

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

尋找最大、最小值

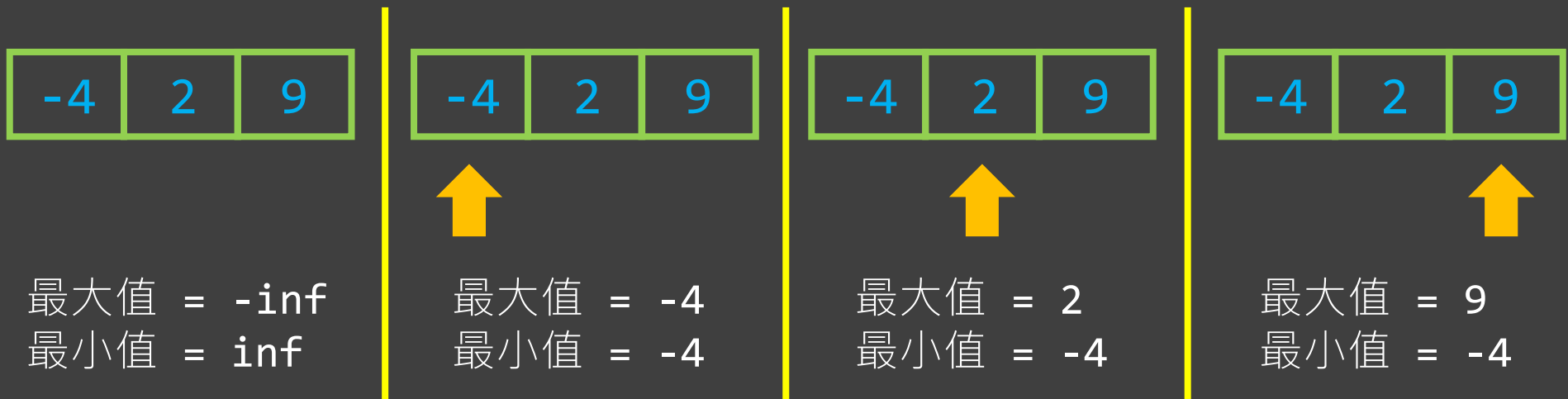
對於多個值，想要找尋最大、最小值

除了對資料排序外，也可利用以下方法：

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);
```

```
    }
```

```
}
```

```
10
```

```
-1 5 -9 8 1000 2 -1999 2 0 1
```

```
max = 1000, min = -1999
```

```
console
```



```
java
```

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

```
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);
    }
}
```

900999

900999 has 6 digit(s). console

123

123 has 3 digit(s). console



java

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

```
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;
        }
    }
}
```



12345		114514	
5		4	
4		1	
3		5	
2		4	
1		1	
1	console	1	console

java

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

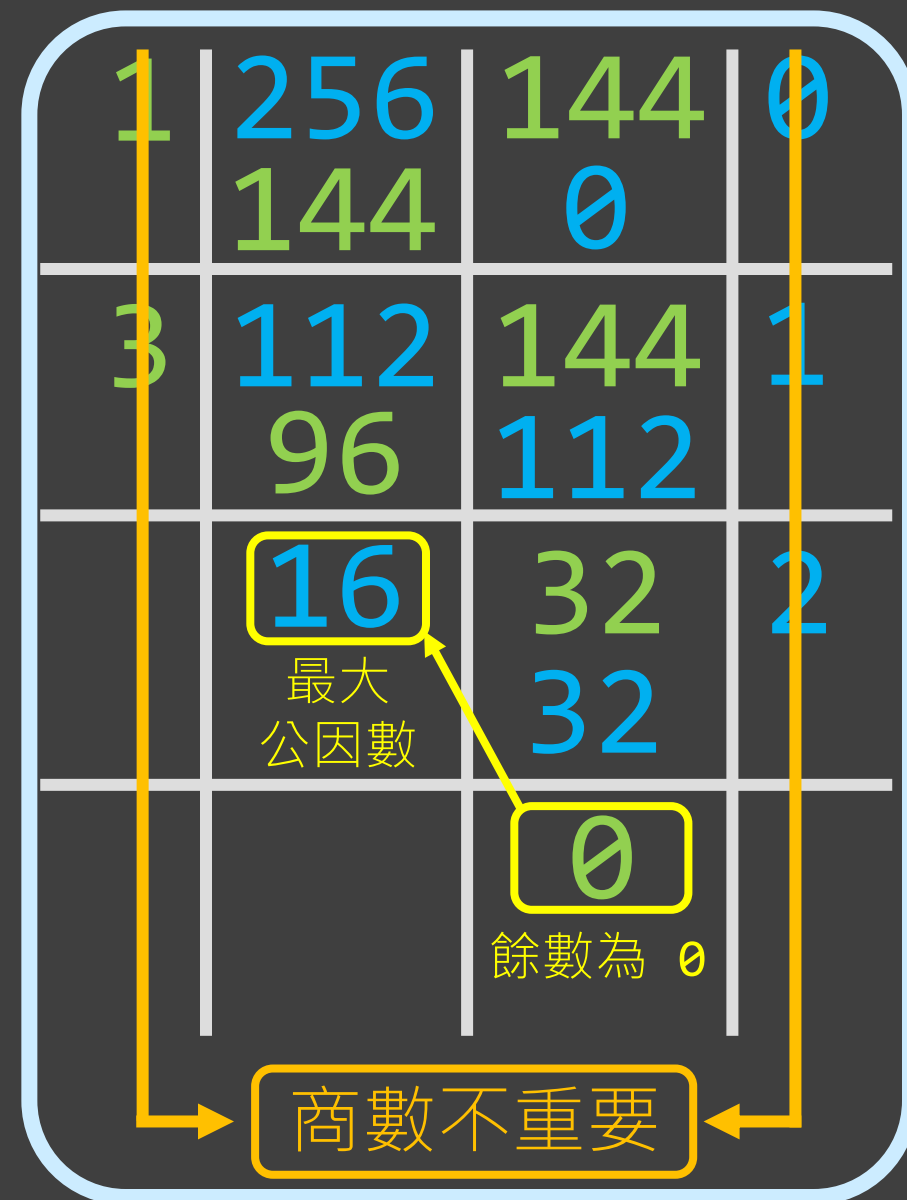
輾轉相除法(歐幾里得算法, Euclidean algorithm)

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

```
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 $\text{lcm}(a, b) \text{gcd}(a, b) = |ab|$

先求出最大公因數，再求出最小公倍數

```
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

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
其原理為依序比對每一個資料直到找到正確的資料

```
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
```

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

因為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

反之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

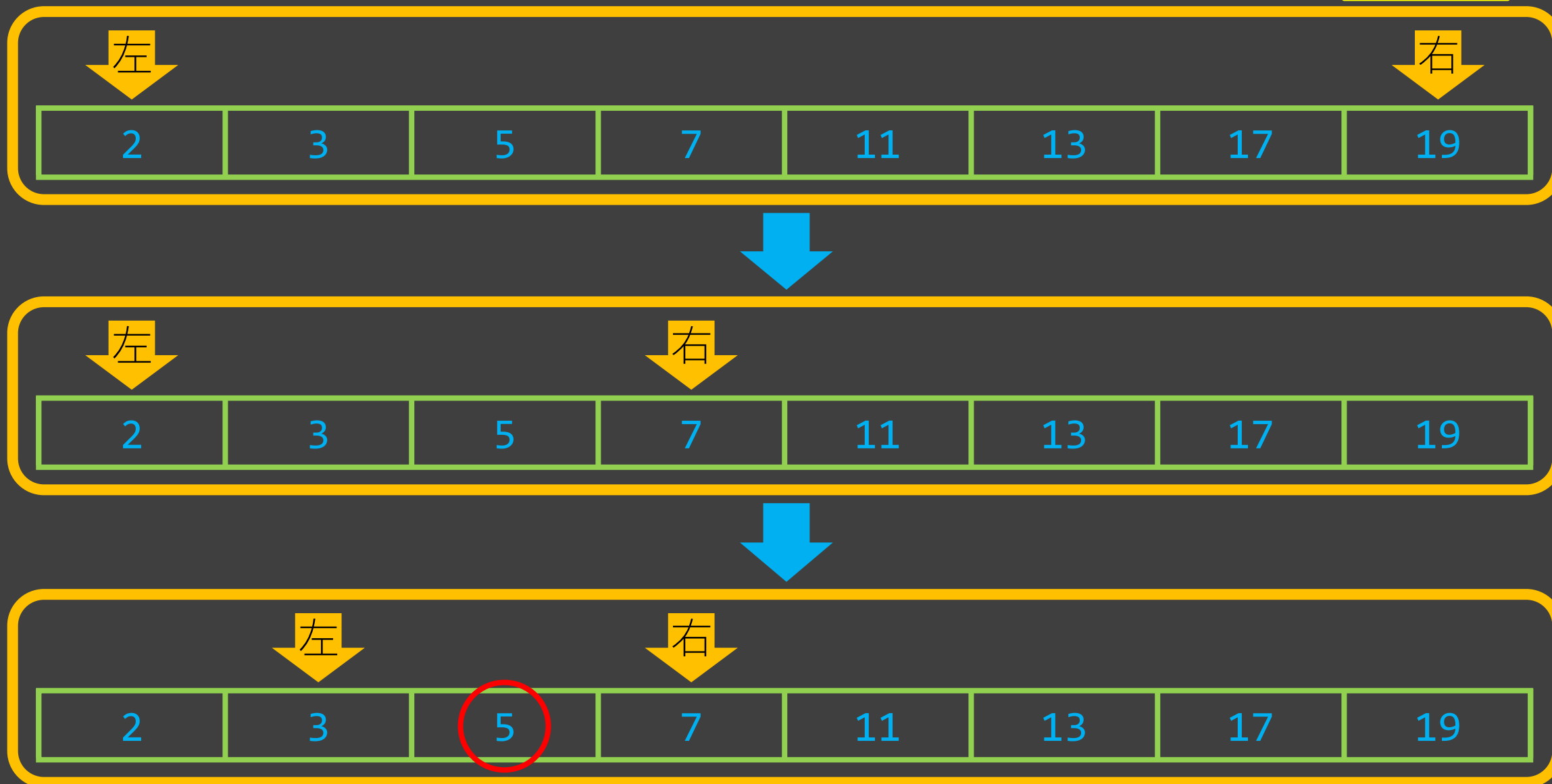
二分搜尋法一次就可以排除一半的可能

相較於循序搜尋法，二分搜尋法效率較高

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

```
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, r = n - 1;
        while (l <= r) {
            int mid = (l + r) / 2;
            if (arr[mid] == target) {
                System.out.println(mid + 1);
                return;
            }
            if (arr[mid] > target) r = mid - 1;
            else l = mid + 1;
        }
        System.out.println("Not found.");
    }
}
```



```
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
```

java