Bài 7) Ước lượng nhanh O.

a)

Ta dễ nhìn thấy 2 hàm fifth_Element, Partial_Sum liên tiếp nhau

+ Xét trong hàm fifth_Element ta có nó chỉ có trả về lệnh trả

T(n) => độ phức tạp O(1)

+ Xét trong hàm Partial_Sum ta có 3 câu lệnh trên.

- sum = 0 => O(1)

- Xét for (int i = 0; i < 42; i++) Vòng for lặp lại 42 lần vì nó chạy từ 0 → 41

sum = sum + A[i]; O(1).

- Xét return sum; O(1)

Ta có 3 câu lệnh trên là liên tiếp => độ phức tạp của hàm

Max (O(sum; O(for), O(return).)

= Max (O(sum); O(for).O(sum); O(return)) = O(42)

Vậy T(n) = O(42).

(1)

```
void sum_First_n(Int n)
{
                O(1)        O(1)
    Int i, sum = 0;                          (1)
O(n)  *  for (i = 1; i <= n; i++) → O(n)    (2)
              sum = sum + i;       O(1)     (3)
}
```

(2)

```
void an_sum_first_n(Int n)
{
          O(1)   O(1)   O(1)
    Int i, k, sum = 0;                        (1)
O(6n) **  for (i = 1; i <= n; i++) → O(n)     (2)
        *   for (k = 1; k < 7; k++)   O(6)    (3)
                sum = sum + i;        O(1)    (4)
}
```

giải

Xét (I) ta có

(1). bao gồm các hàm gán biến => độ phức tạp là ~~(1)~~ O(1)

Xét (*) ta có

(2) lại ~~là~~ mỗi ~~hàm~~ vòng lặp for nó lặp từ giá trị 1 đến n

=> ~~Đ~~ độ phức tạp là O(n)

(3) là mỗi ~~h~~ hàm cơ bản ~~(x)~~ => O(1)

Vậy (1) có độ phức tạp là $Max(\cancel{O(1)}\ (1), (*)) = O(n)$.

Xét (2) ta có:

(1) là các lệnh gán bình $\Rightarrow O(1)$

Xét $(**)$ ta có:

(2) là vòng lặp for nó chạy từ $1 \to n$ $\Rightarrow$ độ phức tạp là $O(n)$.

Xét $(*)$ trong $(**)$ ta có.

(3) là 1 vòng lặp for nó chạy từ $1 \to 6 \Rightarrow O(6)$.

(4) là phép tính $\Rightarrow O(1)$

$\Rightarrow O(*) = O(1) \cdot O(6) = O(6)$.

Độ phức tạp của $(**)$ là $O(6n)$ $= \cancel{O(6)}\ \cancel{(O(n))}$ $Max(O(6n), O(1))$

$\Rightarrow$ Độ phức tạp của chương trình là $Max(O(6n), O(n)) = O(6n)$

$T(n) = O(6n)$.

c)

```
Int binarysearch ( Int a[ ], Int n, Int val)
{
Int l = 1, r = n, m;        O(1)
    while (r >= 1)          (1)
    {
        m = (l + r)/2;      (1)    (2)

(*)     If (a[m] == val)    (1)    (3)
            return m;              (4)
        If (a[m] > val      (1)    (5)
            r = m-1;               (6)
        else  l = m+1;      (1)    (7)
    }
return -1;  (2)
}
```

Giải

Xét hàm trên ta có

(1) (2) là các câu lệnh khởi tạo và trả về nên ta có độ phức tạp là $O(1)$

Xét (n) ta có

⇒ Ta dễ thấy được rằng cái độ phức tạp của (3), (5), (7) là $O(1)$

- Xét while ta có $n \geq 1$ ⇒ độ phức tạp sẽ tồi tệ hơn giả sử trong trường hợp $a[m] < val$ luôn xảy ra thì n sẽ luôn lớn hơn 1.

Vậy $T(n) = \infty$ (đi xa)

TH: $n \geq l$ thì ta mới xác định được rõ độ ~~vô~~ phức tạp

Xét ~~hàm~~ while trong (*) ta có

~~Trong trường hợp thì a[m] ở (3) (1) là xảy ra~~

~~thì ta có độ phức T~~

TH1: Khi $a[m] \geq val$ thì ta có $r = m-1 \Rightarrow \frac{l+r}{2} - 1$

và ta phải xét đến $r \geq l \Rightarrow \frac{l+r}{2} - 1 \geq l \Rightarrow l + r - 2 \geq 2l$

$\Rightarrow r - 2 \geq l$

$\Rightarrow r \geq l + 2$

~~Đến đó~~ Giả sử với n lần xảy ra

thì ta có $r \geq (2n-1) l + 2n$

TH2: Khi $a[m] < val$ luôn xảy ra thì

$r \geq m(m+1)$

$\Rightarrow r - \{2(n-1)+2\} \geq l \cdot l$.

KOKUYO

tương tự khi có sự làm tròn lên cũng đi thay nhưng độ phức tạp là

$\sim O(\log_2(n))$.

Vậy $T(n) = O(\log 2(n))$

(lấy phần giá trị nguyên chặn)

1)

```
int compute_sums (int A[], int n)
{
    int M[n][n];   (1)
    int i, j;      (2)
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            M[i][j] = A[i] + A[j];  (3)
    return M;  (4)
}
```

Giải

Ta đã có (1) và (2) là các lời gọi các biến nên độ phức tạp là O(1)

(4) là trả về giá trị độ phức tạp là O(1)

Xét (**) ta có:

- Trong (*) ta có

+ (3) là gán giá trị cho M[i][j] là tổng của 2 số A[i] và A[j] nên ta có độ phức tạp là O(1)

+ xét vòng lặp for ta có j chạy từ giá trị 0 đến n-1 nên ta có độ phức tạp là O(n).

⇒ Độ phức tạp của (*) là O(n)

- xét vòng lặp for trong (**) ta thấy i chạy từ 0 đến n-1 nên ta có độ phức tạp là O(n).

Độ phức tạp trong (**) là ~~O(...)~~ O(n^2)

= O(n) × max(O(n), O(
(for ngoài) (for trong)

Vậy độ phức tạp của chương trình là

$$T(n) = O(n^2)$$

e) (1) s = 0;

(2) i = 1;

(3) while (i ≤ n)

{

(4) j = n - i;

(**) (5) while (j ≥ 1)

(*) {

(6) s = s + 1; (1)

(7) j = j - 1; (1)

}

Giải

Ta có (1), (2), (4), (6), (7), (8) là các gán giá trị ~~ko~~ của các lệnh nên độ phức tạp của nó chỉ là $O(1)$

Xét trong (**) ta có

+ Vòng While ta có $i \le n$ mà $i = 1$ và nó sẽ tăng lên 1 đơn vị khi ở cuối vòng While → nên độ phức tạp của nó ~~của~~ là $O(n)$

* Xét trong (*) ta có.

Vòng lặp While ta có $j \geq 1$ mà $j = n - i$, và khi thực vòng lặp ta lại có $j = j - 1$; nên ta thấy j lại phụ ∈ vào i nên ta thấy được độ phức tạp của nó là $O\left(\frac{n-1}{2}\right)$

→ Độ phức tạp của (**) là $O\left(n \cdot \frac{n-1}{2}\right)$

= Vậy độ phức tạp của chương trình là Max$\left(O(1), O\left(n \cdot \frac{n-1}{2}\right)\right)$ =

Vậy $T(n) = O\left(\frac{n^2 - n}{2}\right)$

Giải

Ta có (1), (2), (4), (6), (7), (8) là cái gán giá trị ~~lo~~ của các biến

nên độ phức ~~ph~~ tạp của nó chỉ là $O(1)$

Xét trong (**) ta có

+ Vòng While ta có $i \le n$ mà $i = 1$ và nó sẽ tăng lên 1 đơn vị

~~khi~~ ở cuối vòng While → nên độ phức tạp của nó ~~của~~ là $O(n)$

* Xét trong (*) ta có.

Vòng lặp While ta có $j \geq 1$ mà $j = n-i$, và kết

thúc vòng lặp ta lại có $j = j - 1$; nên ta thấy j lại phụ ∈ vào i nên

ta thấy được độ phức tạp của nó là $O\left(\frac{n-1}{2}\right)$

→ Độ phức tạp của (**) là $O\left(n \cdot \frac{n-1}{2}\right)$

= Vậy độ phức tạp của chương trình là ~~Q~~ $Max\left(O(1), O\left(n \cdot \frac{n-1}{2}\right)\right)$

Vậy $T(n) = O\left(\frac{n^2-n}{2}\right)$