ADA Hand-writing

B05902002 資工二 李栢淵

Problem 5

(1) Given each pair of expressions, A and B, in the following table, please find the relation between A and B(A is O, o, Ω, ω, or Θ of B). Assume that k >= 1, E > 0 and c > 1 are constants.

(a) A is O or o of B
(b) A is none of B
(c) A is Ω or ω of B
(d) A is O, Ω or Θ of B
(e) A is O, Ω or Θ of B

(2)

(a)(30%) Give the pseudocode for an optimal divide-and-conquer algorithm that outputs the time to approach and the time to disconnect given the mood sequence. Assume that the ups and downs of mood values are stored in an array M[0...n].

```
typedef struct pair {
	int start;
	int end;
	int changing;
}Pair;

void find_the_most_positive_mood_changing(int M[], n) {
	Pair ans = find_the_max_changing(M[], 0, n);
	printf("%d%d\n", ans->start, ans->end);
	return;
}

Pair find_the_max_changing(int M[], int head, int end) {
	if (head == end)    return (head, end, 0);
	int mid = (head+end)/2;
	Pair out1 = find_the_max_changing(M[], head, mid);
	Pair out2 = find_the_max_changing(M[], mid+1, end)
	Pair out3 = find_the_max_crossing(M[],head, mid, end)
	if (out1->changing >= out2->changing && out1->changing > out3->changing)
		return out1;
	else if (out2->changing > out1->changing && out2->changing > out3->changing)
		return out2;
	else
		return out3;
}
```

```
Pair find_the_max_crossing(int M[], int head, int mid, int end) {
	Pair out;
	int head_min = 2147483647;
	for (int i = mid; i >= head; i--) {
		if (M[i]<head_min)
			head_min = M[i];
	}
	int end_max = -2147483648;
	for (int i = mid; i >= head; i++) {
		if (M[i]>end_max)
			end_max = M[i];
	}
	out->start = head_min;
	out->end = end_max;
	out->changing = end_max - head_min;
	return out;
}
```

(b) (20%) Please show the complexity of your algorithm using Θ and briefly explain how you derive this bound.

1. Divide a list of size n into 2 subarrays of size n/2:
   => Θ(1)

2. Recursive case:
   find_the_max_changing for each subarrays
   => 2T(n/2)
   Base case (head = end):
   return 0
   => Θ(1)

3. find_the_max_crossing for the original list: Θ(n)

$$T(n) = \begin{cases} \Theta(1) & , n = 1 \\ 2T(n/2)+\Theta(n) & , n > 1 \end{cases}$$

=> T(n) = Θ(nlogn)

Problem 6

(1) Solve the following recurrences. Assume that T(n)=1，∀ n<=2.

(a) T(n) = 4T(n/2) + n

Prove or disprove that T(n) = O(n^2).

Suppose $T(n) \le c_1 n^2 - c_2 n$ for n = m-1, m-2, ...

When n = m,

$T(n) = 4T(n/2) + n$

$\le 4c_1(n/2)^2 - 4c_2(n/2) + n$

$= c_1 n^2 - 2c_2 n + n$

$= c_1 n^2 - c_2 n - (c_2 - 1)n$

$\le c_1 n^2 - c_2 n$

(when $c_2 - 1 \ge 0$)

When n=1 and $c_2 \ge 1$, pick a big enough c_1

s.t. $T(n) = 1 \le c_1 n^2 - c_2 n$

so we set $c_1 = 2$, $c_2 = 1$, $n_0 = 1$

that T(n) = O(n^2).

(b) T(n) = 2T(n/2) + n/lg(n)

Prove or disprove that T(n) = O(n).

T(n) = n/lg(n) + 2*(n/2)/lg(n/2) + 4*(n/4)/lg(n/4) +…

= n[1/lg(n) + 1/lg(n/2) + 1/lg(n/4) + …] (共lg(n)項)

let lg(n) = k

= (2^k)[1/k + 1/(k-1) + 1/(k-2) +1/(k-1) + …

+ 1/3 + 1/2 + 1/1]

> (2^k)[1/1 + (1/2)+ (1/4+1/4)+ (1/8+1/8+1/8+1/8)+ …]

> (2^k)[lg(k)/2] (有一個1以及lg(k)-1個1/2)

= n[lg(lg(n))/2]

所以T(n) != O(n)

(c) T(n) = 9T(n/3) + n^3

Prove or disprove that T(n) = O(n^3).

使用遞迴樹法

T(n) = n^3 + 9*(n/3)^3 + 81*(n/9)^3 + . . .

= (n^3)*(1 + 1/3 + 1/9 + 1/27 + . . .) (共log₃n項)

= C*n^3 (因為是公比小於一的等比數列，C為無限項之和)

=> T(n) = O(n^3)

(d) T(n) = T(n/5) + T(7n/10) + n
Prove or disprove that T(n) = O(n).
使用遞迴樹法
T(n) = n + (9/10)*n + (9/10)^2*n + (9/10)^3*n + . . .
= n*(1 + 9/10 + (9/10)^2 + (9/10)^3 + . . .)
= C*n （因為是公比小於一的等比數列，C為無限項之和）

=> T(n) = O(n)

(e) T(n) = n^(1/2)*T(n^(1/2)) + n
Prove or disprove that T(n) = O(n log log n).

let k = lg(n) => n = 2^k
原式=>T(2^k) = 2^(k/2)*T(2^(k/2)) + 2^k

let S(k) = T(2^k)
S(k) = 2^(k/2)*S(k/2) + 2^k
Guess S(k) = c_1(2^k)*lg(k)

Suppose S(k) ≤ c_1(2^k)*lg(k) for k = m-1, m-2, ...
When k = m,
S(k) = 2^(k/2)*S(k/2) + 2^k
≤ 2^(k/2)*[c_1(2^(k/2))*lg(k/2)] + 2^k
= c_1*(2^k)*(lg(k)-1) + 2^k
= c_1(2^k)*lg(k) - c_1*(2^k) + (2^k)
= c_1(2^k)*lg(k) - (c_1-1)*(2^k)
≤ c_1(2^k)*lg(k)
(when c_1-1 ≥ 0)

When k=1 and c_1 ≥ 1, pick a big enough c1
s.t. S(k) = 1 ≤ c_1(2^k)*lg(k)
so we set c_1= 1, k_0 = 2
that S(k) = O((2^k)*lg(k))
=> T(2^k) = O((2^k)*lg(k))
=> T(n) = O(n*lg(lg(n)))

(2) In an n X m binary matrix (i.e., every element 2 {0, 1}), find the largest square sub-matrix which only contains 1s.
Prove or disprove that the time complexity of this problem is O(nm).

方法：

首先建一個大小跟原本binary matrix一樣的的二維陣列S[m][n]，用來表示以自己為右下角，可以有多大的都是1的sub-matrix(如圖)，接著我們做dynamic programming，從左上開始，從左到右，再從上到下，只要原本陣列是0，就填0，原本陣列是1，再判斷他的左邊、左上和上面的S陣列是多少，如果都是小於等於n，就填n+1，在過程的時候一起更新此陣列中的最大值M，如有必要也可以記錄M在哪一列哪一行。

說明：

假設我們現在要填S[a][b]，原本的矩陣叫MAP，此時，如果原本MAP[a][b]是1，就先在S[a][b]填1，代表說自己起碼有個1*1的都是1的sub-matrix，這時候在看左邊(s[a-1][b])、左上(s[a-1][b-1])和上面(s[a][b-1])的值多少，就代表的他有一個多大的都是1的sub-matrix，找其中的最小值n，就可以確保在包涵MAP[a][b]裡，有一個(n+1)*(n+1)的都是1的sub-matrix，如此一來就可以知道s[a][b]的值為n+1。

如此一來，把整個陣列跑一遍就能找到最大的、只包含1的square sub-matrix，所以，時間複雜度為O(mn)