0819823陳子祈 演算法導論 HW1

一、試驗成果

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

根據基本要求1、2，我使用for迴圈設定了不同大小的input array，大小從100逐次成長10倍到100000，根據實測結果如下:

1. Brute-force algorithm to find maximum subarray

|  |  |
| --- | --- |
| Input size | Runtime(ms) |
| 100 | 0 |
| 1000 | 557 |
| 10000 | 469322 |
| 100000 | X |

就Brute-force algorithm而言，理論上10倍的input size，執行時間約成長1000倍，因此我跑過輸入大小為10000之後，利用輸入大小為10000的執行時間推算輸入大小為100000的執行時間為: 469322 \* 1000 = 469322000 (ms) = 469322 (s) = 469322/3600 (hr) ~= 130 (hr)。因為跑程式需要太長時間，我怕會損耗CPU太多，所以我就放棄執行輸入大小為100000的部分。

就因為這樣的結果資料不足，因此我測試了輸入大小從250~8000，每次輸入大小為前一次輸入大小的兩倍，程式碼改為如下:

一張含有 文字 的圖片

自動產生的描述

重製表格如下:

|  |  |  |  |
| --- | --- | --- | --- |
| Input size | Runtime(ms) | This turn Runtime/ Last turn Runtime | This turn Runtime/ ((size/100)^3) |
| 250 | 9 | X | 1.125 |
| 500 | 57 | 57 / 9 = 6.33333 | 0.456 |
| 1000 | 411 | 411 / 57 = 7.21053 | 0.411 |
| 2000 | 3134 | 3134 / 411 = 7.6253 | 0.39175 |
| 4000 | 25941 | 25941 / 3134 = 8.27728 | 0.405328 |
| 8000 | 254962 | 254962 / 25941 = 9.82853 | 0.497973 |

由上表可知，2倍的input size，執行時間約成長8倍，且與n^3成正比，若n為input size，f(n)為此演算法執行的時間，符合Θ(f(n)) = n^3的趨勢。

1. Divide and Conquer algorithm to find max subarray

|  |  |
| --- | --- |
| Input size | Runtime(ms) |
| 100 | 0 |
| 1000 | 0 |
| 10000 | 1 |
| 100000 | 13 |

就Divide and Conquer而言，由於執行時間實在太快，只能觀察到Input size大於10000以上的執行時間，所以我把設定了不同大小的input array的for迴圈改為:

一張含有 文字 的圖片

自動產生的描述

重製表格如下:

|  |  |  |  |
| --- | --- | --- | --- |
| Input size | Runtime(ms) | This turn Runtime/ Last turn Runtime | This turn Runtime/ (size\*lg(size)) \*100000 |
| 10000 | 1 | X | 7.52575 |
| 20000 | 3 | 3 / 1 = 3 | 10.4985 |
| 40000 | 4 | 4 / 3 = 1.33333 | 6.5412 |
| 80000 | 9 | 9 / 4 = 2.25 | 6.90705 |
| 160000 | 23 | 23 / 9 = 2.55556 | 8.31515 |
| 320000 | 36 | 36 / 23 = 1.56522 | 6.15167 |
| 640000 | 74 | 74 / 36 = 2.05556 | 5.99475 |
| 1280000 | 144 | 144 / 74 = 1.9459 | 5.54523 |
| 2560000 | 295 | 295 / 144 = 2.04861 | 5.41319 |
| 5120000 | 597 | 597 / 295 = 2.02373 | 5.23165 |

由上表可知，2倍的input size，執行時間成長略為2倍，且執行時間與nlgn有接近正比的關係，若n為input size，f(n)為此演算法執行的時間，大致符合Θ(f(n)) = n\*lg(n)的趨勢。

二、測試演算法正確性

1. 將第48行註解消除掉，可印出Input array

一張含有 文字 的圖片

自動產生的描述

2. 調整第184、195行為固定size，即可準備測試

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

3. 編譯並執行程式，下方圖則是size=10的範例，max = 14，也就是3+8-6-2+7+4=14，可試著調整不同size看看演算法是否正確

一張含有 文字 的圖片

自動產生的描述

三、心得

這次是第二次演算法的作業，雖與第一次題目不同，但要求很相似，因此我套用第一次作業的格式，並設計出Brute-force及Divide and Conquer演算法來解決Maximum subarray的問題，並觀察兩者執行時間的規律。使用Brute-force的方法來解真的是很沒有效率的演算法，再輸入大小超過10000之後執行時間就非常久，使用Divide and Conquer的方法相較起來有效率很多。細微的探究兩者執行時間的Θ(f(n))，我發現不應該像第一次使用Python畫出兩者執行時間對輸入大小的曲線，因為劃出來仍看不出來執行時間的Θ(f(n))。於是，我逐次調整for迴圈輸入大小兩倍，會發現很接近Θ(f(n))。另外，我每次都印出執行時間/理想Θ(f(n))，會發現有點接近一個常數，無法完全證明理想Θ(f(n))實測結果是百分之百對，但可以說非常接近實測結果了。總之，第二次作業並不難，不過要得出執行時間並觀察這些數據需要花滿多時間跟心力。