Data Structures #2 　　　　　　　　　　 112590022 陳騏逸

1. (a) n: decimal number

(1) initial empty string binary

(2) while n is greater than 0:

$n \% 2$ corresponds to the LSB

⇓

Append the remainder to the left of binary

update $n = n // 2$

$$\begin{array}{l} 2 \,\underline{|\, n \quad} \cdots\ n\%2 \ \text{(MSB)} \\ 2 \,\underline{|\, n/\!/2} \cdots\ (n/\!/2)\%2 \\ \quad \vdots \\ \text{(LSB)} \end{array}$$

(b)

```
Define Decimal_to_binary_Iterable(n):
    binary = empty string
    while(n is not 0):
        if (n % 2 is 0) ⇒ '0' + binary
        else ⇒ '1' + binary
        n = n // 2
    return binary
```

(c)

```
Define Decimal_to_binary_Recursive(n):
    if (n is 0) then return empty string
    else return Decimal_to_binary_Recursive(n//2) + Int_to_string(n%2)
```

2.

(a) $c_1 x^2 \le f(x) \le c_2 x^2$, $x \ge x_0$

$\hookrightarrow 4x^2+2x+1 \le 4x^2+2x^2+x^2 = 7x^2$ 　 $2x+1<0$ 　 $x<-\frac{1}{2}$

$\hookrightarrow 4x^2+2x+1 \ge 4x^2$

$\Rightarrow c_1 = 4,\ c_2 = 7$, $x_0 \ge 1$ or $-\frac{1}{2} < x_0 < -\frac{1}{3}$

$\therefore 4x^2+2x+1 = \theta(x^2)$ #

$4x^2+2x+1 > 7x^2$, $3x^2-2x-1<0$

$\begin{array}{ll} 1x & -1 \\ 3x & 1 \end{array}$ 　 $(x-1)(3x+1) < 0$

(b) $c_1 \log(x) \le f(x) \le c_2 \log(x)$, $x \ge x_0$

$\hookrightarrow x\log x + \sqrt{x} \le 2x\log x$

$\hookrightarrow x\log x + \sqrt{x} \ge x\log x$ 　 $\sqrt{x} \ge 0$

$\Rightarrow c_1 = 1,\ c_2 = 2x$, $x_0 = 10^{-\frac{x}{2}}$

$\therefore x\log x + \sqrt{x} = \theta(x\log x)$ #

$x\log x \ge \sqrt{x}$, $x \ge 10^{-\frac{x}{2}}$

3. (a)

$2^{n+1} \le c \cdot 2^n$

$2^n \times 2 \Rightarrow c = 2$ 　 $\therefore 2^{n+1} = O(2^n)$ #

(b)

$2^{2n} \le c \cdot 2^n$

$(2^n)^2 \le c \cdot 2^n$

$2^n \le c$ 　 no such constant can satisfy it

if $c = 10 \Rightarrow 2^4 > 10$ #

if $c = 30 \Rightarrow 2^5 > 30$

$\vdots$

4. (a)

$(\square , \square) \updownarrow C(5,2)$

$$C(n,2) = \frac{n!}{(n-2)!\,2!} = \frac{n\cdot(n-1)}{2!} = \frac{n(n-1)}{2} \#$$

(b) list ⇒ num_list , n ⇒ len(num_list) , count = 0

```
for i from 0 to n:
    for j from i+1 to n:
        if (num_list[i] > num_list[j]) count += 1
```

	j		
i=1	2 ~ n-1	⇒ n-2 times	
i=2	3 ~ n-1	⇒ n-3 times	$O(n^2)$ #
⋮	⋮	⋮	
i=n-2	n-1	⇒ 1 times	
i=n-1	×		

(c) Define Count_Inversion_Recursive(P, i):

```
if (i is equal to num of P) then return 0
total = 0
for j from i+1 to num of P:
    if P[j] < P[i] then total += 1
return total + count_Inversion_Recursive(P, i+1)
```

$O(n^2)$

time complexity = $O(n^2)$ #
equal to (b)

time complexity = $O(n\log n)$ #

(d) Define Count_Inversion_two recursive(P, left, right):

```
if left+1 < right and right < len(P) then
    mid = (left + right) // 2
    total = Count_Inversion_tworecursive(P, mid+1, right) + Count_Inversion_tworecursive(P, left, mid)
    total += counting(P, left, right)
    return total
elif left+1 == right then return 1 if P[left] > P[right] else 0
else return 0
```

```
for i from left to mid+1
    for j from right down to mid
        if P[i] > P[j] : total += 1
```