5. LOG$_2$n이란?

먼저, 복잡도가 **O(log$_2$n)**이 되는 경우는 다음 2가지이다.

2배씩 증가 : 1 2 4 8 16 32 62 128 256 …

1/2씩 감소 : 1000 500 250 125 …

// 다음 표는 단계수가 1씩 증가할 때 **2배씩 증가**는 값을 나타낸다.

2배씩 증가	1	2	4	8	16	32	• • •	**n**
단계수 k	2^0	2^1	2^2	2^3	2^4	2^5	• • •	2^k

$\therefore\ 2^k = n$

k = $\log_2 n$ → 어떤 값이 **2배씩 증가**하여 n이 되면, 단계수 k는 **log$_2$n**이 된다.

- 예를 들면, 정렬 알고리즘 2-way merge sort에서 단계수가 **O(log$_2$n)**이다.
- 이유는 n개의 자료를 2개씩 합병하는 방식으로 처리하기 때문이다.
- 해서, 2-way merge sort의 시간복잡도는 **O(nlog$_2$n)**이다.

// O(log$_2$n)과 O(n)의 관계

C 코드	수행 횟수(a++;)	빅오 표현
① for(i=1; i<=n; i++) a++;	n번	O(n)
② for(i=1; i<=n; i+=2) a++;	n/2번	O(n)
③ for(i=1; i<=n; i+=5) a++;	n/5번	O(n)
④ for(i=1; i<=n; **i*=2**) a++;	i = 1, 2, 4, 8, 18, 32, 62, … 즉, **log$_2$n**번	**O(log$_2$n)**

// 자료구조에서 복잡도를 나타낼 때

- $O(\log_2 n)$에서 밑 **2**를 생략하고,
- $O(\log n)$ 또는 $O(\lg n)$으로 나타내기도 한다.

기출문제 분석

1. 다음 알고리즘의 시간복잡도를 빅세타(Θ) 표기법으로 표현한 것은? (단, n>0) [2016년 국가 7급]

```
-----------------------------------------
int iterate(int n) {
  int i, count = 0;
  for (i = 1; i < n; i *= 2) count++;
  return count;
}
-----------------------------------------
```

① $\Theta(\log n)$ ② $\Theta(n)$ ③ $\Theta(n \log n)$ ④ $\Theta(n^2)$

♤ 시간복잡도 – 빅세타(Θ) 표기법

---

for (i = 1; i < n; i *= 2) count++; //i 값이 2배식 증가 : $\Theta(\log n)$

정답 : ①

2. 다음은 C 프로그램의 일부이다. 이 프로그램의 시간복잡도는? [2008년 국가 7급]

```
----------------------------
for(i = 1; i < n; i++){
  m = 1;
  while(m<i) m = m * 2;
}
----------------------------
```

① $O(n^2)$ ② $O(\log n)$ ③ $O(n\log n)$ ④ $O(n)$

♤ 시간복잡도

```
for(i = 1; i < n; i++)          //바깥 루프 : 단계수 n
{
   m = 1;
   while(m<i) m = m * 2;    //안쪽 루프 : 단계수 log2 n → 2배씩 증가하므로
}
```

• 시간복잡도 = n × $\log_2 n$ = $O(n\log_2 n)$ = $O(n\log n)$ → 로그 함수의 밑 2는 생략 가능

정답 : ③

3. 다음 코드의 시간복잡도를 바르게 나타낸 것은? [2010년 국가 7급]

```
for(i = 1; i <= n; i++)
    for(j = 1; j <= n; j = j + i)
        for(k = 1; k <= n; k++)
            x = x + k + 1;
```

① $\Theta(n^2)$ ② $\Theta(n^2 \log n)$

③ $\Theta(n^3)$ ④ $\Theta(n^3 \log n)$

♤ **시간복잡도 – 매우 어려운 문제라고 생각될 수 있다.**

• 먼저, 주어진 코드에 대한 시간복잡도는 어려운 문제일 수 있다.

• **두 번째 for문**의 매개변수 j의 증가분 값에 따른 for문의 **평균 반복횟수**를 구하는 것이 관건이다.

i	j의 증가(j = j + i)	두 번째 for문의 반복횟수
1	1 2 3 4 5 6 7 ..... n	n
2	1 3 5 7 9 ..... n	n/2
3	1 4 7 10.....n	n/3
:		
n	1	1

• **두 번째 for문의 평균 반복횟수** = (n + n/2 + n/3 + ... + 1) / n

= n(1 + 1/2 + 1/3 + ... + 1/n) / n

= 1 + 1/2 + 1/3 + ... + 1/n

• 여기서, 1 + 1/2 + 1/3 + ... + 1/n = **Θ(log n)** → 별도로 정리한다.

// 주어진 코드의 시간복잡도

```
for(i = 1; i <= n; i++)          → Θ(n)
    for(j = 1; j <= n; j = j + I)  → Θ(log n)
        for(k = 1; k <= n; k++)    → Θ(n)
            x = x + k + 1;
```

∴ **복잡도** = $\Theta(n^2 \log n)$

◈ 추가 설명

• 주어진 문제에서 log의 밑수(base)가 얼마인지 제시되지 않았다.
• log의 밑수를 2라고 가정하면 이 문제는 쉽게 증명할 수가 있다.
• 단지, 전산 시험에서는 log의 밑수를 2라고 가정하고 푼다.

정답 : ②

◈ y = 1 + 1/2 + 1/3 + ... + 1/n

• 먼저, 수식의 합인 y값은 일반항으로 나타낼 수 없다.
• 수식의 합인 y값은 근사값으로 표현할 수 있다.
• 만약, n이 작은 값이 아니면

---

y = 1 + 1/2 + 1/3 + ... + 1/n ≈ ln n

→ 여기서, 'ln n'은 자연대수를 나타낸다.($\ln n = \log_e n$)

ln n = ln 10 ≈ 2.3025851

• 위의 개념은 자료구조에서 알고리즘을 분석할 때 사용되는 것이다.

◈ 기호 ≈

• 기호 ≈는 "근사적으로 같다"라는 의미이다.
• 기호 ≈는 어떤 값을 근사값으로만 표현할 수 있을 때 사용한다.

π ≈ 3.141592.....

4. 유클리드의 최대공약수를 구하는 알고리즘의 시간복잡도는?(단, 공약수를 구하는 과정은 두 수가 $\frac{1}{2}$씩 줄어들면서 결과를 구하게 된다) [2002년 서울 7급]

① $O(2\log n)$ ② $O(n log_2 n)$ ③ $O(\log_2 n)$
④ O(n) ⑤ $O(n^2)$

♤ 유클리드의 최대공약수

• $\frac{1}{2}$씩 줄어들므로 시간복잡도는 $O(\log_2 n)$이다.

정답 : ③

5. 다음 C 코드에서 함수 A와 B는 최대공약수를 구하는 함수이다. 함수 B가 함수 A와 같은 결과를 출력하도록 ㉠, ㉡에 들어갈 내용을 바르게 연결한 것은? [2021년 국가 7급]

```
int A(int x, int y) {
  int r;
  while (y != 0) {
    r = x % y;
    x = y;
    y = r;
  }
  return x;
}
```

```
int B(int x, int y){
  int r;
  if (    ㉠    ) return x;
  else
  {
    r = x % y;
    return    ㉡    
  }
}
```

	㉠	㉡
①	y==0	B(y, r);
②	y==0	B(r, y);
③	y!=0	B(y, r);
④	y!=0	B(r, y);

♤ 최대공약수 구하는 함수

	C	Python

반복함수	int gcd(int a, int b){
	  int r; // 나머지
	  while(b){ // b!=0과 같음
	  r = a % b;
	  a = b;
	  b = r;
	  }
	  return a;
	}
		def gcd(a, b):
		  while b != 0: #나머지!=0
		  r = a % b
		  a = b
		  b = r
		  return a #나머지==0
		print (gcd(24, 30)) #출력 : 6

순환함수
(재귀함수)
	int gcd(int a , int b){
	  if (b==0) // 완료조건
	  return a;
	  else
	  return gcd(b, a%b);
	}
		def gcd(a, b):
		  if b==0: #완료조건
		  return a #나머지==0
		  else:
		  return gcd(b, a%b) #나머지!=0
		print (gcd(24, 30)) #출력 : 6

- 최대공약수를 구하는 여러 알고리즘 중에 가장 유명한 것은 **유클리드 호제법**이다.
- 유클리드 호제법(互除法)은 두 수를 나누어 나머지가 0이 되는 원리를 이용한다.
- **호제법**은 두 수가 서로(互) 상대방 수를 나누어(除) 원하는 수를 얻는 알고리즘을 나타낸다.

정답 : ①

6. 다음 C 함수의 시간복잡도를 빅오(O) 표기법으로 표현한 것은? (단, n>1) [2017년 국가 7급]

```
--------------------------------
 void testing(int  n)
 {
   int i, j, sum = 0;
   for (i = 0; i < n; i++)
     for (j = n; j > 1; j /= 2)
       sum += 1;
 }
--------------------------------
```

① $O(n)$ ② $O(n^2)$
③ $O(n \log n)$ ④ $O(\log n)$

♤ 시간복잡도 – 빅오(O) 표기법

```
void testing(int  n)
{
   int i, j, sum = 0;
   for (i = 0; i < n; i++)          //O(n)
     for (j = n; j > 1; j /= 2)     //O(log n) - 1/2씩 감소하므로
       sum += 1;
}
```

• 시간복잡도 = $n \times \log_2 n = O(n \log n)$

정답 : ③

//탐구----------

$\log_2 n$의 수학적 증명

다음 표는 단계수가 1씩 증가할 때 2배씩 증가는 값을 나타낸다.

단계수 k	2^0	2^1	2^2	2^3	2^4	2^5	.....	2^k
2배씩 증가	1	2	4	8	16	32	.....	n

$\therefore\ 2^k = n$

k = $\log_2 n$ → 어떤 값이 2배씩 증가하여 n이 되면, 단계수 k는 $\log_2 n$이 된다.

[예제 1] 다음 점화식의 시간복잡도를 빅오로 나타내면?

T(1) = 1
T(n) = T(n-1) + n(단, n≥2)

(풀이) T(n) = T(n−1) + n
= [T(n−2) + (n−1)] + n
= [T(n−3) + (n−2)] + (n−1) + n
:
= T(1) + 2 + 3 + 4 + ⋯ + n
= 1 + 2 + 3 + 4 + ⋯ + n
= n(n+1)/2
= $O(n^2)$

[예제 2] 다음 재귀식(점화식)의 시간복잡도를 빅오로 나타내면?

T(1) = 1
T(n) = T(n/2) + 1
(단, n≥2, n/2는 정수 연산이다)

• 이 문제는 아래 풀이처럼 머리 아프게 풀지 않아도 된다. n의 값이 **1/2씩 줄어들기** 때문에 척 보면 답을 알 수도 있다.

(풀이) T(n) = $T(\frac{n}{2})$ + 1

= $[T(\frac{n}{2^2})$ + 1] + 1

:

= $T(\frac{n}{2^k}) + 1 + 1 + \ldots + 1$ → 여기서, $\frac{n}{2^k} = 1$이고 1의 개수는 k개이다.

= $T(1) + k$ → 여기서, T(1) = 1이고, $\frac{n}{2^k} = 1$이므로 k=$\log_2 n$이다.

= 1 + $\log_2 n$

= $O(\log_2 n)$