Homework 1

B06303131 經濟五 沈家睿

October 16, 2021

Problem 5 - Time Complexity & Recurrence

(1) **Asymptotic Notations**

(a) $\ln n! = O(\ln n^n)$

$\ln n! = \ln n + \ln n - 1 + ... + \ln 1 \leq \ln n + \ln n + ... + \ln n = n \ln n$, $\forall n \geq 1$

所以取 $c = 1,\ n_0 = 1$, $\ln n! \in O(\ln n^n)$

(b) $n^{\ln c} = \Theta(c^{\ln n})$

$n^{\ln c} = c^{\ln n} \leq 2(c^{\ln n}), \forall n \geq 1$

取 $c = 2,\ n_0 = 1$

因此 $n^{\ln c} \in O(c^{\ln n})$

$n^{\ln c} = c^{\ln n} \geq 0.1(c^{\ln n}), \forall n \geq 1$

取 $c = 0.1,\ n_0 = 1$

因此 $n^{\ln c} \in \Omega(c^{\ln n})$

結合以上兩個條件可得 $n^{\ln c} \in \Theta(c^{\ln n})$

(c) $\sqrt{n} = O(n^{\sin n})$

$\sqrt{n} = n^{\frac{1}{2}}$ 而 $\sin n \in [-1, 1]$

e.g. 當 n 取 $\frac{3\pi k}{4}$, 且 k 屬於正整數即產生反例

$n^{\sin n}$ 不可能恆大於等於 $\sqrt{n}$

因此 $\sqrt{n} \notin O(n^{\sin n})$

(d) $(\ln n)^3 = o(n)$

$\lim\limits_{n\to\infty} \frac{n}{(\ln n)^3}$ 使用羅畢達法則求極限值

$$\lim_{n\to\infty} \frac{n}{(\ln n)^3} \overset{L.H.}{=} \lim_{n\to\infty} \frac{n}{3(\ln n)^2} \overset{L.H.}{=} \lim_{n\to\infty} \frac{n}{6(\ln n)} \overset{L.H.}{=} \lim_{n\to\infty} \frac{n}{6} = \infty$$

因此 $(\ln n)^3 \in o(n)$

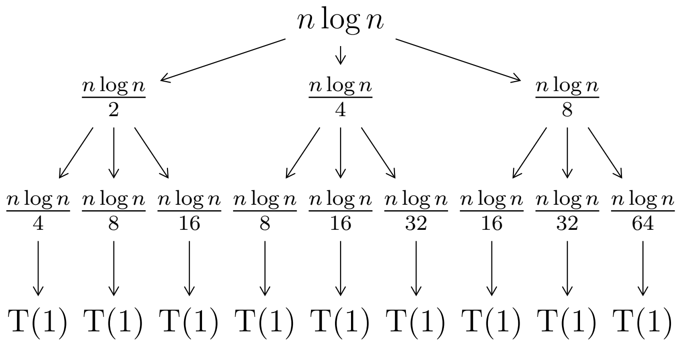
(2) **Solve Recurrences**

(a) $T(n) = 2T(n{-}1) + 1$

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2[T(n-2) + 1] + 1 \\ &= ... = 2^n T(0) + \sum_{i=0}^{n-1} 2^i \\ &= 2^{n+1} - 1 \end{aligned}$$

因此 T(n)$\in O(2^n)$ 且 T(n)$\in \Omega(2^n)$
結合以上兩個條件 T(n) $\in \Theta(2^n)$

(b) $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(\frac{n}{8}) + n\log n$



對於每層而言需花費額外 conquer 的時間, 且會 divide 成三個子問題

因此 Recursion Tree 會如上圖所示

對於第一層的 cost 爲 $n\log n$

第二層爲 $\frac{n\log n}{2} + \frac{n\log n}{4} + \frac{n\log n}{8} = \frac{7n\log n}{8}$

第三層則爲 $\frac{49n\log n}{64}$

依此類推下去直到收斂爲 T(1) 爲止

另外這整棵樹的結構實際上並非對稱, 而是左低右高的結構

因此 total cost=$n\log n \sum_{i=0}^{k}(\frac{7}{8})^i$, 其中 $k \in [\log_8 n, \log_2 n]$

則 total cost $\geq n\log n$, 因此 T(n)$\in \Omega(n\log n)$
若 $k = \infty$ 則 $\sum_{i=0}^{k}(\frac{7}{8})^i = 8$

因此 total cost $\leq 8(n\log n)$, i.e.T(n)$\in O(n\log n)$

結合以上兩個條件, 可得 T(n)$\in \Theta(n\log n)$

(c) $T(n) = 4T(\frac{n}{2}) + n\log n$

$n^{\log_b a} = n^{\log_2 4} = n^2$
因爲 $f(n) = n\log n \in O(n^{2-\epsilon})$, $\epsilon = 0.01 > 0$
因此爲 case1,$T(n) \in \Theta(n^2)$

(d) $T(n) = \sqrt{n}T(\sqrt{n}) + n$

$$\begin{aligned} T(n) &= \sqrt{n}T(\sqrt{n}) + n, \text{ 令 } n = 2^k \\ T(2^k) &= 2^{\frac{k}{2}}T(2^{\frac{k}{2}}) + 2^k, \text{ 接著同除 } 2^k \\ \frac{T(2^k)}{2^k} &= \frac{T(2^{\frac{k}{2}})}{2^{\frac{k}{2}}} + 1, \text{ 接著令 } S(k) = \frac{T(2^k)}{2^k} \\ S(k) &= S(\frac{k}{2}) + 1 \end{aligned}$$

使用 $Master\ Theorem$ 求解複雜度, 可得爲 $case2$, 因此 $S(k) \in \Theta(\log k)$
而 $S(k) = \frac{T(2^k)}{2^k} \implies T(2^k) = 2^k S(k)$
因此 $T(n) \in \Theta(n\log\log n)$

Problem 6 - Ghost Leg

Reference: b08501098 柯晨緯 & b08208032 胡材溢

(1) 題目所求之條件等價於求解逆序數對量
因此使用 merge sort 求解
對著 B 進行 merge sort
以下假設 merge 時
左邊的 index 令爲 i, 右邊的 index 令爲 j
在 merge 時, 初始的 i, j 皆爲 0
因左邊與右邊的陣列具單調遞增之特性
且對於左邊的元素其原本陣列之 index 皆小於右邊元素的 index
因此可在 O(N) 時間內算出逆序數對的數量
每次拿出一個左邊的元素往右邊的陣列看去
將 j 移動到小於這一元素的 index
則此一左邊元素的逆序數對爲 j
下個左邊元素取出來時再從這個 j 往後找到小於這個左邊元素爲止
依此類推下去
左邊元素遍歷完時即可算出所有逆序數對數量
演算法複雜度爲 $T(N) = 2T(\frac{N}{2}) + O(N) + O(N)$
因此依然爲 $O(N \log N)$

(2) 如上題所述本質上我的演算法依然是在做 merge sort
只是我在每次的 conquer 時我會額外花一次 O(N) 算解
接著再進行 O(N) 的 merge
這樣我每次的 conquer 依然是 O(N)
如此一來我的演算法複雜度仍與 merge sort 一樣爲 $O(N \log N)$

(3) 因爲對於 bubble sort 而言
每次的交換都會消滅一個逆序數對
當 bubble sort 完成時的陣列並不存在任何的逆序數對
因此 bubble sort 的交換數就會等於逆序數對的數量

(4) 我可以先將 constraint 放到正確的 index 上
上述的步驟會花 O(N)
接著再將剩餘的數字有序的放入空的 index
此一步驟依然會花 O(N)
接著我再執行 (1) 的演算法求解
則這步驟花 $O(N \log N)$
因此所有的步驟總共會花 $O(N \log N)$

(5) 若兩直線中任意多加一條橫 bar
則要再多一條橫 bar 才能達到原本之 constraint
因此達成 constraint 橫 bar 數爲最小即可
且對於每個橫 bar 而言等價於做一次交換
每做一次交換即會產生一組逆序數對
根據 (3) 可知交換次數等價於求解逆序數對之數量
因此可用 (4) 之演算法求解

Problem 7 - Unfair Lucas

Reference:
https://leetcode.com/problems/maximum-sum-circular-subarray/discuss/178422/One-Pass & p10922001 黃佳琪 & b08501098 柯晨緯

(1) 最大值爲 16

(2) **procedure** CASE1(num[N]) ▷ max in the middle

```
procedure Case1(num[N])                          ▷ max in the middle
    cur_max ← num[0]
    ans_max ← num[0]
    start ← 0
    end ← 0
    for i = 1 to N-1 do
        if num[i] > ans_max then
            start ← i
        end if
        cur_max ← max(cur_max + num[i], num[i])
        if cur_max > ans_max then
            end ← i
        end if
        ans_max ← max(cur_max, ans_max)
    end for
    return ans_max, start, end                   ▷ return max value and start, end
end procedure

procedure Case2(num[N])                          ▷ max cross head and tail
    total_sum ← num[0]
    cur_min ← num[0]
    ans_min ← num[0]
    start ← 0
    end ← 0
    for i = 1 to N-1 do
        total_sum ← total_sum + num[i]
        if num[i] < ans_min then
            start ← i
        end if
        cur_min ← min(cur_min + num[i], num[i])
        if cur_min < ans_min then
            end ← i
        end if
        ans_min ← min(cur_min, ans_min)
    end for
    return total − ans_min, start, end           ▷ return max value and start, end
end procedure
```

環形陣列求解最大值需分兩種 case 討論
case 1.
不會跨越頭尾, 則問題等價於一般陣列求 max sum of subarray
case 2.
跨越頭尾, 則問題等價於全部元素和減去 min sum of subarray
簡單證明如下

$$\begin{aligned} max(prefix + suffix) &= max(totalsum - subarray) \\ &= totalsum + max(-subarray) \\ &= totalsum - min(subarray) \end{aligned}$$

另外當 max sum of subarray 爲負值時代表元素全爲負值
因此須直接取 max sum of subarray 而不用算 case 2.
因爲全爲負值 min sum of subarray 爲全取
一旦與 total sum 相減則爲全不取違反題意
因爲在遍歷時有額外開兩個變數 start, end 紀錄頭尾
因此可以藉由這兩個變數在 O(N) 時間內重建所選取的範圍
空間複雜度爲 O(1)
因爲兩種 case 中皆只有一個迴圈所以時間複雜度爲 O(N)

(3)

```
procedure Case3(num[N])                          ▷ max in the middle and skip once
    cur_max ← num[0]
    cur_max_skip ← 0                             ▷ should skip once current max value
    ans_max ← num[0]
    skip_index ← 0
    start ← 0
    end ← 0
    for i = 1 to N-1 do
        if cur_max > cur_max_skip + num[i] then
            skip_index ← i
        end if
        cur_max_skip ← max(cur_max, cur_max_skip + num[i])
        if num[i] > ans_max then
            start ← i
        end if
        cur_max ← max(cur_max + num[i], num[i])
        if cur_max_skip > ans_max then
            end ← i
        end if
        ans_max ← max(cur_max_skip, ans_max)
    end for
    return ans_max, skip_index, start, end
end procedure
```

```
procedure Case4(num[N])                ▷ max cross head and tail and skip once
    Left[N] ← (0, 0, 0), Right[N] ← (N − 1, N − 1, 0) ▷ start, end, MinValue
    total_sum, cur_min, ans_min ← num[0]
    start, end ← 0
    for i = 1 to N-1 do
        total_sum ← total_sum + num[i]
        Left[i] ← (start, end, ans_min)
                                        ▷ Left subarray MinValue exclude num[i]
        if num[i] < ans_min then
            start ← i
        end if
        cur_min ← min(cur_min + num[i], num[i])
        if cur_min < ans_min then
            end ← i
        end if
        ans_min ← min(cur_min, ans_min)
    end for
    cur_min, ans_min ← num[N − 1]
    start, end ← N − 1
    for i = N-2 to 0 do
        Right[i] ← (end, start, ans_min)
                                        ▷ Right subarray MinValue exclude num[i]
        if num[i] < ans_min then
            start ← i
        end if
        cur_min ← min(cur_min + num[i], num[i])
        if cur_min < ans_min then
            end ← i
        end if
        ans_min ← min(cur_min, ans_min)
    end for
    skip_index ← 0, skip_range ← (0, 0), ans_min ← ∞
    for i = 0 to N-1 do
        cur_min ← min(num[i] + Left[i].min, num[i] + Right[i].min)
        if cur_min < ans_min then
            skip_index ← i
            if cur_min == num[i] + Left[i].min then
                skip_range ← (Left[i].start, Left[i].end)
            else
                skip_range ← (Right[i].start, Right[i].end)
            end if
        end if
        ans_min ← min(cur_min, ans_min)
    end for
    return total − ans_min, skip_index, skip_range
end procedure
```

此題比上題多了兩種 case
case 3.
最大值一樣產生在中間
但一定要跳過一個元素
因此使用額外的變數 cur_max_skip 紀錄此狀態
cur_max_skip 的更新機制爲
當前面有跳過一個元素時則當前的 num[i] 不能跳過
或當前沒跳過元素時 num[i] 可跳過
因此可得最大值產生在中間且一定跳過一個元素
case 4.
爲貫穿頭尾且必跳過一個元素之最大值
所求爲全部元素和減去跳過的一個點及一段區間
因此多開兩條陣列 Left, Right 使其可在 O(N) 內完成
Left[i] 所存資料爲
不含 num[i] 之左邊子陣列的最小值區間及其頭尾的資訊
Right[i] 所存資料爲
不含 num[i] 之右邊子陣列的最小值區間及其頭尾的資訊
接著就可利用這兩條陣列在 O(N) 時間內算出
一個欲跳過的點加上一個區間的最小值
最後一樣用全部元素和減去此一最小值即爲此 case 中的最大值
最後此題所求爲 case1,case2,case3,case4 中最大值
且在遍歷時皆有紀錄頭尾的資訊
因此可以在 O(N) 時間內重建所選區間
因爲是多開兩條長度爲 N 的陣列
因此空間複雜度爲 O(N)
全部的 case 皆在 O(N) 時間內可完成因此時間複雜度爲 O(N)