基礎資料結構與演算法

TYIC 桃高資訊社

資料結構與演算法

資料結構與演算法

(Data Structure & Algorithm，簡稱 DSA)

在程式設計中有著非常重要的地位

使用好的資料結構和演算法

可能會使程式的執行速度變得更快

而使用不妥當的資料結構和演算法

則可能會使程式的執行速度變得緩慢

O (大 O 符號)

$O(x)$ 是用來表示一個函數趨近的上界，其定義為：

若有一個足夠大的正實數 x_0 和兩個 x 的函數 $f(x)$ 和 $g(x)$

且 $(\forall x \geq x_0)\ (\exists M > 0)\ (|f(x)| \leq M|g(x)|)$

則 $\lim_{x \to \infty} f(x) = \lim_{x \to \infty} O(g(x))$

又常將 $\lim_{x \to \infty}$ 省略，故簡寫為 $f(x) = O(g(x))$

舉例：若有一演算法，資料量對時間函數為 $T(n) = 3n^2 + 10n + 2$

可取 $M = 15$，則 $T(n) = O(n^2)$

Ω (大 Ω 符號)

與 $O(x)$ 相似，$\Omega(x)$ 是用來表示一個函數趨近的下界，其定義為：

若有一個足夠大的正實數 x_0 和兩個 x 的函數 $f(x)$ 和 $g(x)$

且 $(\forall x \geq x_0)\ (\exists M > 0)\ (|f(x)| \geq M|g(x)|)$

則 $\lim_{x \to \infty} f(x) = \lim_{x \to \infty} \Omega(g(x))$

又常將 $\lim_{x \to \infty}$ 省略，故簡寫為 $f(x) = \Omega(g(x))$

舉例：若有一演算法，資料量對時間函數為 $T(n) = 3n^2 + 10n + 2$

可取 $M = 3$，則 $T(n) = \Omega(n^2)$

Θ (大 Θ 符號)

$\Theta(x)$ 是在函數的趨近上下界相等時

用來表示函數的趨近界線

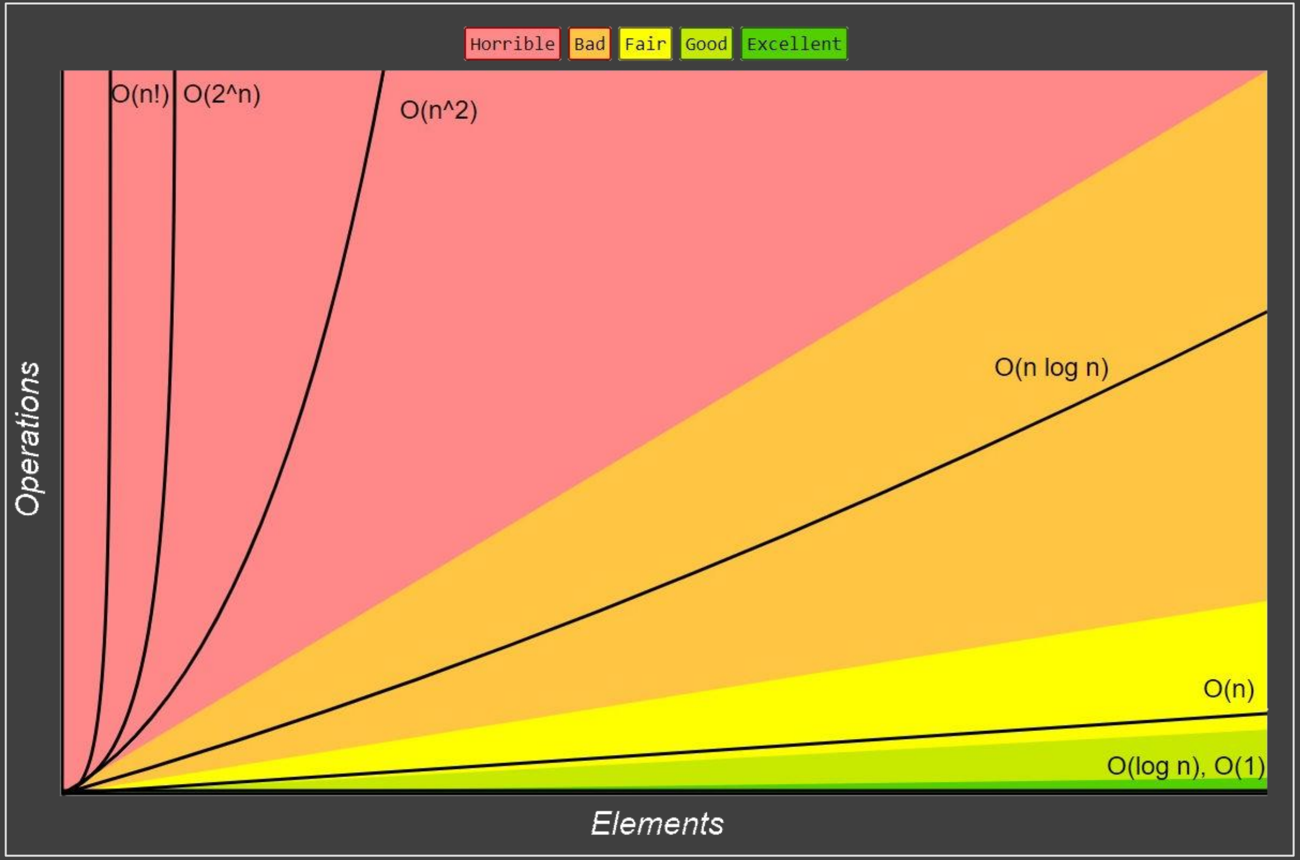
即若 $f(x) = O(g(x)) = \Omega(g(x))$，則 $f(x) = \Theta(g(x))$

舉例：若有一演算法，資料量對時間函數為 $T(n) = 3n^2 + 10n + 2$

已知 $T(n) = O(n^2) = \Omega(n^2)$，則 $T(n) = \Theta(n^2)$

時間複雜度

時間複雜度(time complexity)
是用於描述某一演算法
資料量與執行時間的關係
常用大 O 符號或大 Θ 符號來表示
常見的時間複雜度有：
常數時間 $O(1)$、對數時間 $O(\log n)$
線性時間 $O(n)$、二次時間 $O(n^2)$
線性對數時間 $O(n \log n)$
需注意，因為 n 不可能趨近無限大
故各時間複雜度大小之排序
不一定為真實執行時間多寡之排序



空間複雜度

空間複雜度(space complexity)
是用於描述某一演算法資料量與所需儲存空間的關係
常用大 O 符號或大 Θ 符號來表示
常見的空間複雜度有：$O(1)$、$O(\log n)$、$O(n)$、$O(n^2)$、$O(n \log n)$
同樣的，因為 n 不可能趨近無限大
故各空間複雜度大小之排序不一定為真實使用空間多寡之排序

通常在研究演算法時，時間複雜度的重要性會大於空間複雜度

尋找最大、最小值

對於多個值，想要找尋最大值、最小值

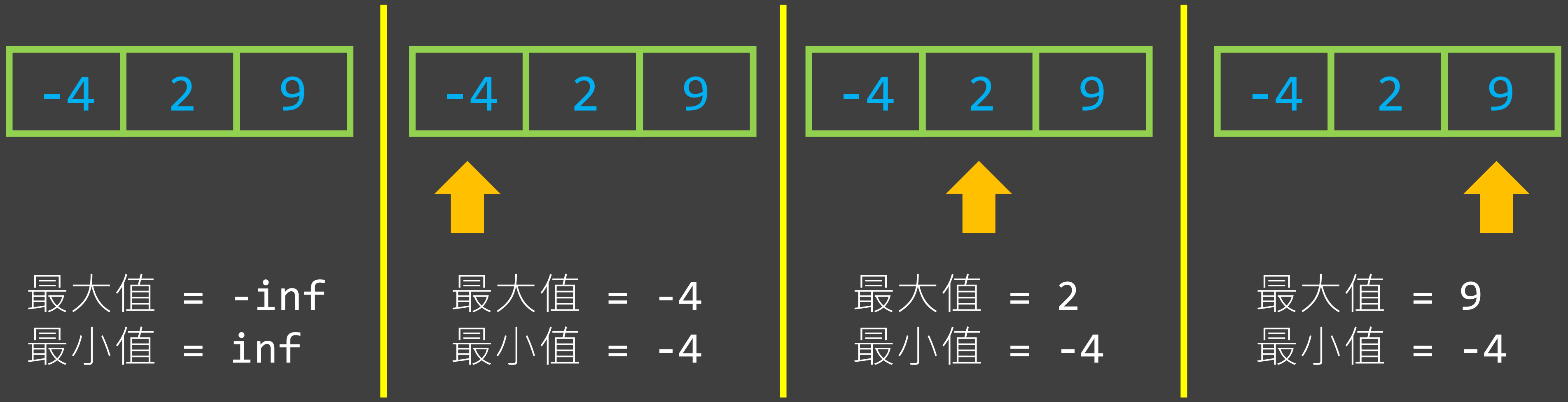
除了對資料排序外，也可利用以下方法，時間複雜度為 $O(n)$：

依序讀取每個值，若較當前的最大值大或最小值小

則將最大值或最小值變為該值

特別注意，最大值須初始化成比所有可能值小的數

最小值須初始化成比所有可能值大的數



尋找最大、最小值

```
import java.util.Scanner;

public class Main1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int max = -2147483648, min = 2147483647;
        for (int i = 0; i < n; i++) {
            int p = scanner.nextInt();
            if (p > max) max = p;
            if (p < min) min = p;
        }
        System.out.printf("max = %d, min = %d", max, min);
    }
}
```

java

```
10
-1 5 -9 8 1000 2 -1999 2 0 1
max = 1000, min = -1999
```

console

獲取一正整數位數

若一正整數 n 滿足 $10^n \leq x = a \times 10^n < 10^{n+1}$ $(1 \leq a < 10)$

則 $\log(10^n) = n \leq \log(x) = n + \log(a) < \log(10^{n+1}) = n + 1$

又 $0 \leq \log(a) < 1$，得 $[\log(x)] = n$　(註：$[m]$ 為下取整函數，如 $[2.7] = 2$)

又已知 10^n 為 $n + 1$ 位數，故 x 為 $n + 1 = [\log(x)] + 1$ 位數

```java
import java.util.Scanner;

public class Main2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        System.out.printf("%d has %d digit(s).", n, (int) Math.log10(n) + 1);
    }
}
```

java

```
900999
900999 has 6 digit(s).
```

console

```
123
123 has 3 digit(s).
```

console

獲取一正整數之每一位數

末位數字即為該正整數除以 **10** 的餘數

該正整數除以 **10** 的商即為去除末位數字後的其他位數字

```java
import java.util.Scanner;

public class Main3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        while (n != 0) {
            System.out.println(n % 10);
            n /= 10;
        }
    }
}
```

java

```
12345
5
4
3
2
1
```

console

```
114514
4
1
5
4
1
1
```

console

最大公因數

最大公因數(greatest common divisor，簡稱 gcd)

程式實現常使用程式碼簡潔的

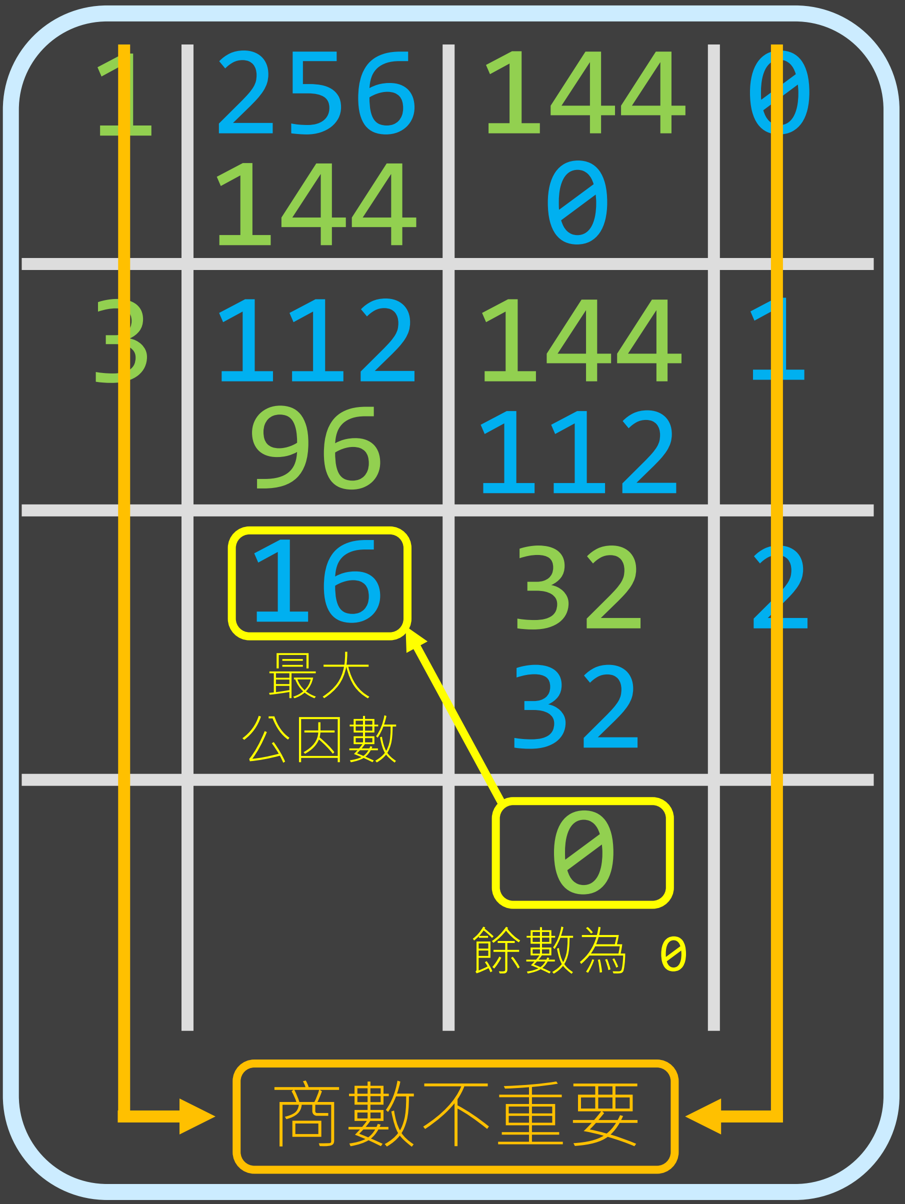
輾轉相除法(歐幾里得算法，Euclidean algorithm)

其說明：若 $a = bq + r$，則 $\gcd(a, b) = \gcd(b, r)$

```java
static int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}

static int gcd(int a, int b, int c) {
    return gcd(gcd(a, b), c);
}
```

java



補充：輾轉相除法證明

1. 已知 $a = bq + r$ $(a, b, r \in \mathbb{N})$，設 $\gcd(a, b) = g$ $(g \in \mathbb{N})$

則 $a = mg, b = ng$ $(m, n \in \mathbb{N})$，且 $\gcd(m, n) = 1$

故 $r = a - bq = mg - nqg = g(m - nq)$ 必有因數 g

2. 設 $\gcd(b, r) = pg$ $(p \in \mathbb{N})$

則 $n = pu, (m - nq) = pv = m - puq$ $(u, v \in \mathbb{N})$

得 $m = pv + puq = p(v + uq)$ 必有因數 p

$\gcd(m, n) = p = 1$，故 $\gcd(b, r) = g$

3. 當 $r = 0$ 時 $a = bq$，則 $\gcd(a, b) = b = g$

1	256 144	144 0	0
3	112 96	144 112	1
	16 最大 公因數	32 32	2
		0 餘數為 0	

最小公倍數

最小公倍數(least common multiple，簡稱 lcm)

程式實現常使用數學性質 $\mathrm{lcm}(a\,,b)=\dfrac{|ab|}{gcd(a\,,b)}$

先求出最大公因數，再求出最小公倍數

```java
static int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}

static int lcm(int a, int b) {
    return a * b / gcd(a, b);
}

static int lcm(int a, int b, int c) {
    return lcm(lcm(a, b), c);
}
```
java

氣泡排序法

氣泡排序法(Bubble Sort)是一種非常簡單的排序法

其原理為：

由左到右，依序比較兩個相鄰的資料，最後一個元素除外

若第一個資料比第二個資料大，便交換這兩個資料

重複 n 次，其中 n 為資料個數

最終就會將資料由小到大排序完成

總共須比較資料 $n-1+n+\cdots+1=\frac{n^2-n}{2}$ 次

故時間複雜度為 $O(n^2)$

氣泡排序法

7	9	2	5	1

7	9	2	5	1

循序搜尋法

循序搜尋法(線性搜尋法，Linear Search)是一種常見的搜尋法

其為依序比對每一個資料直到找到正確的資料，時間複雜度為 $O(n)$

```java
import java.util.Scanner;

public class Main4 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(); // 獲取資料個數
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = scanner.nextInt(); // 讀入資料
        int target = scanner.nextInt(); // 讀入目標資料
        for (int i = 0; i < n; i++) {
            if (arr[i] == target) {
                System.out.println(i + 1);
                return;
            }
        }
        System.out.println("Not found.");
    }
}
```

java

```
10
-2 5 9 10 22 33 44 89 101 777
101
9
```

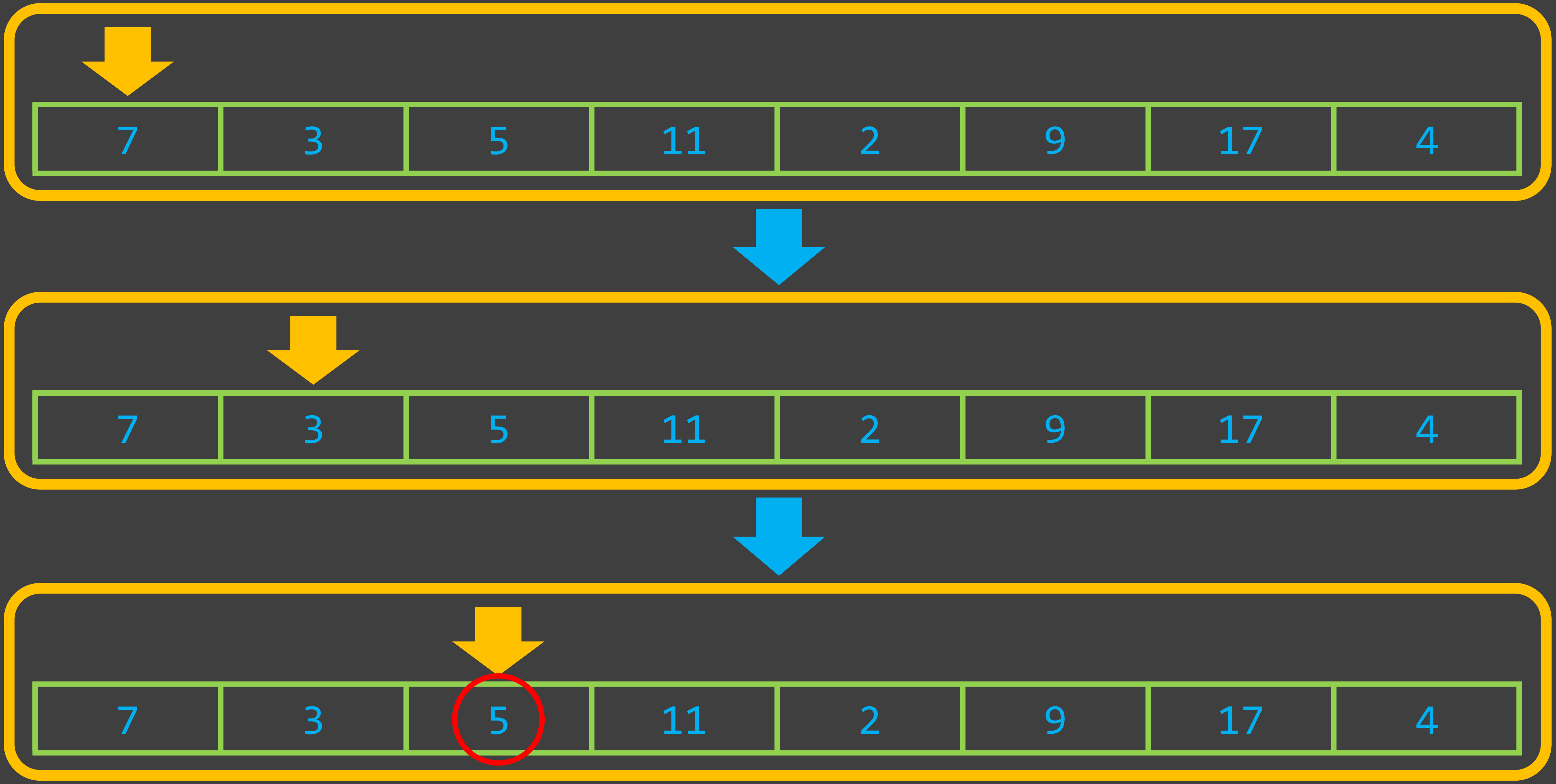
console

```
10
-2 5 9 10 22 33 44 89 101 777
102
Not found.
```

console

循序搜尋法

找 5

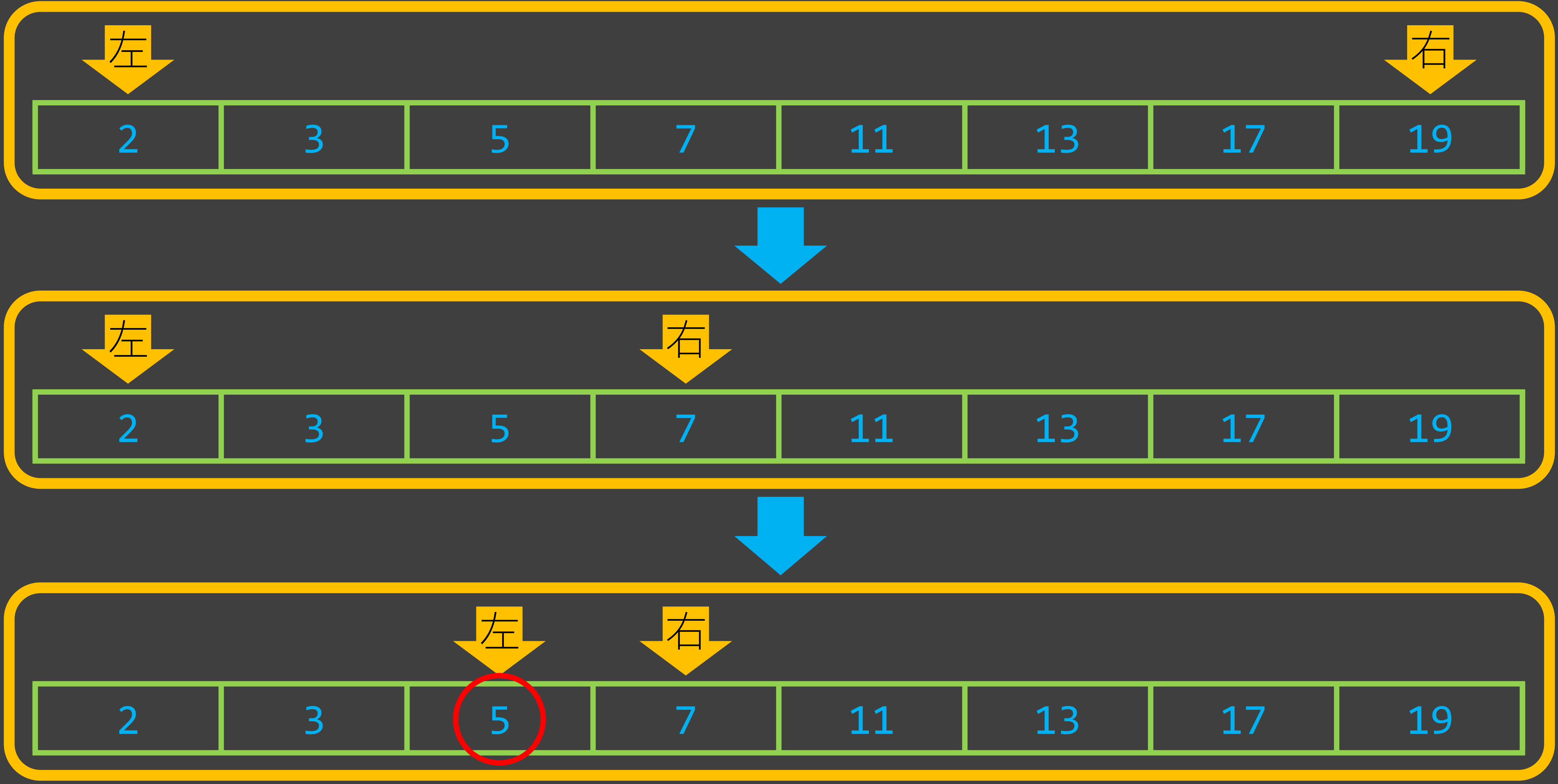


二分搜尋法

二分搜尋法(binary search)是一種常見的搜尋法
在使用二分搜尋法前須將資料由小到大排序
其原理為：每次只會搜尋可能區間中間的資料
若在搜尋到較目標大的資料時，下次搜尋只會搜尋較小的資料
若在搜尋到較目標小的資料時，下次搜尋只會搜尋較大的資料
重複直到搜尋到目標資料，或是可能區間無效，即找不到目標資料
由於二分搜尋法一次就可以排除一半的可能
故時間複雜度為 $O(\log n)$，效率較循序搜尋法較高
但循序搜尋法的資料不須排序，而二分搜尋法需要

二分搜尋法

找 5



二分搜尋法

```java
import java.util.Scanner;

public class Main5 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(); // 獲取資料個數
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = scanner.nextInt(); // 讀入資料
        int target = scanner.nextInt(); // 讀入目標資料

        int l = 0; // 左邊界，目標的最小可能索引值
        int r = n - 1; // 右邊界，目標的最大可能索引值
        while (l <= r) {
            int mid = (l + r) / 2; // 取中間的資料
            if (arr[mid] == target) {
                System.out.println("Target Index: " + mid);
                return;
            }
            if (arr[mid] > target) r = mid - 1;
            else l = mid + 1;
        }
        System.out.println("Target Not found.");
    }
}
```
java

```
10
-2 5 9 10 22 33 44 89 101 777
101
Target Index: 8
```
console

```
10
-2 5 9 10 22 33 44 89 101 777
102
Target Not found.
```
console

二分搜尋法-衍伸應用

二分搜尋法在找不到目標時，可以找大於目標的最小索引值

也就是若要將目標插入資料時，要插入到的索引值

```java
import java.util.Scanner;

public class Main6 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(); // 獲取資料個數
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = scanner.nextInt(); // 讀入資料
        int target = scanner.nextInt(); // 讀入目標資料

        int l = 0; // 左邊界，目標的最小可能索引值
        int r = n - 1; // 右邊界，目標的最大可能索引值
        while (l <= r) {
            int mid = (l + r) / 2; // 取中間的資料
            if (arr[mid] == target) {
                System.out.println("Target Index: " + mid);
                return;
            }
            if (arr[mid] > target) r = mid - 1;
            else l = mid + 1;
        }
        System.out.println("Target Insert Index: " + l);
        // 資料應插入的索引值即為左邊界
    }
}
```

java

```
0
1 2 3 4 5
Target Insert Index: 0
```

console

```
10
-2 5 9 10 22 33 44 89 101 777
102
Target Insert Index: 9
```

console

二分搜尋法-衍生應用

找 4