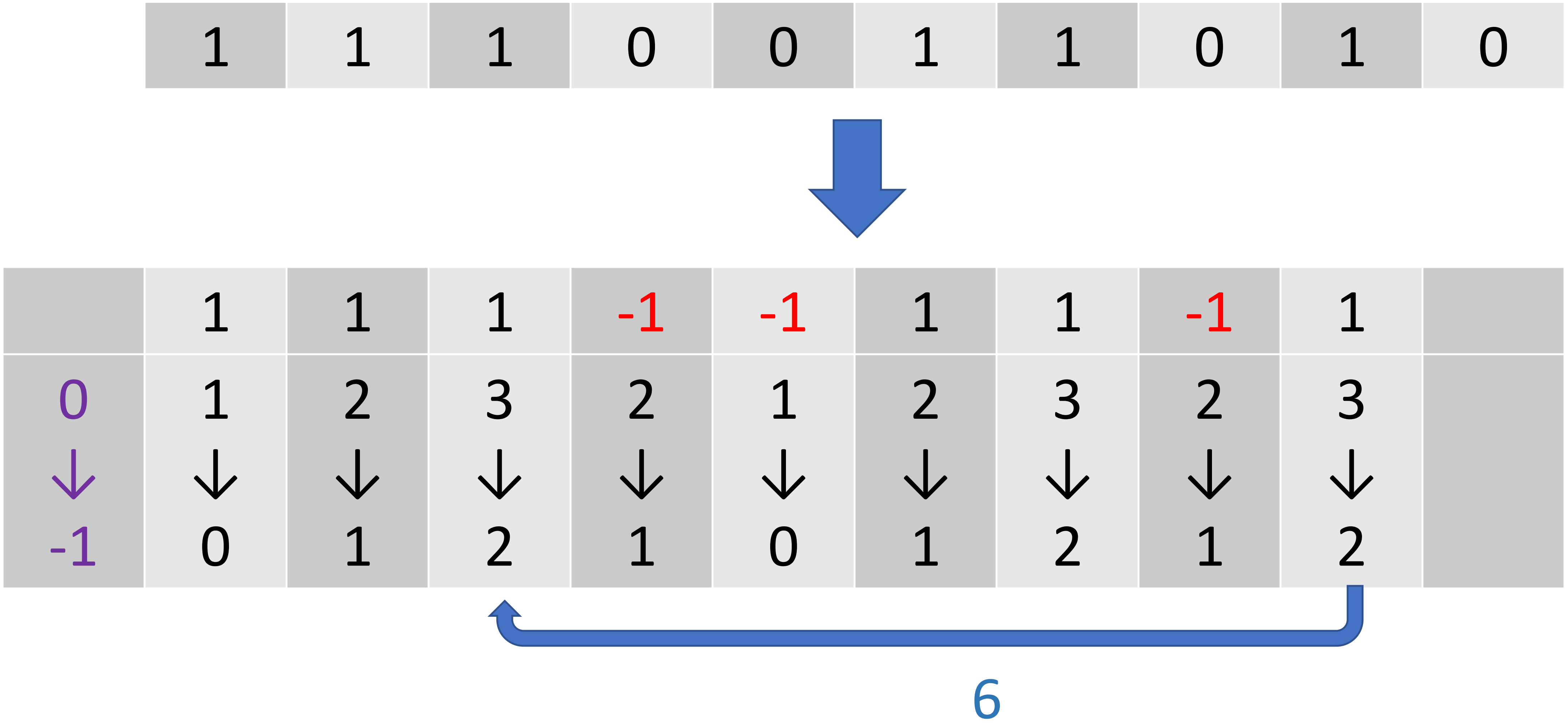
Practice



1	1	1	0	0	1	1	0	1	0

	1	1	1	-1	-1	1	1	-1	1	-1
0 ↓ -1	1 ↓ 0	2 ↓ 1	3 ↓ 2	2 ↓ 1	1 ↓ 0	2 ↓ 1	3 ↓ 2	2 ↓ 1		

Practice



Practice

1	1	1	0	0	1	1	0	1	0

↓

	1	1	1	-1	-1	1	1	-1	1	-1
0 ↓ -1	1 ↓ 0	2 ↓ 1	3 ↓ 2	2 ↓ 1	1 ↓ 0	2 ↓ 1	3 ↓ 2	2 ↓ 1	3 ↓ 2	2 ↓ 1