第十二週

王子銜, 陳毅軒, 吳尚龍

電機通訊程式設計

May 6, 2024

Outline

Time complexity

Time complexity

要如何評估程式跑多快?

Time complexity

你要突然想要看鐵達尼號

1. 走去出租店來回共需要 25 min
2. 從伊利論壇下載需要 20 min

伊利論壇下載快

Time complexity

你要突然想要看鐵達尼號 + 阿凡達 + 魔戒⋯等 100 部片

1. 走去出租店來回共需要 25 min
2. 從伊利論壇下載需要 20 min

	出租店	伊利論壇
租 1 部	25	20
租 25 部	25	20×25
租 n 部	25	$20 \times n$

Time complexity

時間複雜度:
一個演算法根據輸入大小估算運算時間的成長幅度

1. Big O notation
2. Omega notation
3. Theta notation

Big O

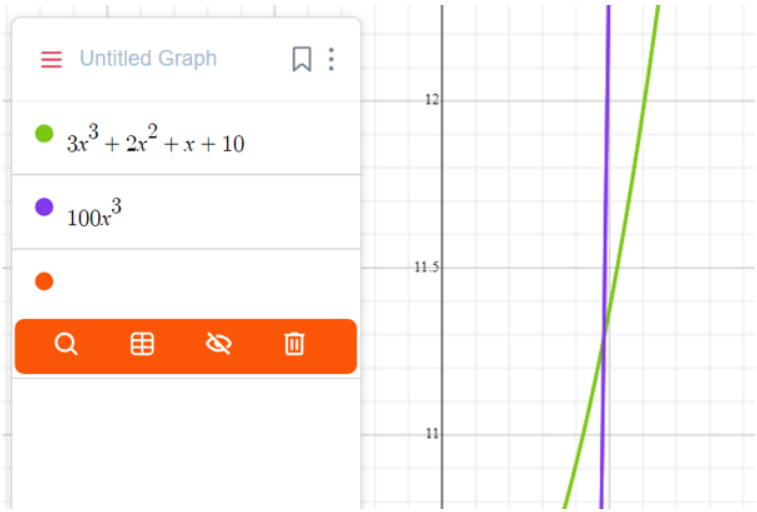
Big O
表示程式的執行時間上限表示最壞不會超過這個層級
Let f, g be a function

$$f(n) = O(g(n)) \iff \exists c, n_0 \text{ s.t. } \forall n \geq n_0, f(n) \leq c \times g(n)$$

Big O

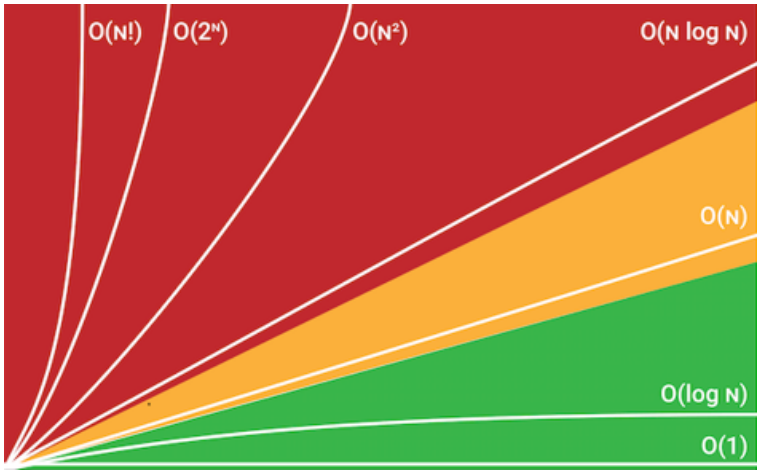
給定函數 $f(n) = 3n^3 + 2n^2 + n + 10$
我們要找到一個函數 $g(n) = n^3$ 和一個常數 $c = 100$
使得對於所有 $n \geq 1$，$3n^3 + 2n^2 + n + 10 \leq 100n^3$

$$f(n) = 3n^3 + 2n^2 + n + 10 \text{ is } O(n^3)$$



Big O

$$\mathcal{O}(1) < \mathcal{O}(\log n) < \mathcal{O}(n) < \mathcal{O}(n \log n) < \mathcal{O}(n^2) < \mathcal{O}(2^n) < \mathcal{O}(n!)$$



Big O

以上週提到的 sort 來舉例:
quicksort 和 bubblesort 的速度差了 $\frac{n^2}{nlogn}$ 倍
不要小看這個數字，當要排序一筆有 20 萬個元素的資料時，這兩個排序算法所花費的時間可以到一萬倍以上

Timelimit

在我們的 Domjudge 中，伺服器每秒可以跑約 10^9 次單位運算
因此計算完複雜度後若超過這個數字，就可能造成 Timelimit

Timelimit

Problem B

奈落之底

Time limit: 1 second

Memory limit: 256 megabytes

舉例而言，HW11 的 B 題，因爲繩子最多有 10^5 條，因此若使用 bubble sort 去排序繩子的長度，會因爲 $10^9 < 10^{10}$ 造成 Timelimit

Linear Search

Linear Search

線性搜尋（Linear Search）又稱順序搜尋法，原理是從資料結構的一端開始，逐一檢查每個元素，直到找到所尋找的特定元素或是遍歷完所有元素爲止。
每次檢查一個元素，若該元素與目標元素相符，則返回該元素的位置，表示搜尋成功；若不符，則繼續檢查下一個元素。如果所有元素都已檢查過，且未找到目標元素，則返回一個指示未找到的結果，表示搜尋失敗。
線性搜尋不需要資料預先排序，適用於資料量不大或無法預先排序的情況。

Linear Search

```
int LinearSearch(int arr[], int n, int target) {
    for (int i = 0; i < n; i++) {
            if (arr[i] == target) {
                return i;
            }
        }
    return -1;
}
```

Linear Search

時間複雜度

檢查次數 n
時間複雜度爲 $O(n)$

Discuss

無論陣列是否有排序，Linear Search 所需要的執行次數是一樣的，如果陣列長度 n 更大時，Linear Search 會沒有效率。
對於一個有排序的陣列，有沒有更快的方法找到 target?

Discuss

因爲陣列是經過排序過的，表示你搜尋到的數字只會越來越大或者越來越小。
如果直接看最中間的數字是多少，就可以知道目標是在左半還是右半。
要搜尋的長度就直接砍半了。

Binary Search

Binary Search

二分搜尋法（Binary Search）又稱折半搜尋法，原理是在一個已經排序的資料結構中，從中間開始比較目標值與中間元素的大小。
如果目標值等於中間元素，則搜尋成功並返回該元素的位置；如果目標值較小，則繼續在左側的子序列中進行搜尋；如果目標值較大，則在右側的子序列中進行搜尋。接著再從新的子序列中選取中間元素，重複之前的比較過程。
每進行一次比較，搜尋範圍就減少一半，直到找到目標值或子序列範圍縮減到零。
二分搜尋法的效率較高，適用於處理大量已排序 的數據。

Binary Search

mid = (left+right)/2
Target= 9
Left index = 0
Right index = 11
Mid index = 5
搜尋 mid，接著修改 left 或 right 的值。

Binary Search

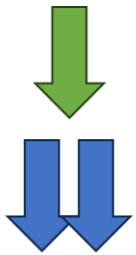
mid = (left+right)/2
Target= 9
Left index = 6
Right index = 11
Mid index = 8
搜尋 mid，接著修改 left 或 right 的值。

Binary Search

mid = (left+right)/2
Target= 9
Left index = 6
Right index = 7
Mid index =6
搜尋 mid，接著修改 left 或 right 的值。

Binary Search

mid = (left+right)/2
Target= 9
Left index = 7
Right index = 7
Mid index = 7



1 2 3 5 6 7 8 9 10 12 15 18

Binary Search

```
int BinarySearch(int arr[], int n, int target){
  //arr必須是排序後的陣列
  int left = 0, right = n - 1;
  int mid;
  while(1){
      mid = (left + right) / 2;
      if (arr[mid] == target)
          return mid;
      if (arr[mid] < target) {
          left = mid + 1;
      } else {
          right = mid - 1;
      }
      if (left > right) {
          break;
      }
  }
  return -1;
}
```

Linear Search

時間複雜度

搜尋區間砍半再砍半，最多也只會執行 $log_2 n$ 次
時間複雜度爲 $O(logn)$