# 01. 2개의 정수를 서로 교환하는 알고리즘을 의사 코드로 작성해보자.

change(a, b) {

temp <- a // temp holds value of a

a <- b // a has value of b

b <- temp // b has value of temp, or original value of a

return a, b

}

# 02. 사용자로부터 받은 2개의 정수 중에 더 큰 수를 찾는 알고리즘을 의사 코드로 작성해보자.

find\_max(a, b) {

if (a >= b) {

return a

}

else {

return b

}

}

# 03. 1부터 n까지의 합을 계산하는 알고리즘을 의사 코드로 작성해보자.

get\_sum(n) {

sum 🡨 0

for (i, i < n, i++) {

sum += i;

}

return sum

}

# 04. Set(집합) 추상 자료형을 정의하라. 다음과 같은 연산자들을 포함시켜라. Create, Insert, Remove, Is\_In, Union, Intersection, Difference

객체: 원소라고 불리는 데이터들의 모임

함수:

1. Create(s) := 집합 s를 만든다

2. Insert(s, x) := 집합 s에 원소 x를 넣는다

3. Remove(s, x) := 집합 s에 원소 x를 삭제한다

4. Is\_In(s, x) := 집합 s에 원소 x가 있는지 확인한다

5. Union(s1, s2) := 집합 s1과 s2의 합집합을 구한다

6. Intersection(s1, s2) := 집합 s1과 s2의 교집합을 구한다

7. Difference(s1, s2) := 집합 s1과 s2의 차집합을 구한다

# 05. Boolean 추상 자료형을 정의하고 다음과 같은 연산자들을 포함시켜라. And, Or, Not, Xor

객체: 0과 1 (혹은 참과 거짓)

함수:  
1. And(a, b) := a와 b 둘 다 1일 때 1을 출력하고, 아닐 시 0을 출력한다  
2. Or(a, b) := a와 b 중 하나가 1일 때 1을 출력하고, 아닐 시 0을 출력한다  
3. Not(a) := a가 1일 때 0을 출력하고, 0일 때 1을 출력한다  
4. Xor(a, b) := a와 b 값이 다를 시 1을 출력하고, 아닐 시 0을 출력한다

# 06. 다음과 같은 코드의 시간 복잡도는? 여기서 n이 프로그램의 입력이라고 가정하자. for (i = 1; i < n; i \*= 2) { printf("Hello"); }

O(log(n))  
t 횟수 뒤에 프로그램이 끝난다고 가정하면, n=2ᵗ일 때 프로그램이 끝난다. 즉, log₂(n) 횟수 뒤에 끝난다.

# 07. 다음과 같은 코드의 시간 복잡도는? 여기서 n이 프로그램의 입력이라고 가정하자. for (i = 0; i < n; i++) { for (j = 1; i < n; j \*= 2) { printf("Hello"); } }

O(n log(n))  
안쪽의 for loop 같은 경우는 시간 복잡도가 log₂(n)인데 바깥쪽 for loop으로 인해 안쪽 루프가 n번 실행됨으로 n log₂(n) 횟수 실행된 뒤 프로그램이 종료된다.

# 08. 시간 복잡도 함수 n² + 10n + 8를 빅오 표기법으로 나타내면? (1) O(n) (2) O(n log₂(n)) (3) O(n²) (4) O(n²log₂(n))

(3)  
가장 큰 차수가 시간 복잡도 함수에서 가장 큰 영향을 미치므로 다른 항들은 무시할 수 있다.

# 09. 시간 복잡도 함수가 7n + 10 이라면 이것이 나타내는 것은 무엇인가? (1) 연산의 횟수 (2) 프로그램의 수행시간 (3) 프로그램이 차지하는 메모리의 양 (4) 입력 데이터의 총개수

(1)  
입력 데이터가 n개일 때 프로그램에서 연산을 몇 번 하는지를 측정한다.

# 10. O(n²)의 시간 복잡도를 가지는 알고리즘에서 입력의 개수가 2배로 되었다면 실행시간은 어떤 추세로 증가하는가? (1) 변함없다 (2) 2배 (3) 4배 (4) 8배

(3)  
기존에 t개의 입력 데이터를 가졌다고 가정하면, 프로그램이 t² 횟수 후에 종료되었을 것이다. 입력데이터가 2배가 되면, 4t²의 횟수가 되는데, O(n)은 여전히 O(n²)으로 유지되지만 4배로 증가한다.

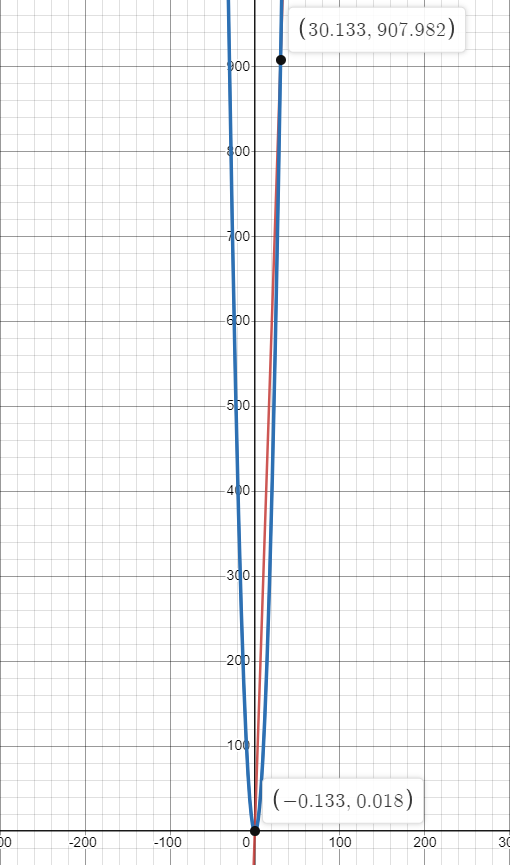
# 11. f(n)에 대하여 엄격한 상한을 제공하는 표기법은 무엇인가? (1) 빅오메가 (2) 빅오 (3) 빅세타 (4) 존재하지 않는다

(2)

# 12. 다음의 빅오 표기법들을 수행시간이 적게 걸리는 것부터 나열하라. O(1) O(n) O(log(n)) O(n²) O(n log(n)) O(n!) O(2ⁿ)

O(1) O(log(n)) O(n) O(n log(n)) O(n²) O(2ⁿ) O(n!)

# 13. 두 함수 30n+ 4와 n²를 여러 가지 n값으로 비교하라. 언제 30n + 4가 n²보다 작은 값을 갖는지를 구하라. 그래프를 그려보라.

n = 30.133

30n + 4 < n²

n² - 30n - 4 > 0

n > 30.133 or n < -0.133

# 14. 다음은 실제로 프로그램의 수행시간을 측정하여 도표로 나타낸 것이다. 도표로부터 이 프로그램의 시간 복잡도를 예측하여 빅오 표기법을 나타내라.

|  |  |
| --- | --- |
| 입력의 개수 n | 수행시간 (초) |
| 2 | 2 |
| 4 | 8 |
| 8 | 25 |
| 16 | 63 |
| 32 | 162 |

O(n log(n))  
O(n²)에 비해서는 수행시간이 너무 더디게 증가하고 그러기에는 O(n)이기에는 수행시간이 너무 기하급수적으로 증가한다.

# 15. 빅오 표기법의 정의를 사용하여 다음을 증명하라. 5n² + 3 = O(n²)

빅오 표기법의 정의에 따르면, 두 개의 함수 f(n) 와 g(n)가 주어졌을 때, n > n₀에 대하여 |f(n)| ≤ c |g(n)|을 만족하는 두 상수 c와 n₀가 존재하면 f(n) = O(g(n))이다. 그러므로, n >1 에 대하여 |5n² + 3| ≤ 8 |n²|이 만족되기에 5n² + 3 = O(n²)이다.

# 16. 빅오 표기법의 정의를 이용하여 6n² + 3n이 O(n)이 될 수 없음을 보여라.

빅오 표기법의 정의에 따르면, 두 개의 함수 f(n) 와 g(n)가 주어졌을 때, n > n₀에 대하여 |f(n)| ≤ c |g(n)|을 만족하는 두 상수 c와 n₀가 존재하면 f(n) = O(g(n))이다. 이 말은 즉슨, 6n² + 3n = O(n)이 되기 위해서, n > n₀에 대하여 |6n² + 3n| ≤ c |n|을 만족하는 두 상수 c와 n₀가 존재해야 한다. 그러나, 조건을 만족하는 두 상수가 없기 때문에 (값을 무엇으로 정하더라도 결국 이차함수가 일차함수보다 값이 더 커진다) 6n² + 3n이 O(n)이 될 수 없다.

# 17. 배열에 정수가 들어 있다고 가정하고 다음의 작업의 최악, 최선의 시간 복잡도를 빅오 표기법으로 말하라. (1) 배열의 n번째 숫자를 화면에 출력한다. (2) 배열 안의 숫자 중에서 최소값을 찾는다. (3) 배열의 모든 숫자를 더한다.

(1) 최악: O(1) 최선: O(1)  
배열의 경우, n번째 숫자를 인덱싱으로 바로 접근할 수 있다.

(2) 최악: O(n) 최선: O(n)  
최소값이 무엇인지 확인하기 위해 배열의 첫 요소부터 마지막 요소까지 다 접근해야 한다.

(3) 최악: O(n) 최선: O(n)  
배열의 모든 숫자를 더해야 하기에 배열의 모든 요소들을 접근해야 한다.