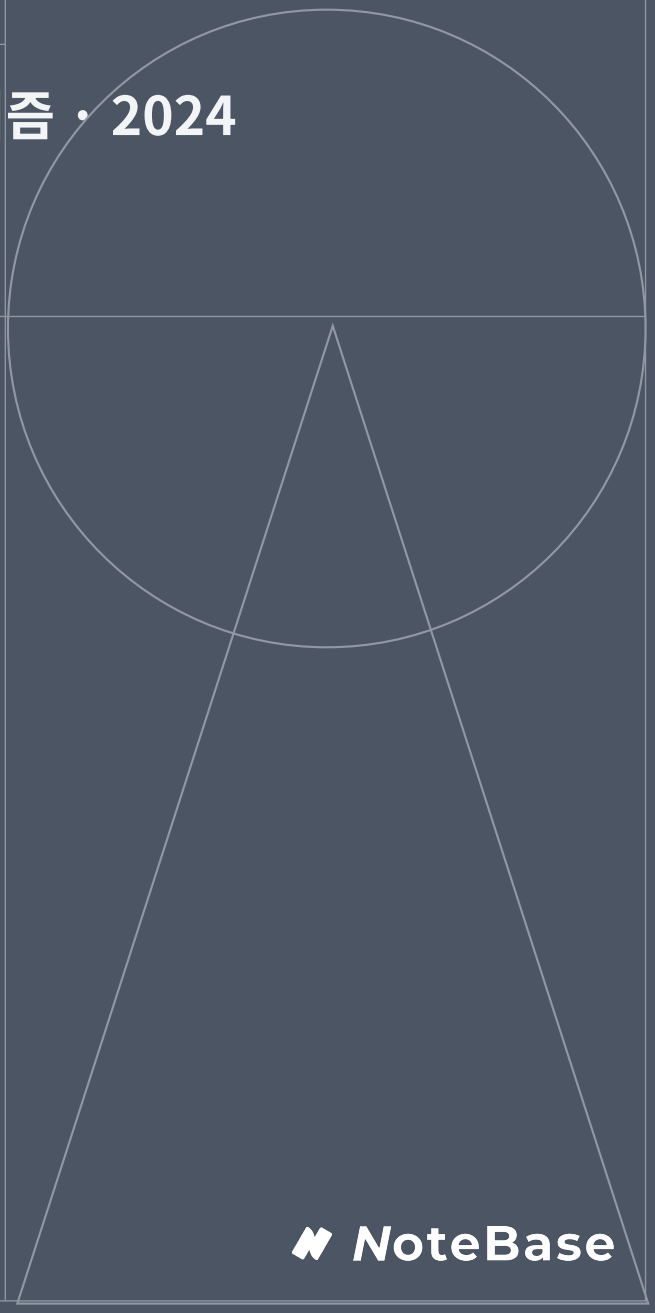
기출OX 해설집

공무원/국가직/9급 · 알고리즘 · 2024



NoteBase

기출OX 해설집

공무원/국가직/9급 · 알고리즘 · 2024

공무원/국가직/9급 · 알고리즘 · 2024

문 1. 다음은 빅오(O) 표기법에 해당하는 설명이다. (가), (나)에 들어갈 관계연산자를 바르게 연결한 것은?

어떤 양의 상수 c와 n_0이 존재하여 모든 n (가) n_0에 대하여 $f(n)$ (나) $c \cdot g(n)$을 만족하면 $f(n)$은 $O(g(n))$에 속한다.

① (가): $\geq$, (나): $\geq$

② (가): $\leq$, (나): $\leq$

③ (가): $\leq$, (나): $\geq$

④ (가): $\geq$, (나): $\leq$

정답 4번

간단한 해설

빅오는 입력 크기가 **충분히 커진 이후** 함수가 일정한 상수배의 **상한을 넘지 않음**을 나타낸다. 따라서 $n \geq n_0$인 모든 입력에서 $f(n) \leq c \cdot g(n)$이 성립해야 한다.

빈칸별 해설

(가): $\geq$. n_0는 부등식이 계속 성립하기 시작하는 기준 입력 크기이다. 빅오는 작은 입력에서의 예외를 허용하되, **기준값 이상인 모든 입력**에서 조건이 성립할 것을 요구한다. 따라서 조건은 $n \geq n_0$이다.

(나): $\leq$. $f(n) \in O(g(n))$은 $g(n)$의 양의 상수배로 $f(n)$의 점근적 상한을 정할 수 있다는 뜻이다. n에 따라 바뀌지 않는 양의 상수 c를 정하여 $f(n) \leq c \cdot g(n)$을 만족해야 한다. 즉, $f(n)$이 $g(n)$ 자체보다 작아야 하는 것이 아니라 **일정한 상수배 이하**이면 된다. 예를 들어 $f(n) = 3n + 2$, $g(n) = n$이면 $n \geq 1$에서 $3n + 2 \leq 5n$이므로 $c = 5$, $n_0 = 1$로 잡을 수 있다.

(가)에 $\geq$, (나)에 $\leq$를 넣는 조합은 ④와 일치한다.

NoteBase

문 2. 안정 정렬(stable sort) 알고리즘에 해당하지 않는 것은?

① 퀵 정렬(quick sort)

② 버블 정렬(bubble sort)

③ 삽입 정렬(insertion sort)

④ 병합 정렬(merge sort)

정답 1번

간단한 해설

안정 정렬은 **키가 같은 원소의 기존 상대적 순서**를 유지한다. 일반적인 제자리 퀵 정렬은 피벗을 기준으로 원소를 교환하는 과정에서 그 순서가 뒤바뀔 수 있어 **불안정 정렬**에 해당한다.

선지별 해설

안정성은 키가 같은 원소들이 정렬 전후에도 같은 상대적 순서를 유지하는지를 뜻한다. 예를 들어 키가 모두 2인 원소 2A와 2B가 처음에 A, B 순서였다면, 안정 정렬 후에도 A가 B보다 앞에 있어야 한다. 각 알고리즘의 통상적인 구현을 이 기준으로 비교한다.

① 퀵 정렬: 불안정 정렬이다. 피벗을 기준으로 원소를 분할하면서 멀리 떨어진 원소를 교환할 수 있어 **같은 키의 상대적 순서를 보장하지 않는다**. 예를 들어 [2A, 2B, 1]에서 마지막 원소 1을 피벗으로 삼고, 분할을 마치며 첫 원소 2A와 피벗을 교환하면 [1, 2B, 2A]가 된다. 두 원소의 키는 모두 2이지만 B가 A보다 앞서게 된다.

② 버블 정렬: 안정 정렬이다. 인접한 두 원소를 비교하여 **앞의 키가 뒤의 키보다 클 때만 교환**한다. 키가 같은 두 원소는 서로 교환하지 않으므로 기존 상대적 순서가 유지된다.

③ 삽입 정렬: 안정 정렬이다. 정렬된 부분에 새 원소를 넣을 때 **새 원소보다 키가 큰 원소만 뒤로 이동**시킨다. 키가 같은 기존 원소는 이동시키지 않고 그 뒤에 새 원소를 넣으므로 기존 상대적 순서가 유지된다.

④ 병합 정렬: 안정 정렬이다. 두 정렬된 부분을 합칠 때 **키가 같으면 왼쪽 부분의 원소를 먼저 선택**한다. 왼쪽 부분의 원소가 원래 배열에서도 앞에 있었으므로, 각 부분 내부의 순서와 함께 같은 키의 기존 상대적 순서가 유지된다.

따라서 같은 키의 기존 상대적 순서를 보장하지 않는 퀵 정렬이 안정 정렬에 해당하지 않는다.

NoteBase

문 3. 다음과 같은 총 20 kg의 분할 가능한 금속 분말을 17 kg의 무게까지 허용 가능한 배낭에 넣으려 할 때, 그리디(greedy) 알고리즘으로 얻을 수 있는 최대 이익은?

금속 분말 이름	A	B	C
무게	10 kg	6 kg	4 kg
이익	200원	300원	400원

① 600원

② 750원

③ 840원

④ 900원

정답 3번

간단한 해설

분할 가능한 분말은 **1 kg당 이익이 큰 순서**로 담는다. C와 B를 전부 담은 뒤 남은 7 kg을 A로 채우면 최대 이익은 840원이다.

계산 과정

선택 기준: 배낭의 허용 무게는 17 kg이고, 분말은 총 20 kg이다. 분말을 **분할할 수 있으므로** 일부만 담을 수 있으며, 이익도 담은 무게에 비례한다. 같은 무게를 담을 때 이익이 더 큰 분말을 우선하면 된다. 이익이 작은 분말을 담고 이익이 큰 분말을 남겨 두었다면, 같은 무게만큼 교체하여 이익을 늘릴 수 있기 때문이다.

1 kg당 이익: 표의 이익은 각 분말 전체의 이익이므로 무게로 나누어 비교한다. A는 $\frac{200}{10} = 20$원/kg, B는 $\frac{300}{6} = 50$원/kg, C는 $\frac{400}{4} = 100$원/kg이다. 따라서 **C → B → A 순서**로 담는다.

담는 무게와 이익: C 4 kg과 B 6 kg을 모두 담으면 10 kg을 사용하고, 이익은 각각 400원과 300원이다. 남은 용량은 17 − 10 = 7 kg이므로 A 10 kg 중 7 kg만 담는다. 이는 A 전체의 $\frac{7}{10}$에 해당하므로 A에서 얻는 이익은 $200 \cdot \frac{7}{10} = 140$원이다.

최대 이익: 총무게는 4 + 6 + 7 = 17 kg으로 허용 무게를 채운다. 총이익은 400 + 300 + 140 = 840원이며, ③의 840원과 일치한다.

NoteBase

문 4. 입력 크기 n에 대한 수행 횟수를 점근적 표기법으로 표현했을 때 옳지 않은 것은?

① $n^3 + 10 \rightarrow \Theta(n^3)$

② $n^2 + 100n \rightarrow O(n^3)$

③ $4n\log n + 2\log n \rightarrow \Theta(n\log n)$

④ $2n^2 + \log n + 1000 \rightarrow \Omega(n^2\log n)$

정답 4번

간단한 해설

④의 수행 횟수는 n^2에 비례하는 차수로 증가하지만, $n^2\log n$은 그보다 더 빠르게 증가한다. 따라서 $n^2\log n$의 양의 상수배 이상이라는 **점근적 하한을 만족하지 않으며**, 올바르게는 $\Theta(n^2)$로 표현할 수 있다.

선지별 해설

점근적 표기법은 입력 크기 n이 충분히 클 때의 증가율을 나타낸다. $O(g(n))$은 수행 횟수가 $g(n)$의 어떤 양의 상수배 이하라는 상한이고, $\Omega(g(n))$은 어떤 양의 상수배 이상이라는 하한이다. $\Theta(g(n))$은 두 조건을 모두 만족한다. 이때 상수는 n에 따라 달라져서는 안 된다.

① O: n≥1이면 n^3≤ n^3+10≤11n^3이다. 수행 횟수가 n^3의 양의 상수배 사이에 있으므로 $\Theta(n^3)$이다. 상수항 10은 입력이 커질 때 지배적인 n^3의 증가 차수를 바꾸지 않는다.

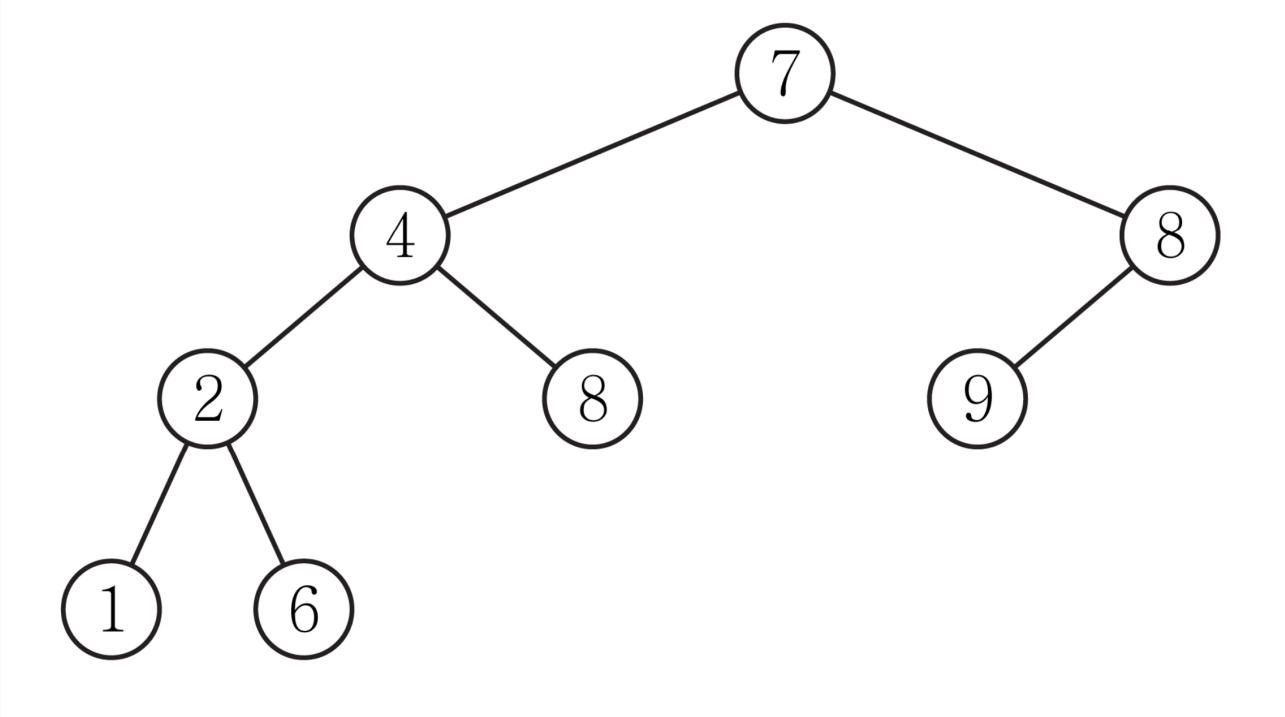
② O: n≥1이면 n^2+100n≤101n^2이고, 101n^2≤101n^3이다. 따라서 n^3의 상수배로 상한을 정할 수 있어 $O(n^3)$으로 표현할 수 있다. 더 정확한 차수는 $\Theta(n^2)$이지만, **O 표기는 가장 촘촘한 상한일 필요가 없다**.

③ O: 식을 묶으면 $(4 + \frac{2}{n})n\log n$이다. n≥2에서 계수 $4 + \frac{2}{n}$은 4 이상 5 이하이므로, 수행 횟수는 **두 양의 상수배 사이**에서 $n\log n$과 함께 증가한다. 따라서 $\Theta(n\log n)$이다.

④ X: $f(n) = 2n^2 + \log n + 1000$이라 두면 지배적인 항은 $2n^2$이므로 $f(n)$은 $\Theta(n^2)$이다. 반면 $n^2\log n$은 n^2보다 빠르게 증가하여 $\frac{f(n)}{n^2\log n} \rightarrow 0$이다. 즉, 아무리 작은 양의 상수 c를 정해도 충분히 큰 모든 n에서 $f(n) \geq c \cdot n^2\log n$을 유지할 수 없다. 따라서 **하한을 나타내는 Ω 표기가 성립하지 않는다**. $\Omega(n^2\log n)$은 $\Omega(n^2)$로 고칠 수 있으며, 상한과 하한을 함께 나타내면 $\Theta(n^2)$이다.

NoteBase

문 5. 다음 이진트리(binary tree)가 이진탐색트리(binary search tree)의 조건을 만족하려면 제거되어야 할 단말 노드(leaf node)의 최소 개수는?



① 1

② 2

③ 3

④ 4

정답 3번

간단한 해설

이진탐색트리의 대소 관계는 부모뿐 아니라 **모든 조상 노드의 조건**도 만족해야 한다. 노드 6은 조상 4의 조건을, 왼쪽의 노드 8은 루트 7의 조건을, 노드 9는 부모 8의 조건을 위반하므로 이 세 단말 노드를 제거해야 한다.

NoteBase

계산 과정

이진탐색트리는 각 노드에 대해 왼쪽 서브트리의 모든 값이 그 노드보다 작고, 오른쪽 서브트리의 모든 값이 그 노드보다 커야 한다. 단말 노드는 자식이 없는 노드이며, 이를 제거해도 다른 노드의 위치는 바뀌지 않는다.
그림의 단말 노드는 1, 6, 왼쪽의 8, 9이다.

노드 6: 부모 2의 오른쪽 자식이므로 2보다 크다는 조건은 만족한다. 그러나 **조상 4의 왼쪽 서브트리**에 있으므로 4보다도 작아야 한다. 6은 4보다 크므로 제거해야 한다.

왼쪽의 노드 8: 부모 4의 오른쪽 자식이므로 4보다 크다는 조건은 만족한다. 그러나 **루트 7의 왼쪽 서브트리**에 있으므로 7보다 작아야 한다. 8은 7보다 크므로 제거해야 한다. 루트 7의 오른쪽 자식인 다른 노드 8과 구별해야 한다.

노드 9: 루트 7의 오른쪽 서브트리에 있어 7보다 크다는 조건은 만족하지만, **부모 8의 왼쪽 자식**이므로 8보다 작아야 한다. 9는 8보다 크므로 제거해야 한다.

최소 개수 확인: 노드 1은 2, 4, 7의 왼쪽 서브트리에 있고 이들보다 모두 작으므로 유지할 수 있다. 위의 세 단말 노드를 제거하면 루트 7의 왼쪽에는 4, 2, 1이 남고 오른쪽에는 8이 남아 모든 대소 관계를 만족한다. 세 노드 중 하나라도 남기면 해당 위반이 해소되지 않으므로 최소 제거 개수는 1 + 1 + 1 = 3개이며, ③의 값과 일치한다.

NoteBase

문 6. $n \times n$ **행렬** A**와** $n \times n$ **행렬** B**에 대하여 행렬 곱셈을 하고자 한다. 다음과 같은 행렬** C**의 각 원소를 구하는 공식을 이용하여 행렬 곱셈 결과인 행렬** C**를 만들 때 시간복잡도(time complexity)는? (단,** $0 \le i, j \le n-1$**이다)**

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} B_{kj}$$

① $\Theta(n^2)$

② $\Theta(n^3)$

③ $\Theta(2^n)$

④ $\Theta(n \log n)$

정답 2번

간단한 해설

행렬 C의 원소는 모두 n^2개이며, 각 원소를 구하려면 n**개 항의 곱을 합산**해야 한다. 따라서 전체 시간복잡도는 $\Theta(n^3)$이다.

계산 과정

계산 대상: i와 j는 각각 0부터 $n-1$까지 변하므로, 행렬 C에서 구해야 할 원소는 $n \cdot n = n^2$개이다. 행렬 원소 하나의 곱셈과 덧셈은 각각 일정한 시간이 걸리는 기본 연산으로 본다.

원소 하나의 계산: C_{ij}는 행렬 A의 i번째 행과 행렬 B의 j번째 열에서 대응하는 원소들을 곱해 더한 값이다. 합의 첨자 k가 0부터 $n-1$까지 변하므로 곱셈은 n번, 그 곱들을 합치는 덧셈은 $n-1$번 필요하다. 따라서 **원소 하나를 구하는 시간**은 $\Theta(n)$이다.

전체 계산: 같은 계산을 n^2개 원소 각각에 수행하므로, 전체 곱셈 횟수는 $n^2 \cdot n = n^3$이고 덧셈 횟수는 $n^2(n-1)$이다. 전체 연산 횟수의 최고차항이 n^3이므로 시간복잡도는 $\Theta(n^3)$이며, 이는 ②에 해당한다.

NoteBase

문 7. 힙(heap)과 힙 정렬에 대한 설명으로 옳은 것만을 모두 고르면? (단, 힙은 이진트리이고, n은 원소의 개수를 나타낸다)

ㄱ. 힙은 완전이진트리(complete binary tree) 구조를 가진다.
ㄴ. 힙은 배열로 구현하기에 적합하지 않다.
ㄷ. 힙에 하나의 노드를 추가하는 데 걸리는 시간은 $O(\log n)$이다.
ㄹ. 힙 정렬의 수행시간은 $O(n \log n)$이다.

① ㄱ, ㄴ
② ㄱ, ㄹ
③ ㄴ, ㄷ
④ ㄱ, ㄷ, ㄹ

정답 4번

간단한 해설

힙은 완전이진트리이므로 노드를 층별로 왼쪽부터 빈틈없이 저장할 수 있어 **배열 구현에 적합하다**. 별도의 연결 포인터 없이도 배열 인덱스로 부모와 자식을 찾을 수 있다.

NoteBase

보기별 해설

ㄱ. O: 이진 힙은 **완전이진트리** 구조를 유지한다. 마지막 층을 제외한 모든 층이 가득 차 있고, 마지막 층의 노드는 왼쪽부터 차례로 채워져 있다.

ㄴ. X: 힙은 **배열로 구현하기에 적합하다**. 완전이진트리의 노드를 위층부터, 같은 층에서는 왼쪽부터 배열에 저장하면 중간에 빈자리가 생기지 않는다. 배열 인덱스를 1부터 시작하면 루트는 1번, 그 두 자식은 2번과 3번에 놓이는 식으로 부모·자식의 위치를 인덱스로 계산할 수 있다.

ㄷ. O: 새 노드는 완전이진트리를 유지하도록 마지막 층의 다음 빈자리에 넣는다. 이후 부모와 비교하여 힙의 대소 관계를 위반하면 위쪽으로 이동시킨다. 최대 힙은 부모가 자식 이상이고, 최소 힙은 부모가 자식 이하인 관계를 유지한다. 이동은 **삽입 위치에서 루트까지의 한 경로**에서만 일어나며, 원소가 n개인 완전이진트리의 높이가 $O(\log n)$이므로 삽입의 최악 시간복잡도는 $O(\log n)$이다. 관계가 일찍 충족되면 루트까지 이동하지 않고 끝날 수 있다.

ㄹ. O: 오름차순 힙 정렬에서는 최대 힙을 만든 뒤, 루트의 최댓값을 힙의 마지막 원소와 교환하여 뒤쪽에 확정한다. 힙의 범위를 하나 줄이고 새 루트를 아래로 이동시켜 **남은 원소의 힙 관계를 복구**한다. 최초 힙 구성은 $O(n)$에 가능하며, 이후 최대 $n-1$번의 추출·복구에서 매번 $O(\log n)$ 시간이 든다. 따라서 전체 수행시간은 $O(n \log n)$이다.

옳은 보기는 ㄱ, ㄷ, ㄹ이며, 이 조합은 ④에 해당한다.

NoteBase

문 8. 입력 크기가 n인 정렬 알고리즘에 대한 시간복잡도가 바르게 연결되지 않은 것은?

① 정렬 알고리즘: 삽입 정렬(insertion sort), 최악의 경우: $O(n^2)$, 평균적인 경우: $O(n^2)$, 최선의 경우: $O(n)$

② 정렬 알고리즘: 선택 정렬(selection sort), 최악의 경우: $O(n^2)$, 평균적인 경우: $O(n^2)$, 최선의 경우: $O(n\log n)$

③ 정렬 알고리즘: 퀵 정렬, 최악의 경우: $O(n^2)$, 평균적인 경우: $O(n\log n)$, 최선의 경우: $O(n\log n)$

④ 정렬 알고리즘: 병합 정렬, 최악의 경우: $O(n\log n)$, 평균적인 경우: $O(n\log n)$, 최선의 경우: $O(n\log n)$

정답 2번

간단한 해설

선택 정렬은 **이미 정렬된 입력에서도 남은 원소 전체를 비교**하여 최솟값을 찾는다. 따라서 최선의 경우에도 시간복잡도는 $O(n^2)$이며, ②의 $O(n\log n)$ 연결이 잘못되었다.

선지별 해설

① O: 삽입 정렬은 앞부분을 정렬된 상태로 유지하면서 다음 원소를 알맞은 위치에 끼워 넣는다. **이미 오름차순으로 정렬되어 있으면** 각 원소가 바로 앞 원소와 한 번 비교한 뒤 제자리에 머무르므로 최선은 $O(n)$이다. 반대로 역순이면 삽입할 때마다 앞의 원소들을 모두 뒤로 옮겨야 하므로 최악은 $O(n^2)$이다. 일반적인 무작위 입력에서도 원소당 이동 횟수가 앞부분의 길이에 비례하므로 평균은 $O(n^2)$이다.

② X: 선택 정렬은 아직 정렬되지 않은 부분을 끝까지 살펴 최솟값을 찾고, 그 값을 해당 부분의 맨 앞에 놓는다. 비교 횟수는 입력 순서와 관계없이 처음에 n − 1회, 다음에 n − 2회, 마지막에 1회로, 총 $\frac{n(n-1)}{2}$회이다. 이미 정렬되어 있어 교환을 줄일 수 있어도 **최솟값을 찾는 비교는 줄어들지 않는다**. 따라서 최악·평균·최선 모두 $O(n^2)$이며, 최선의 경우를 $O(n\log n)$에서 $O(n^2)$으로 고쳐야 한다.

③ O: 퀵 정렬은 피벗을 기준으로 원소를 나눈 뒤 각 부분을 재귀적으로 정렬한다. 한 번 분할할 때에는 해당 부분의 원소 수에 비례하는 시간이 든다. **분할이 균형을 이루면** 재귀 깊이가 $O(\log n)$이고 각 깊이에서 처리하는 원소 수의 합이 $O(n)$이므로 최선은 $O(n\log n)$이다. 무작위 입력에 대한 평균도 $O(n\log n)$이다. 반면 피벗이 매번 최솟값이나 최댓값이 되어 한쪽에 원소가 몰리면 처리할 크기가 하나씩만 줄어들어 최악은 $O(n^2)$이 된다.

④ O: 표준 병합 정렬은 입력을 절반씩 나누어 정렬한 뒤, 두 정렬된 부분을 하나로 합친다. **입력 순서와 관계없이 절반씩 분할**하므로 재귀 깊이는 $O(\log n)$이다. 각 깊이에서 병합에 드는 전체 작업량은 $O(n)$이므로 최악·평균·최선 모두 $O(n\log n)$이다. 이는 이미 정렬되었는지 검사하여 병합을 생략하는 등의 별도 최적화를 적용하지 않은 표준 구현을 기준으로 한다.

NoteBase

문 9. 다음은 문자들에 대한 빈도수를 나타낸다. 이 문자들에 대한 허프만 코드(Huffman code)를 생성할 경우에 가장 긴 코드를 가진 문자의 비트수는?

문자	a	b	c	d	e	f
빈도수	1	9	18	48	11	12

① 2

② 3

③ 4

④ 5

정답 3번

간단한 해설

허프만 트리에서 **코드의 비트수는 루트에서 해당 문자까지의 간선 수**이다. 빈도수가 가장 작은 두 노드를 차례로 합치면 a와 b가 가장 깊은 곳에 놓이며, 두 문자의 코드 길이는 4비트이다.

NoteBase

계산 과정

허프만 코드는 **현재 남은 노드 중 빈도수가 가장 작은 두 노드**를 골라 하나의 부모 노드로 합치는 과정을 반복하여 만든다. 부모의 빈도수는 두 자식의 빈도수 합이다. 각 간선에 0 또는 1을 부여하므로, 루트에서 문자까지 지나가는 간선 하나가 코드의 1비트에 해당한다. 표의 빈도수는 a 1, b 9, c 18, d 48, e 11, f 12이다.

1단계: a와 b를 합쳐 1 + 9 = 10인 노드를 만든다. 남은 빈도수는 10, 11, 12, 18, 48이다.

2단계: 새 노드 10과 e의 11을 합쳐 10 + 11 = 21인 노드를 만든다. 이 노드에는 a, b, e가 포함되며, 남은 빈도수는 12, 18, 21, 48이다.

3단계: f의 12와 c의 18을 합쳐 12 + 18 = 30인 노드를 만든다. 남은 빈도수는 21, 30, 48이다.

4단계: 노드 21과 30을 합쳐 21 + 30 = 51인 노드를 만든다. 이제 d의 48과 새 노드 51만 남는다.

5단계: 48과 51을 합쳐 48 + 51 = 99인 루트 노드를 만든다.

코드 길이: 루트에서 a까지의 경로는 99 → 51 → 21 → 10 → a로, **간선이 4개**이다. b도 같은 깊이에 있으므로 4비트이고, c·e·f는 각각 3비트, d는 1비트이다. 따라서 가장 긴 코드의 길이는 4비트이며, ③의 값에 해당한다.

NoteBase

문 10. 다음 파이썬 코드의 시간복잡도는? (단, n은 1보다 큰 정수이다)

```
def abc(n) :
    if n == 1 : return 1
    else : return n * abc(n - 1)
```

① $O(1)$

② $O(\log n)$

③ $O(n)$

④ $\Theta(2^n)$

정답 3번

간단한 해설

각 호출은 입력을 1 줄여 **재귀 호출을 한 번만** 수행한다. 입력이 n에서 1이 될 때까지 총 n번 호출되므로 시간복잡도는 $O(n)$이다.

계산 과정

호출당 작업: n은 1보다 큰 정수이다. 기본 산술 연산을 상수 시간으로 보는 일반적인 시간복잡도 분석에서, n이 1이면 조건 검사 후 1을 반환하고 종료한다. n이 1보다 크면 abc(n - 1)을 한 번 호출하고, 그 반환값에 n을 곱한다. **n을 곱하는 것은 재귀 호출을 n번 한다는 뜻이 아니다.**

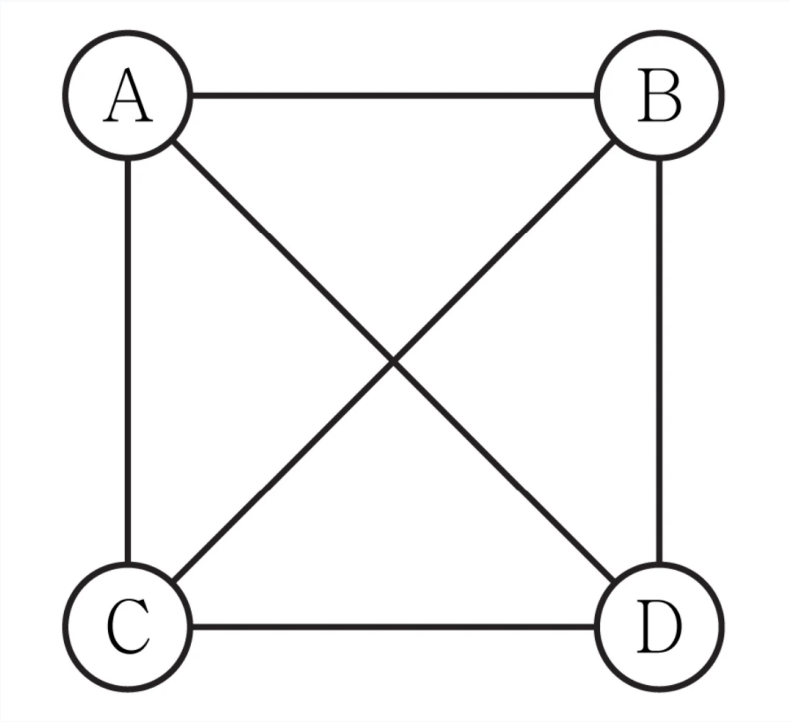
점화식: 입력이 n일 때의 실행 시간을 $T(n)$이라 하면, 재귀 호출 외에는 조건 검사·뺄셈·곱셈·반환만 수행하므로 $T(n) = T(n-1) + \Theta(1)$이고 $T(1) = \Theta(1)$이다. 여기서 $\Theta(1)$은 각 호출에서 추가되는 작업량이 입력 크기와 무관하게 일정하다는 뜻이다.

실행 횟수: 호출 인수는 n, n - 1, ⋯, 1로 줄어든다. 따라서 **종료 호출을 포함해 총 n번** 호출되며, 곱셈은 종료 호출을 제외한 n - 1번 수행된다. 예를 들어 n = 4이면 abc(4) → abc(3) → abc(2) → abc(1)로 호출되므로 호출은 4번, 곱셈은 3번이다.

전체 시간: 일정한 작업을 총 n번 수행하므로 $T(n) = \Theta(n)$이다. 즉 실행 시간은 입력 n에 비례하여 증가하며, 선택지에서는 ③의 $O(n)$에 해당한다.

NoteBase

문 11. 다음 그래프에서 생성이 가능한 신장 트리(spanning tree)의 최대 개수는?



① 4

② 8

③ 16

④ 32

정답 3번

간단한 해설

네 정점을 잇는 신장 트리는 간선 3개로 이루어진다. 가능한 간선 조합 20개에서 **삼각형을 이루어 한 정점을 고립시키는 4개**를 제외하면 16개이다.

NoteBase

계산 과정

그래프의 정점은 A, B, C, D의 4개이고, 간선은 AB, AC, AD, BC, BD, CD의 6개이다. 대각선의 교차점은 별도의 정점이 아니다. 신장 트리는 원래 그래프의 모든 정점을 포함하면서 **연결되어 있고 사이클이 없는** 부분 그래프이므로, 정점이 4개이면 간선은 4 − 1 = 3개여야 한다.

간선 조합 세기: 서로 다른 간선 6개 중 3개를 순서 없이 고르는 경우의 수는 $\binom{6}{3} = \frac{6 \cdot 5 \cdot 4}{3 \cdot 2 \cdot 1} = 20$개이다. 이 20개에는 신장 트리가 아닌 조합도 포함되어 있다.

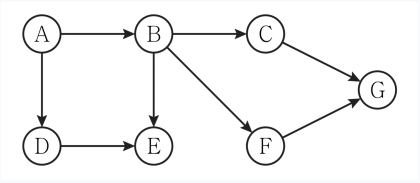
제외할 조합: 간선이 3개이므로 사이클이 생긴다면 삼각형이어야 한다. 삼각형 ABC, ABD, ACD, BCD를 이루는 4개 조합은 각각 나머지 한 정점을 고립시킨다. 반대로 간선 3개로 네 정점이 연결되지 않는 경우도 이 네 삼각형뿐이다. 정점이 두 개씩 나뉘면 간선을 최대 2개만 가질 수 있기 때문이다. 따라서 **삼각형 4개 조합만 제외**하면 나머지는 모두 신장 트리이다.

신장 트리의 개수: 20 − 4 = 16개이며, 이는 ③의 16에 해당한다.

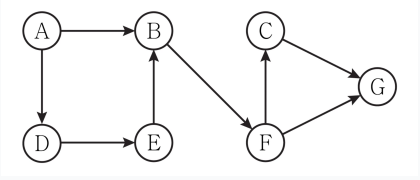
NoteBase

문 12. 위상 정렬(topological sort)을 적용할 때 모든 정점을 포함하는 결과가 생성될 수 없는 그래프는?

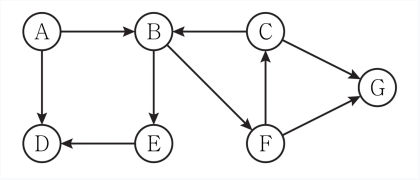
①



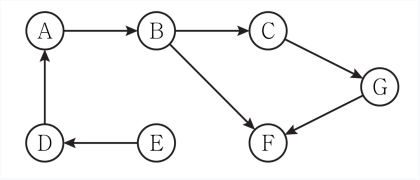
②



③



④



정답 3번

간단한 해설

③에는 **B → F → C → B의 방향 사이클**이 있다. 위상 정렬은 간선의 출발 정점을 도착 정점보다 앞에 두어야 하므로, 이 사이클의 정점들을 모두 배치할 수 없다.

공통 해설

위상 정렬은 방향 간선의 출발 정점이 도착 정점보다 앞에 오도록 모든 정점을 일렬로 배치하는 것이다. 예를 들어 B → F라는 간선이 있으면 B를 F보다 먼저 놓아야 한다. 모든 정점을 포함하는 위상 정렬이 존재할 필요충분조건은 **방향 사이클이 없는 것**이다. 이러한 그래프를 방향 비순환 그래프(DAG)라고 한다.

③의 화살표를 따라가면 B → F, F → C, C → B가 이어진다. B를 F보다 앞에, F를 C보다 앞에, 다시 C를 B보다 앞에 놓아야 하므로 **B가 자기 자신보다 앞에 와야 하는 모순**이 생긴다. 다른 정점부터 처리하더라도 이 사이클의 선후 관계를 만족시킬 수 없어 모든 정점을 포함하는 결과는 만들 수 없다.

나머지 그래프는 같은 선후 관계 조건을 만족하도록 모든 정점을 배치할 수 있다. 가능한 순서의 예는 ①에서 A → B → D → C → E → F → G, ②에서 A → D → E → B → F → C → G, ④에서 E → D → A → B → C → G → F이다. 각각의 순서에서 모든 간선은 앞에 놓인 정점에서 뒤에 놓인 정점으로 향하므로 위상 정렬 조건을 충족한다.

NoteBase

문 13. 다음 설명에 해당하는 알고리즘은?

○ 모든 쌍 최단 거리(all pairs shortest path)를 구하는 알고리즘이다.
○ 음수의 가중치를 가진 간선(edge)이 있어도 수행될 수 있다.
○ 동적 계획법(dynamic programming)의 원리를 이용한다.
○ 시간복잡도는 $O(n^3)$이다. (단, n은 정점의 수이다)

① 프림(Prim) 알고리즘
② 플로이드-워셜(Floyd-Warshall) 알고리즘
③ 다익스트라(Dijkstra) 알고리즘
④ KMP(Knuth-Morris-Pratt) 알고리즘

정답 2번

간단한 해설

플로이드-워셜 알고리즘은 **모든 정점 쌍의 최단 거리**를 동적 계획법으로 구한다. 음수 간선도 처리할 수 있으며, 각 정점을 경유하는 경우를 차례로 검토하므로 시간복잡도는 $O(n^3)$이다.

공통 해설

플로이드-워셜 알고리즘은 **모든 정점 쌍 사이의 최단 거리**를 구하는 동적 계획법이다. 각 출발 정점과 도착 정점 사이의 거리를 행렬에 저장하고, 중간 정점으로 사용할 수 있는 정점의 범위를 하나씩 넓혀 간다.

새로운 중간 정점을 허용할 때는 기존 거리와 그 정점을 거쳐 가는 두 구간의 거리 합을 비교하여 더 작은 값으로 갱신한다. 예를 들어 기존 거리가 8이고 새 중간 정점을 거치는 두 구간의 거리가 각각 3과 2라면, 거리를 3 + 2 = 5로 줄인다. 이미 구한 부분 경로의 최단 거리를 재사용한다는 점에서 동적 계획법에 해당한다.

간선의 가중치가 음수여도 이러한 갱신을 수행할 수 있다. 다만 모든 정점 쌍의 최단 거리를 유한한 값 또는 도달 불가능으로 확정하려면 **음수 사이클이 없어야 한다**. 음수 사이클은 한 바퀴의 가중치 합이 음수인 순환 경로이며, 이를 반복해서 지날 수 있는 경로는 거리를 계속 줄일 수 있어 유한한 최단 거리가 정해지지 않는다.

정점 수를 n이라 하면, 중간 정점을 고르는 n번의 단계마다 모든 출발·도착 정점 쌍 n^2개를 검사한다. 따라서 시간복잡도는 $O(n^3)$이며, 지문의 네 가지 특징이 모두 플로이드-워셜 알고리즘의 작동 원리와 일치한다.

NoteBase

문 14. 다음 C언어로 작성된 함수의 입력값으로 13이 입력될 경우 출력되는 결과는?

```
void foo(int n) {
    if(n != 0){
        foo(n/2);
        printf("%d", n%2);
    }
}
```

① 1010
② 1011
③ 1100
④ 1101

정답 4번

간단한 해설

2로 나눈 몫을 재귀 호출에 넘기고, **재귀 호출이 끝난 뒤 나머지를 출력**한다. 따라서 호출할 때 구한 나머지의 역순으로 1, 1, 0, 1이 출력되어 결과는 1101이다.

NoteBase

계산 과정

연산의 의미: n은 정수형이므로 n / 2는 2로 나눈 정수 몫이고, n % 2는 나머지이다. 이 함수는 몫을 인자로 재귀 호출한 뒤 현재 호출의 나머지를 출력한다. **각 호출의 n은 따로 유지**되므로 재귀 호출에서 돌아와도 현재 호출의 n으로 나머지를 구한다.

재귀 호출: 입력값 13에서 시작하면 몫에 따라 foo(13) → foo(6) → foo(3) → foo(1) → foo(0) 순서로 호출된다. 아직 앞선 호출의 printf는 실행되지 않는다.

종료 조건: foo(0)에서는 n != 0이 거짓이므로 재귀 호출과 출력을 모두 하지 않고 돌아간다. 따라서 종료 조건의 0 자체는 출력되지 않는다.

복귀와 출력: **가장 안쪽 호출부터** 실행을 마치므로 n이 1, 3, 6, 13인 호출 순서로 printf가 실행된다. 각각 2로 나눈 나머지는 1, 1, 0, 1이다. 특히 n이 6인 호출도 n != 0을 만족하므로 나머지 0을 출력한다.

출력 결과: printf의 형식 문자열에는 공백이나 줄바꿈이 없어 네 숫자가 이어진 1101이 출력된다. 이는 ④의 1101과 일치한다.

NoteBase

문 15. 텍스트 문자열 01001에 대하여 패턴 문자열 001을 찾기 위해 브루트-포스(brute-force) 문자열 검색 알고리즘을 사용할 경우, 문자를 비교한 총 횟수는? (단, 패턴 문자열이 한번 검색되면 더 이상 검색하지 않는다)

① 1

② 3

③ 5

④ 6

정답 4번

간단한 해설

브루트-포스 검색은 문자가 다르면 **시작 위치를 한 칸 옮기고 패턴의 첫 문자부터 다시 비교**한다. 각 시작 위치에서 2회, 1회, 3회 비교하므로 총 6회이며, 패턴을 찾은 뒤에는 종료한다.

계산 과정

텍스트 '01001'의 위치를 왼쪽부터 1~5번째로 센다. 패턴 '001'을 텍스트의 각 시작 위치에 맞추어 왼쪽부터 비교하고, 불일치하면 **시작 위치를 한 칸 옮겨 패턴의 첫 문자부터** 다시 비교한다. 일치 여부를 확인한 문자 쌍마다 비교 횟수를 1회씩 센다.

1번째 위치에서 시작: 텍스트의 1번째 문자 0과 패턴의 첫 문자 0은 일치한다. 이어 텍스트의 2번째 문자 1과 패턴의 두 번째 문자 0이 다르므로 이 시도에서는 2회 비교한다.

2번째 위치에서 시작: 텍스트의 2번째 문자 1과 패턴의 첫 문자 0이 곧바로 다르므로 1회 비교한다.

3번째 위치에서 시작: 텍스트의 3~5번째 문자는 '001'이다. 패턴과 차례로 비교하면 0과 0, 0과 0, 1과 1이 모두 일치하므로 3회 비교한다. **패턴을 한 번 찾으면 종료**한다는 조건에 따라 여기서 검색을 마친다.

총 문자 비교 횟수는 2 + 1 + 3 = 6회로, ④의 6에 해당한다.

NoteBase

문 16. 다음 중 그리디 알고리즘에 해당하는 것만을 모두 고르면?

ㄱ. 라빈-카프(Rabin-Karp) 알고리즘
ㄴ. 병합 정렬 알고리즘
ㄷ. 다익스트라 알고리즘
ㄹ. 플로이드-워셜 알고리즘

① ㄷ
② ㄱ, ㄴ
③ ㄱ, ㄷ
④ ㄱ, ㄷ, ㄹ

정답 1번

간단한 해설

다익스트라 알고리즘은 간선 가중치가 음수가 아닐 때, 미확정 정점 중 **출발점으로부터의 잠정 거리가 가장 짧은 정점**을 선택해 최단 거리를 확정한다. 이렇게 현재의 최선인 선택을 차례로 확정하는 방식이 그리디 전략이다.

NoteBase

보기별 해설

그리디 알고리즘은 매 단계에서 현재 기준으로 가장 유리한 선택을 확정하며 해를 구성하는 방식이다. 각 알고리즘이 해를 만드는 원리를 기준으로 분류한다.

ㄱ. 라빈-카프: **해시를 이용한 문자열 검색** 알고리즘이다. 패턴의 해시값과 본문에서 패턴과 길이가 같은 부분 문자열의 해시값을 비교하고, 검색 위치를 옮길 때 해시값을 갱신한다. 해시값이 같으면 실제 문자열도 비교하여 일치 여부를 확인한다. 현재의 최선인 선택을 확정해 최적해를 구성하는 그리디 방식에 해당하지 않는다.

ㄴ. 병합 정렬: **분할 정복** 알고리즘이다. 배열을 작은 부분 배열로 나누고 각각 정렬한 뒤, 정렬된 부분 배열들을 병합하여 전체를 정렬한다. 병합할 때 작은 원소부터 선택하더라도, 전체 알고리즘의 설계 원리는 문제의 분할과 부분 해의 결합이다.

ㄷ. 다익스트라: **그리디** 알고리즘이다. 간선 가중치가 모두 0 이상인 그래프에서 하나의 출발점으로부터 각 정점까지의 최단 거리를 구한다. 아직 최단 거리가 확정되지 않은 정점 중 잠정 거리가 가장 짧은 정점을 선택해 그 거리를 확정하고, 인접 정점의 잠정 거리를 갱신한다. 가중치가 음수가 아니므로 나중에 다른 미확정 정점을 거치는 경로로도 방금 확정한 거리를 더 줄일 수 없어 이 선택이 성립한다.

ㄹ. 플로이드-워셜: **동적 계획법** 알고리즘이다. 중간 정점으로 허용하는 정점 집합을 차례로 늘리면서, 기존 경로와 새로 허용한 정점을 거치는 경로 중 더 짧은 경로로 각 정점 쌍의 거리를 갱신한다. 이전 단계에서 구한 최단 거리를 이용해 다음 단계의 최단 거리를 구하는 구조이며, 한 정점을 탐욕적으로 선택해 최단 거리를 확정하는 다익스트라와 구별된다.

따라서 그리디 알고리즘에 해당하는 보기는 ㄷ뿐이다.

NoteBase

문 17. 퀵 정렬 시 시간복잡도가 최악의 경우가 되는 것으로 가장 적절한 것은?

① 피벗(pivot)을 최대값으로 정한다.

② 피벗을 랜덤(random)하게 정한다.

③ 피벗을 중간값(median)으로 정한다.

④ 피벗을 파티션(partition)의 중간에 위치한 값으로 정한다.

정답 1번

간단한 해설

매번 **최대값을 피벗**으로 고르면 나머지 원소가 모두 한쪽에 모여 분할이 극단적으로 불균형해진다. 처리할 구간의 크기가 한 번에 하나씩만 줄어들어 시간복잡도가 $O(n^2)$이 된다.

NoteBase

선지별 해설

퀵 정렬은 피벗을 기준으로 원소를 나눈 뒤 각 부분 구간을 다시 정렬한다. 시간복잡도를 가르는 핵심은 재귀적으로 처리할 두 구간의 크기가 얼마나 균형을 이루는가이다. 다음은 서로 다른 원소를 정렬하는 경우의 비교이다.

① 최대값: 서로 다른 원소로 이루어진 구간에서 최대값을 피벗으로 고르면, 피벗보다 큰 원소는 없고 나머지는 모두 작다. 따라서 원소가 n개인 구간은 피벗을 제외하고 0개와 $n-1$개로 나뉜다. 매번 이 방식으로 고르면 **한쪽 구간만 계속 남는 분할**이 반복된다. 각 분할에서 남은 원소들을 살펴야 하므로 전체 작업량은 이차적으로 증가하여 $O(n^2)$이 된다.

② 랜덤 선택: 피벗의 크기 순위를 무작위로 선택하므로 분할이 항상 한쪽으로 치우치도록 정해져 있지 않다. 기대 시간복잡도는 $O(n\log n)$이다. 다만 **불운한 선택이 반복되면** 최대값이나 최솟값만 연속해서 골라 $O(n^2)$의 최악 상황이 생길 수는 있다.

③ 중간값(median): 원소를 값의 크기순으로 보았을 때 중앙에 해당하는 값을 뜻한다. 각 구간의 실제 중간값을 피벗으로 고르면 작은 원소와 큰 원소의 수가 거의 같아져 **균형 잡힌 분할**이 이루어진다. 따라서 최대값을 반복해서 고를 때와 같은 극단적인 불균형을 일으키지 않는다.

④ 중간 위치의 값: 배열에서 가운데 위치에 저장된 원소를 고르는 방식이다. **위치가 가운데라는 사실은 값의 크기 순위와 무관**하므로 실제 중간값을 고르는 것과 다르다. 입력 배열에 따라 분할이 균형을 이룰 수도 있고, 그 원소가 최대값이나 최솟값이어서 불균형해질 수도 있다.

네 방식 중 매 분할에서 최대값을 고르는 ①이 극단적인 불균형을 반복해서 일으키는 방식이므로, 최악의 경우가 되는 것으로 가장 적절하다.

NoteBase

문 18. 분할 정복(divide-conquer) 문제 해결 기법을 이용하여 토너먼트 방식으로 64개의 축구팀 중 1등을 결정하기 위해 치르게 되는 최소의 경기 횟수는? (단, 모든 경기는 팀별 1:1 방식으로 진행되며 무승부 및 기권은 없다)

① 61

② 62

③ 63

④ 64

정답 3번

간단한 해설

한 경기에서 **한 팀씩 탈락**하므로, 우승팀 하나를 남기려면 나머지 63팀이 탈락해야 한다. 따라서 필요한 최소 경기 수는 64 − 1 = 63경기이다.

계산 과정

모든 경기는 두 팀이 맞붙으며 무승부와 기권이 없으므로, 한 경기마다 승자와 패자가 하나씩 결정된다. 패자가 탈락하는 토너먼트에서 우승팀 하나를 남기려면 **64팀 중 63팀을 탈락**시켜야 하므로 최소 63경기가 필요하다.

분할 정복 방식에서는 팀들을 두 집단으로 나누어 각 집단의 우승팀을 결정한 뒤, 두 우승팀의 경기로 전체 우승팀을 결정한다. 이를 반복하는 토너먼트에서 64팀이 32팀으로 줄어드는 첫 라운드는 32경기이고, 이후 라운드의 경기 수는 차례로 16, 8, 4, 2, 1경기이다.

전체 경기 수는 32 + 16 + 8 + 4 + 2 + 1 = 63경기이다. 실제로 63경기로 우승팀을 결정할 수 있으므로 최소 경기 수는 63경기이며, ③의 값과 일치한다.

NoteBase

문 19. 다음 설명의 (가)에 들어갈 용어로 옳은 것은? (단, P ≠ NP이다)

어떤 문제 A가 다음을 모두 만족하면 (가)이다.
◯ 문제 A는 NP에 속한다.
◯ 모든 NP 문제들은 다항식 시간에 문제 A로 변환할 수 있다.

① P 문제
② NP 문제
③ NP-하드(NP-hard) 문제
④ NP-완전(NP-complete) 문제

정답 4번

간단한 해설

모든 NP 문제를 A로 다항식 시간에 환원할 수 있다는 것은 A가 **NP-하드**라는 뜻이다. 여기에 **A가 NP에 속한다**는 조건까지 결합하면 NP-완전 문제가 된다.

공통 해설

NP-완전 문제는 **NP에 속하면서 NP-하드인 문제**이다. NP에 속한다는 것은 '예'인 입력에 대해 그 사실을 입증하는 인증서를 다항식 시간에 검증할 수 있다는 뜻이다. 다항식 시간은 입력 크기의 다항식으로 제한되는 시간을 말한다.

지문의 '모든 NP 문제들은 다항식 시간에 문제 A로 변환할 수 있다'는 NP-하드의 조건이다. 각 NP 문제의 입력을 답이 보존되도록 A의 입력으로 바꿀 수 있다는 의미이며, 환원 방향은 **각 NP 문제에서 A로** 향한다. 따라서 A를 푸는 방법이 있으면 변환을 거쳐 다른 NP 문제도 풀 수 있다. NP라는 조건만으로는 이러한 환원 가능성까지 보장하지 않으며, NP-하드라는 조건만으로는 A가 NP에 속한다고 보장하지 않는다. 지문은 두 조건을 모두 제시하므로 (가)는 NP-완전이다.

주어진 P ≠ NP 조건에서는 A가 P에 속할 수 없다. A를 다항식 시간에 풀 수 있다면 모든 NP 문제를 A로 변환한 뒤 풀어 모든 NP 문제를 다항식 시간에 해결할 수 있기 때문이다. 이는 P = NP를 뜻하므로 문제의 가정에 어긋난다.

NoteBase

문 20. 다음 의사코드(pseudo-code)로 표현된 알고리즘의 수행시간을 $T(n)$으로 나타낼 때, $T(n)$의 계산식과 $T(n)$의 점근적 복잡도로 옳은 것은? (단, n은 1보다 큰 정수이고 c는 양의 상수이다)

```
algo(n)
{
    if (n ≤ 1) return 0;
    return 1 + algo(n / 2);
}
```

① $T(n) = T(\frac{n}{2}) + c$, $\Theta(n)$

② $T(n) = T(\frac{n}{2}) + c$, $\Theta(\log_2 n)$

③ $T(n) = 2T(\frac{n}{2}) + c$, $\Theta(n)$

④ $T(n) = 2T(\frac{n}{2}) + n$, $\Theta(n\log_2 n)$

정답 2번

간단한 해설

입력 크기를 절반으로 줄인 **재귀 호출을 한 번만** 수행하고, 그 밖의 작업에는 상수 시간이 걸린다. 따라서 $T(n) = T(\frac{n}{2}) + c$이며, 크기가 1 이하가 될 때까지의 호출 단계 수가 로그에 비례하므로 복잡도는 $\Theta(\log_2 n)$이다.

NoteBase

계산 과정

점화식 구성: $T(n)$은 입력 크기가 n일 때의 수행시간이다. n이 1 이하이면 즉시 반환하므로 종료 단계의 수행시간은 상수이다. n이 1보다 크면 algo(n / 2)를 **한 번만 호출**한다. 조건 검사, 나눗셈, 덧셈, 반환에 드는 시간을 양의 상수 c로 묶으면 $T(n) = T(\frac{n}{2}) + c$이다. 반환값에 1을 더하는 것은 재귀 호출을 추가하는 일이 아니며, 입력 크기에 비례하는 반복 작업도 없다.

점화식 전개: 두 단계까지 대입하면 $T(n) = T(\frac{n}{4}) + 2c$가 된다. 이를 k단계 반복하면 $T(n) = T(\frac{n}{2^k}) + kc$이다. 여기서 kc는 **각 단계의 상수 시간 작업을 누적한 값**이다.

종료 단계와 복잡도: 계산을 간단히 보이기 위해 n이 2의 거듭제곱인 경우를 보면, $\frac{n}{2^k} = 1$에서 종료 단계에 도달하므로 $k = \log_2 n$이다. 따라서 $T(n) = T(1) + c\log_2 n$이며, $T(1)$과 c는 상수이므로 $T(n) = \Theta(\log_2 n)$이다. n이 2의 거듭제곱이 아니어도 크기를 절반으로 줄이는 횟수는 로그에 비례하므로 같은 점근적 복잡도를 갖는다.

도출한 점화식 $T(n) = T(\frac{n}{2}) + c$와 복잡도 $\Theta(\log_2 n)$의 짝은 ②에 해당한다.

NoteBase