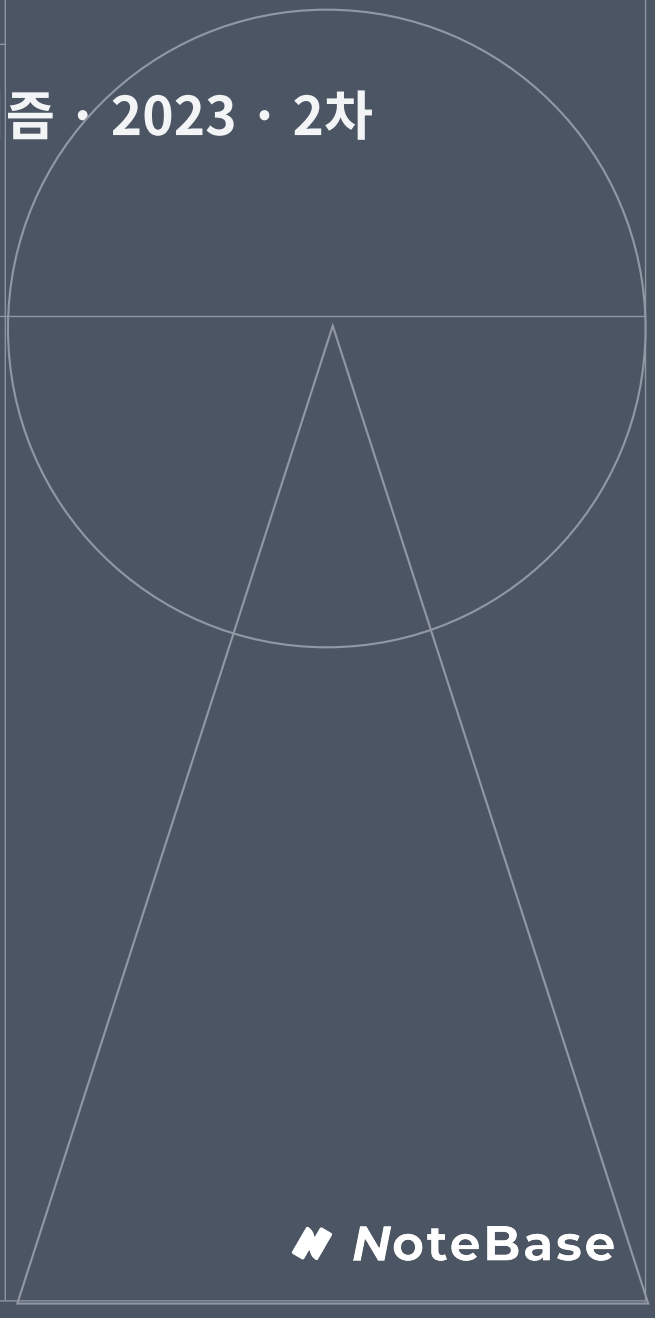
기출OX 해설집

공무원/국가직/7급 · 알고리즘 · 2023 · 2차



NoteBase

기출OX 해설집

공무원/국가직/7급 · 알고리즘 · 2023 · 2차

공무원/국가직/7급 · 알고리즘 · 2023 · 2차

문 1. 컴퓨터 알고리즘(algorithm)의 특성에 대한 설명으로 옳지 않은 것은?

① 알고리즘은 1개 이상의 결과(output)를 생성해야 한다.

② 알고리즘에는 반드시 입력(input)이 1개 이상 존재해야 한다.

③ 알고리즘은 한정된 횟수의 단계 실행 후 반드시 종료해야 한다.

④ 알고리즘의 각 명령은 모호하지 않고 명확해야 한다.

정답 2번

출제 포인트

이 문제는 알고리즘의 기본 특성인 입력, 출력, 유한성, 명확성을 정확히 구별하는지를 묻는다.

핵심 개념

입력은 알고리즘이 처리할 초기 자료로서 0개 이상일 수 있다. 출력은 알고리즘 수행 결과로서 1개 이상이어야 한다. 유한성은 한정된 단계 안에 종료되는 성질이다. 명확성은 각 명령의 의미가 모호하지 않고 분명한 성질이다.

선지별 해설

1번 O: 알고리즘은 문제를 해결한 결과를 내야 하므로 출력은 1개 이상 존재해야 한다.
2번 X: 틀린 부분은 "반드시 입력(input)이 1개 이상"이라는 표현이다. 알고리즘은 입력이 없을 수도 있으므로 올바르게는 "입력은 0개 이상 존재할 수 있다"이다.
3번 O: 알고리즘은 무한히 반복되면 문제 해결 절차가 될 수 없으므로 유한한 단계 후 반드시 종료되어야 한다.
4번 O: 알고리즘의 각 명령은 실행 주체가 해석에 혼동이 없도록 명확하고 모호하지 않아야 한다.

오답 포인트

입력과 출력을 모두 "1개 이상"으로 묶어 외우면 2번에 걸린다. 알고리즘 특성에서 입력은 없어도 되지만 출력은 반드시 있어야 한다는 점이 함정이다.

다시 나오면 이렇게 판단

알고리즘 특성 문제는 입력, 출력, 유한성, 명확성을 하나씩 분리해서 확인하면 된다. 특히 입력은 0개 이상, 출력은 1개 이상이라는 기준으로 먼저 판단한다.

NoteBase

암기 포인트

입력은 없어도 되고 출력은 반드시 나온다.

NoteBase

문 2. 다음 피보나치 수열의 n번째 항을 계산하는 알고리즘에 대한 설명으로 옳지 않은 것은? (단, $n \geq 0$이다)

```
int fib(n) {
   if (n == 0)
      return 0;
   if (n == 1)
      return 1;
   return fib(n - 1) + fib(n - 2);
}
```

① fib(7) 호출 시, fib(3)은 5번 호출된다.

② 알고리즘의 시간 복잡도는 $O(\log_2(n))$이다.

③ 알고리즘의 점화식은 F_n = cases 0, & n = 0 \ 1, & n = 1 \ F_n-1 + F_n-2, & n > 1 cases이다.

④ 반복문을 사용하여 알고리즘 성능을 개선할 수 있다.

정답 2번

출제 포인트

이 문제는 재귀로 구현한 피보나치 알고리즘의 점화식, 부분문제 중복, 시간복잡도를 함께 판별하는 문제이다.

핵심 개념

피보나치 수열은 $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$로 정의된다.
주어진 코드는 같은 부분문제를 반복 호출하는 순수 재귀이므로 호출 수가 급격히 늘어나 시간복잡도는 $O(\phi^n)$이며 보통 상한으로 $O(2^n)$라고 본다.
반복문이나 동적 계획법을 쓰면 같은 값을 다시 계산하지 않아 성능을 개선할 수 있다.

NoteBase

선지별 해설

① O: fib(7)에서 fib(3)의 호출 횟수는 fib(6)에서의 횟수와 fib(5)에서의 횟수를 더한 값이다. 이를 따라가면 1, 1, 2, 3, 5로 늘어나므로 fib(7)에서는 fib(3)이 5번 호출된다.
② X: 틀린 부분은 "시간 복잡도는 $O(\log_2 n)$"이라는 표현이다. 이 코드는 두 개의 재귀호출을 반복해 같은 값을 여러 번 계산하므로 로그 시간이 아니라 지수 시간이며, 올바르게는 시간복잡도가 $O(\phi^n)$ 또는 상한 기준으로 $O(2^n)$이다.
③ O: 코드의 종료 조건은 $n = 0$일 때 0, $n = 1$일 때 1을 반환하고, 그 외에는 두 이전 항의 합을 반환하므로 피보나치 점화식과 일치한다.
④ O: 반복문으로 이전 두 항만 갱신하면 같은 값을 중복 계산하지 않는다. 따라서 순수 재귀보다 더 효율적인 $O(n)$ 시간 알고리즘으로 개선할 수 있다.

오답 포인트

재귀식 $F_n = F_{n-1} + F_{n-2}$만 보고 곧바로 시간복잡도도 작을 것이라고 착각하기 쉽다. 이 문항의 함정은 수열의 점화식과 알고리즘의 수행시간 점화식이 다르다는 점이다.

다시 나오면 이렇게 판단

재귀 알고리즘 문제에서는 먼저 종료 조건을 확인하고, 다음으로 같은 부분문제가 중복 호출되는지 본다.
부분문제 중복이 그대로 남아 있으면 로그 시간이 아니라 대개 지수 시간으로 본다.

암기 포인트

피보나치 순수 재귀는 점화식은 단순하지만 호출 수는 폭증하므로 수열 정의와 시간복잡도를 분리해서 본다.

NoteBase

문 3. 문제 해결을 위한 알고리즘을 기술하는 방법이 아닌 것은?

① 순서도(flowchart)를 사용하여 도식화한다.

② 의사 코드(pseudo code)를 사용하여 기술한다.

③ 추상 데이터 타입(abstract data type)만으로 기술한다.

④ 단계별로 명확한 문장(자연어)으로 기술한다.

정답 3번

출제 포인트

이 문제는 알고리즘을 기술하는 대표 방식과 추상 데이터 타입의 개념을 구별할 수 있는지를 묻는다.

핵심 개념

순서도는 처리 절차를 기호와 흐름선으로 도식화한 표현 방식이다.
의사 코드는 프로그램과 유사한 형식으로 알고리즘 절차를 서술한 표현 방식이다.
자연어 기술은 단계별 절차를 일상 언어 문장으로 설명하는 방식이다.
추상 데이터 타입은 자료와 그 자료에 허용되는 연산을 추상적으로 정의한 개념으로, 알고리즘 절차 자체를 기술하는 방식과는 다르다.

선지별 해설

1번 O: 순서도는 시작, 처리, 판단, 반복 흐름을 그림으로 나타내므로 알고리즘을 도식화하여 기술하는 대표 방법이다. 발문은 '아닌 것'을 묻기 때문에 정답은 아니다.
2번 O: 의사 코드는 특정 프로그래밍 언어에 얽매이지 않고 알고리즘의 절차를 단계적으로 적는 방식이므로 알고리즘 기술 방법에 해당한다. 발문에서는 오답 선택지이다.
3번 X: 틀린 부분은 '추상 데이터 타입만으로 기술한다'이다. 추상 데이터 타입은 스택, 큐처럼 데이터의 구조와 연산을 정의하는 개념이지 알고리즘 절차를 기술하는 대표 표현 방식이 아니다. 올바르게는 '알고리즘은 순서도, 의사 코드, 자연어 등으로 기술하고, 추상 데이터 타입은 자료와 연산을 추상화하는 데 사용한다'이다.
4번 O: 자연어로 각 단계를 명확히 서술하는 것도 알고리즘 기술 방법에 해당한다. 다만 모호성이 생길 수 있어 보조적 한계가 있을 뿐, 방법 자체는 맞다.

오답 포인트

3번의 '추상 데이터 타입'을 알고리즘 표현 방식으로 착각하면 틀린다. 추상 데이터 타입은 무엇을 저장하고 어떤 연산을 허용하는지를 정의하는 개념이고, 알고리즘 기술은 그 연산을 어떤 순서로 수행하는지를 표현하는 일이라는 점이 함정이다.

NoteBase

다시 나오면 이렇게 판단

선지가 절차를 어떻게 표현하는가를 말하면 알고리즘 기술 방식으로 본다. 선지가 자료의 구성 요소와 허용 연산을 정의하면 추상 데이터 타입 쪽 개념으로 판단한다.

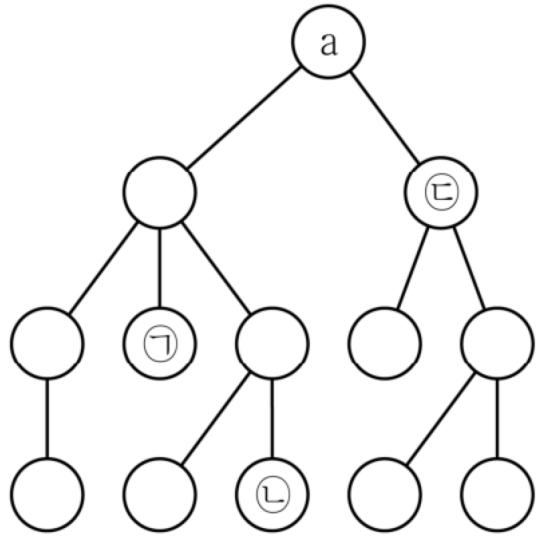
암기 포인트

알고리즘은 절차를 표현하고, 추상 데이터 타입은 자료와 연산을 정의한다.

NoteBase

문 4. 다음 (가) 트리(tree)를 깊이 우선 탐색(DFS, Depth First Search) 알고리즘으로 방문한 순서가 (나)와 같을 때, ㉠ ~ ㉢에 들어갈 노드는?

(가)



(나) a → b → c → d → e → f → g → h → i → j → k → l → m

① ㉠: e, ㉡: g, ㉢: c

② ㉠: e, ㉡: h, ㉢: i

③ ㉠: k, ㉡: g, ㉢: i

④ ㉠: k, ㉡: h, ㉢: c

정답 2번

출제 포인트

이 문제는 트리 도식에서 DFS 전위 순회를 왼쪽 자식부터 적용하여 빈 노드 ㄱ, ㄴ, ㄷ의 방문 순서를 대응시키는 문제이다.

NoteBase

선지별 해설

① X
DFS는 부모를 먼저 방문한 뒤 왼쪽 자식 서브트리부터 끝까지 내려간다. 왼쪽 큰 서브트리의 방문 순서는 b → c → d → e → f → g → h이므로 ㄱ은 e까지는 맞지만 ㄴ은 g가 아니라 h이다. 또한 오른쪽 큰 서브트리의 루트가 다음 방문 노드 i이므로 ㄷ은 c가 아니라 i이다. 따라서 올바르게는 'ㄱ=e, ㄴ=h, ㄷ=i'이다.

② O
전체 전위 순서는 a → b → c → d → e → f → g → h → i → j → k → l → m이다. 따라서 왼쪽 큰 서브트리의 가운데 리프인 ㄱ은 e, 그 오른쪽 서브트리의 아래 리프인 ㄴ은 h, 오른쪽 큰 서브트리의 루트인 ㄷ은 i가 되어 조합이 정확하다.

③ X
오른쪽 큰 서브트리의 루트 ㄷ을 i로 본 점은 맞다. 그러나 ㄱ은 왼쪽 큰 서브트리에서 c, d 다음에 바로 방문되는 리프이므로 k가 아니라 e이고, ㄴ은 f의 왼쪽 자식 g를 방문한 다음에 방문되므로 g가 아니라 h이다. 따라서 올바르게는 'ㄱ=e, ㄴ=h, ㄷ=i'이다.

④ X
ㄴ을 h로 본 점은 맞다. 그러나 ㄱ은 왼쪽 큰 서브트리에서 세 번째 자식 위치의 리프가 아니라 b의 두 번째 자식 리프이므로 k가 아니라 e이고, ㄷ은 오른쪽 큰 서브트리의 시작 노드이므로 c가 아니라 i이다. 따라서 올바르게는 'ㄱ=e, ㄴ=h, ㄷ=i'이다.

오답 포인트

DFS를 '왼쪽으로만 내려가는 순회'로 막연히 외우면 형제 노드 사이의 방문 시점을 놓치기 쉽다. 이 문항에서는 왼쪽 큰 서브트리 전체를 b부터 h까지 모두 끝낸 뒤에야 오른쪽 큰 서브트리의 루트 i로 넘어간다는 점을 놓치면 ㄷ을 c나 k처럼 잘못 배치하게 된다.

다시 나오면 이렇게 판단

전위 DFS는 '부모 방문 → 자식들을 왼쪽부터 재귀적으로 모두 방문'의 순서로 판단한다. 각 빈칸이 루트인지, 리프인지, 어느 부모의 몇 번째 자식인지 먼저 확인한 뒤 방문 순서를 대입하면 된다.

암기 포인트

전위 DFS는 부모 먼저, 서브트리는 왼쪽부터 끝까지 방문한다.

NoteBase

문 5. 1부터 n까지의 합을 계산하는 다음 알고리즘의 빈칸에 들어갈 코드는? (단, n ≥ 1인 정수이다)

```
int sum(n){
   if (n == 1)
      return 1;
   else
      [          ];
}
```

① return n + sum(n - 1)

② return n - sum(n - 1)

③ return n + sum(n + 1)

④ return n - sum(n + 1)

정답 1번

출제 포인트

이 문제는 1부터 n까지의 합을 재귀 함수로 표현할 때 종료 조건과 재귀 호출 식을 올바르게 연결하는지를 묻는 문제이다.

핵심 개념

재귀 함수는 자기 자신을 호출하면서 더 작은 문제로 줄여 가는 함수이다.
이 문제의 핵심은 sum(1)=1을 종료 조건으로 두고, sum(n)은 현재 값 n을 더한 뒤 더 작은 문제인 sum(n-1)로 내려가야 한다는 점이다.
즉 1부터 n까지의 합은 sum(n)=n+sum(n-1)의 형태가 되어야 한다.

NoteBase

선지별 해설

① O
return n + sum(n - 1)은 현재의 n을 결과에 포함하고, 호출 인자를 1씩 줄여 결국 n==1에 도달하므로 올바른 재귀식이다.

② X
틀린 부분은 '-'이다.
1부터 n까지의 합은 이전까지의 합에 현재 값 n을 더해야 하므로 빼면 안 된다.
올바르게는 return n + sum(n - 1)이다.

③ X
틀린 부분은 'sum(n + 1)'이다.
재귀 호출은 종료 조건인 n==1 쪽으로 가까워져야 하는데 n+1은 문제가 더 커져 종료 조건에 도달하지 못한다.
올바르게는 return n + sum(n - 1)이다.

④ X
틀린 부분은 '-'와 'sum(n + 1)'이다.
합을 구하는데 빼기를 사용했고, 재귀 호출도 n을 줄이지 않아 종료 조건으로 수렴하지 않는다.
올바르게는 return n + sum(n - 1)이다.

오답 포인트

②는 합을 구하면서도 누적 방향을 '+'가 아니라 '-'로 바꾼 점이 함정이다. ③과 ④는 재귀 호출 인자를 n+1로 두어 종료 조건으로 가지 못하게 만든 점이 핵심 함정이다. 재귀 문제에서는 연산 부호와 함께 호출 인자가 종료 조건 쪽으로 줄어드는지를 같이 봐야 한다.

다시 나오면 이렇게 판단

재귀식 문제는 먼저 종료 조건이 무엇인지 확인하고, 그다음 재귀 호출이 그 종료 조건 방향으로 한 단계씩 가까워지는지 본다. 합 문제라면 현재 항을 누적하는 연산이 '+'인지까지 함께 확인하면 된다.

암기 포인트

재귀는 현재 값 반영 + 더 작은 문제 호출의 구조이며, 1부터 n까지의 합은 sum(n)=n+sum(n-1), sum(1)=1이다.

NoteBase

문 6. 힙(heap)에 대한 설명으로 옳지 않은 것은? (단, n은 데이터의 개수이다)

① 힙을 구성하는 완전 이진 트리는 1차원 배열을 사용하여 구현할 수 있다.

② 최대 힙(max heap)의 모든 부모 노드 값은 그들의 자식 노드 값보다 크거나 같다.

③ 새로운 데이터 삽입 후 힙 성질을 만족하기 위한 연산의 시간 복잡도는 $O(n\log_2(n))$이다.

④ 루트 노드에서 데이터 삭제 후 힙 성질을 만족하기 위한 연산의 시간 복잡도는 $O(\log_2(n))$이다.

정답 3번

출제 포인트

이 문제는 힙의 구조 성질과 단일 삽입·삭제 연산의 시간 복잡도를 정확히 구분하는지를 묻는 문제이다.

핵심 개념

힙은 완전 이진 트리를 바탕으로 부모와 자식 사이의 대소 관계를 유지하는 자료구조이다.
최대 힙은 모든 부모 노드의 값이 각 자식 노드의 값보다 크거나 같은 힙이다.
새 원소 삽입 후 up-heap과 루트 삭제 후 down-heap은 모두 트리 높이만큼만 이동하므로 시간 복잡도는 $O(\log_2(n))$이다.

선지별 해설

① O: 힙은 완전 이진 트리이므로 노드를 레벨 순서대로 저장하면 1차원 배열 인덱스로 부모·자식 관계를 계산할 수 있어 배열 구현이 가능하다.
② O: 최대 힙의 핵심 성질은 모든 부모 노드 값이 자식 노드 값 이상이라는 점이다. 전체가 정렬되는 것은 아니지만 부모-자식 비교에서는 이 관계가 항상 유지된다.
③ X: 틀린 부분은 "$O(n\log_2(n))$"이다. 새 데이터를 맨 마지막 자리에 넣은 뒤 부모와 비교하며 위로 올리는 과정은 트리의 높이만큼만 진행되므로 $O(\log_2(n))$이다. 올바르게는 "새로운 데이터 삽입 후 힙 성질을 만족하기 위한 연산의 시간 복잡도는 $O(\log_2(n))$이다."이다.
④ O: 루트 노드를 삭제하면 마지막 노드를 루트로 올린 뒤 더 큰 자식과 교환하며 내려 보내는데, 이 과정도 트리 높이만큼만 수행되므로 $O(\log_2(n))$이다.

오답 포인트

③은 단일 삽입 한 번의 재정렬과 여러 데이터를 연속해서 넣어 힙을 만드는 전체 비용을 혼동하게 만드는 선지이다. 문제에서 묻는 것은 삽입 후 한 번의 힙 복구 연산이므로 $O(n\log_2(n))$까지 커지지 않는다.

다시 나오면 이렇게 판단

힙 문제에서는 먼저 완전 이진 트리 여부와 최대 힙·최소 힙의 부모-자식 대소 관계를 확인한다. 그다음 시간 복잡도는 한 번의 삽입·삭제인지, 아니면 n개의 데이터를 대상으로 전체 힙을 구성하는지부터 구분하면 된다.

NoteBase

암기 포인트

힙은 배열로 구현하는 완전 이진 트리이며, 단일 삽입과 루트 삭제 후 재정렬은 모두 높이만큼만 움직여 $O(\log_2(n))$이다.

NoteBase

문 7. 병합(merge) 정렬과 퀵(quick) 정렬에 대한 설명으로 옳지 않은 것은?

① 병합 정렬과 퀵 정렬은 모두 재귀 함수(recursive function)로 구현할 수 있다.

② 병합 정렬과 퀵 정렬은 모두 분할 정복(divide-and-conquer) 알고리즘에 속한다.

③ 퀵 정렬은 병합 정렬과 달리 배열 내 요소 간 데이터를 교환(swap)하지 않는다.

④ 퀵 정렬은 입력의 피봇(pivot) 값을, 병합 정렬은 중간 위치를 사용하여 배열을 분할한다.

정답 3번

출제 포인트

이 문제는 병합 정렬과 퀵 정렬의 공통점과 차이점, 특히 분할 방식과 원소 교환 여부를 구별하는지를 묻는 문제이다.

핵심 개념

병합 정렬은 배열을 중간 위치를 기준으로 반씩 나눈 뒤 정렬된 부분 배열을 다시 병합하는 분할 정복 정렬이다.
퀵 정렬은 피벗을 기준으로 작은 값과 큰 값을 나누도록 분할하면서 정렬하는 분할 정복 정렬이다.
병합 정렬은 보통 병합 과정에서 추가 메모리를 사용하며, 퀵 정렬은 분할 과정에서 배열 내부 원소를 서로 교환하는 방식으로 많이 구현된다.

NoteBase

선지별 해설

① O
병합 정렬과 퀵 정렬은 모두 재귀적으로 작은 부분 문제를 해결하는 방식으로 구현할 수 있다. 실제 표준적인 설명에서도 두 정렬 모두 재귀 호출 구조로 제시된다.

② O
두 정렬 모두 큰 문제를 더 작은 부분 문제로 나눈 뒤 해결하는 분할 정복 알고리즘에 속한다. 병합 정렬은 반으로 나누고, 퀵 정렬은 피벗 기준으로 나눈다는 차이만 있다.

③ X
틀린 부분은 "퀵 정렬은 ... 데이터를 교환(swap)하지 않는다"는 표현이다.
실제로는 퀵 정렬은 분할 과정에서 피벗보다 작은 값과 큰 값을 제자리에서 재배치하기 위해 원소 간 교환을 사용한다. 올바르게는 "퀵 정렬은 병합 정렬과 달리 분할 과정에서 배열 내 원소를 교환하는 방식으로 구현되는 경우가 많다"이다.

④ O
퀵 정렬은 입력에서 피벗 값을 정하고 그 기준으로 배열을 분할한다. 병합 정렬은 일반적으로 배열의 중간 위치를 기준으로 왼쪽과 오른쪽 부분 배열로 나눈다.

오답 포인트

③은 퀵 정렬이 빠르다는 이미지 때문에 병합 정렬과 비슷하게 단순 분할만 한다고 착각하면 틀리기 쉽다. 실제 함정은 퀵 정렬의 핵심이 피벗 기준 분할과 그 과정의 원소 교환이라는 점을 놓치게 만드는 데 있다.

다시 나오면 이렇게 판단

정렬 알고리즘 비교 문제에서는 먼저 분할 기준이 무엇인지, 그다음 배열 내부에서 원소 교환이 일어나는지, 추가 병합 과정이 필요한지를 순서대로 확인하면 된다. 피벗과 swap이 나오면 퀵 정렬, 중간 분할과 병합이 나오면 병합 정렬로 판단할 수 있다.

암기 포인트

병합 정렬은 중간 분할 후 병합, 퀵 정렬은 피벗 분할 중 swap이 핵심이다.

NoteBase

문 8. 다음 버블(bubble) 정렬 알고리즘 (가)에 대한 입력 배열이 (나)일 때, 해당 알고리즘에서 데이터 교환(swap 함수 실행) 직후 (다) 배열 상태에서의 end와 cur의 값은?

(가)
```
bubble_sort(arr[], arr_size) {
    // arr 배열의 인덱스는 0 ~ arr_size-1
    for end ← arr_size - 1 to 1
        for cur ← 0 to end - 1
            if (arr[cur] > arr[cur + 1])
                swap(arr[cur], arr[cur + 1]);
}
```

(나) 6, 2, 3, 4, 0, 5, 1, 9, 8, 7
(다) 2, 3, 0, 4, 5, 1, 6, 8, 7, 9

① end: 8, cur: 2
② end: 8, cur: 4
③ end: 7, cur: 2
④ end: 7, cur: 4

정답 1번

출제 포인트

이 문제는 버블 정렬의 바깥 반복 변수 end와 안쪽 반복 변수 cur가 진행되면서 어느 시점에 특정 배열 상태가 만들어지는지를 추적하는 문제이다.

NoteBase

올바른 순서

1. 첫 번째 패스는 end=9에서 cur=0부터 8까지 진행된다.
2. 이 패스가 끝나면 가장 큰 값 9가 맨 뒤로 이동하여 배열은 2, 3, 4, 0, 5, 1, 6, 8, 7, 9가 된다.
3. 다음 패스는 end=8에서 다시 cur=0부터 시작하며, cur=0과 cur=1에서는 교환이 없다.
4. cur=2에서 4와 0을 비교하면 4>0이므로 교환되고 배열이 2, 3, 0, 4, 5, 1, 6, 8, 7, 9가 된다.

왜 이 순서: 버블 정렬은 한 패스가 끝날 때마다 끝쪽 원소 하나가 확정되므로 먼저 end=9 전체 패스가 끝나야 하고, 그다음 end=8 패스의 cur=2에서야 (다) 상태가 나온다.

선지별 해설

① O: (다) 배열은 end=8인 두 번째 패스에서 cur=2일 때 4와 0을 교환한 직후 상태와 정확히 일치한다.

② X: end=8은 맞지만 cur=4는 아니다. cur=4까지 진행하면 5와 1도 추가로 교환되어 배열이 2, 3, 0, 4, 1, 5, 6, 8, 7, 9가 된다. 틀린 표현은 "cur=4"이고, 올바르게는 "cur=2"이다.

③ X: cur=2에서 비슷한 교환이 다시 나올 수 있다고 착각하기 쉽지만 end=7인 시점은 이미 end=8 패스를 끝낸 뒤이다. 따라서 배열 상태가 더 진행되어 있어 (다)와 같을 수 없다. 틀린 표현은 "end=7"이고, 올바르게는 "end=8"이다.

④ X: end=7, cur=4는 두 번째 패스보다 한 단계 더 뒤의 시점이므로 배열이 이미 더 많이 정렬된 상태이다. 틀린 표현은 "end=7, cur=4"이고, 올바르게는 "end=8, cur=2"이다.

오답 포인트

버블 정렬에서 교환이 일어나는 위치만 보고 cur를 맞추고, 그 교환이 어느 패스에서 발생했는지인 end를 놓치기 쉽다. 특히 첫 패스가 끝나서 9가 맨 뒤에 확정된 뒤라는 점을 놓치면 ③이나 ④처럼 더 뒤의 패스로 잘못 판단하게 된다.

다시 나오면 이렇게 판단

버블 정렬 추적 문제는 먼저 한 패스가 끝날 때마다 맨 뒤에 확정되는 원소를 확인하고, 그다음 해당 패스 안에서 cur를 0부터 차례로 움직이며 교환 직후 배열을 비교하면 된다.

암기 포인트

버블 정렬은 한 패스 종료마다 최댓값 하나가 오른쪽 끝에 확정되고, end는 그 확정 구간을 하나씩 줄여 간다.

NoteBase

문 9. 다음 알고리즘에 해당하는 정렬은?

```
sort(arr[], n){ // arr 배열의 인덱스는 0 ~ n-1
   for i ← 1 to n - 1 {
      cur ← arr[i];
      j ← i - 1;
      while (j ≧ 0) and (arr[j] > cur) {
         arr[j + 1] ← arr[j];
         j ← j - 1;
      }
      arr[j + 1] ← cur;
   }
}
```

① 쉘(shell) 정렬
② 기수(radix) 정렬
③ 선택(selection) 정렬
④ 삽입(insertion) 정렬

정답 4번

출제 포인트

이 문제는 의사코드의 핵심 동작을 보고 삽입 정렬, 선택 정렬, 셸 정렬, 기수 정렬을 구별하는 문제이다.

핵심 개념

삽입 정렬은 현재 원소를 `cur`에 저장한 뒤, 앞쪽의 정렬된 구간에서 자신보다 큰 값을 한 칸씩 오른쪽으로 밀고 빈 자리에 `cur`를 끼워 넣는 정렬이다.
선택 정렬은 매 단계에서 최솟값이나 최댓값을 찾아 선택한 뒤 한 번 교환하는 정렬이다.
셸 정렬은 일정 간격 `gap`을 두고 떨어진 원소들끼리 부분 삽입 정렬을 수행하는 정렬이다.
기수 정렬은 자리수별로 버킷에 분배하고 다시 모으는 비교하지 않는 정렬이다.

NoteBase

선지별 해설

① X이다. 틀린 부분은 이 코드를 셸 정렬로 본 점이다. 실제로는 `gap` 변수가 전혀 없고 인접한 앞 원소들과만 비교하면서 큰 값을 오른쪽으로 미는 구조이므로 셸 정렬이 아니라 삽입 정렬이다.
② X이다. 틀린 부분은 이 코드를 기수 정렬로 본 점이다. 실제로는 자리수별 버킷 분배 과정이 없고 `arr[j] > cur`처럼 원소 크기 비교를 직접 수행하므로 기수 정렬이 아니다.
③ X이다. 틀린 부분은 이 코드를 선택 정렬로 본 점이다. 실제로는 최솟값을 따로 찾아 한 번 교환하지 않고, 현재 원소를 앞의 정렬 구간 속 알맞은 위치로 밀어 넣으므로 선택 정렬이 아니다.
④ O이다. `cur <- arr[i]`로 현재 값을 잡고, `while (j >= 0) and (arr[j] > cur)` 동안 큰 값을 `arr[j+1]`로 한 칸씩 뒤로 민 다음, 마지막에 `arr[j+1] <- cur`로 끼워 넣는다. 이는 삽입 정렬의 전형적인 의사코드이다.

오답 포인트

정렬 이름보다 의사코드의 동작 순서를 보지 않으면 헷갈리기 쉽다. 특히 `앞에서 뒤로 한 칸씩 민다`는 동작은 삽입 정렬의 신호인데, 이를 단순히 '중간에 넣는 느낌' 정도로만 보면 셸 정렬이나 선택 정렬과 혼동하기 쉽다.

다시 나오면 이렇게 판단

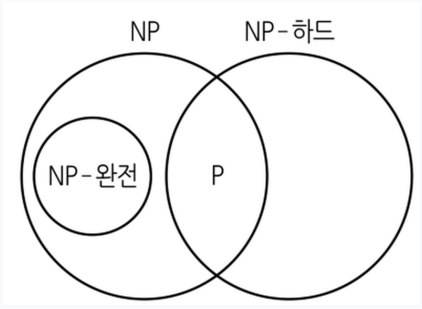
현재 원소를 임시 변수에 저장하고, 앞쪽 원소들을 비교해 큰 값만 한 칸씩 뒤로 민 뒤 빈 자리에 다시 넣으면 삽입 정렬로 판단한다. 반대로 최솟값 탐색 후 교환이면 선택 정렬, `gap`이 있으면 셸 정렬, 자리수 버킷이면 기수 정렬로 구분된다.

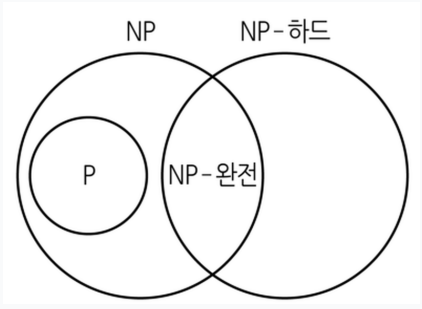
암기 포인트

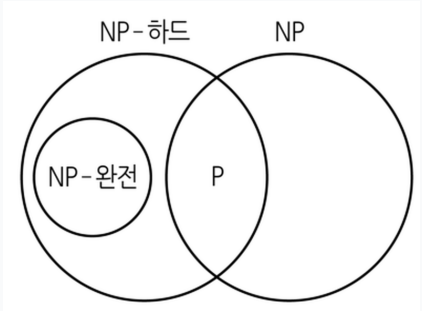
`cur` 저장 → 큰 값 오른쪽 이동 → 빈자리 삽입이면 삽입 정렬이다.

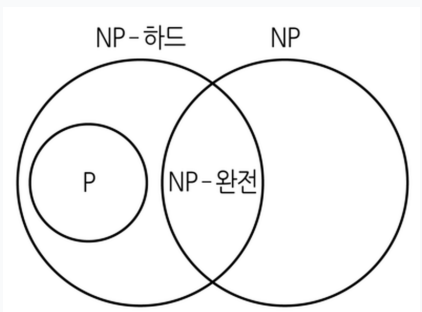
NoteBase

문 10. 문제 분류(class)간 포함관계도로 옳은 것은? (단, P ≠ NP일 경우이다)

① 

② 

③ 

④ 

정답 2번

출제 포인트

이 문제는 $P \neq NP$일 때 $P \subset NP$, NP-완전 = NP NP-하드, 그리고 NP-하드에는 NP 바깥 문제도 포함될 수 있다는 관계를 도식으로 정확히 배치하는지를 묻는 문제이다.

NoteBase

선지별 해설

① X: 이 도식은 P를 NP와 NP-하드의 겹치는 부분에 두고, NP-완전을 NP 안쪽 단독 영역에 둔다. $P \neq NP$이면 P 안에 NP-완전 문제가 있을 수 없고, NP-완전은 반드시 NP와 NP-하드의 교집합에 있어야 한다. 틀린 표현은 P와 NP-완전의 위치 설정이며, 올바르게는 P는 NP 내부이되 NP-하드와 겹치지 않고 NP-완전은 교집합에 있어야 한다.
② O: P가 NP 내부에 있고 NP-하드와는 분리되어 있으며, NP-완전이 NP와 NP-하드의 겹치는 부분에 있다. 이는 $P \subset NP$, NP-완전 = NP NP-하드, $P \neq NP$이면 P와 NP-하드가 겹치지 않는다는 관계를 모두 반영한 도식이다.
③ X: 이 도식은 원 집합 이름이 바뀐 상태에서 P를 교집합에 두고 NP-완전을 NP-하드 내부 단독 영역에 둔다. 그러나 P는 NP 안쪽에 있어야 하고 NP-완전은 NP에도 속해야 하므로 교집합 밖에 단독으로 있을 수 없다. 틀린 표현은 P를 교집합에 둔 점과 NP-완전을 NP-하드 단독 영역에 둔 점이며, 올바르게는 P는 NP 내부 단독 영역, NP-완전은 교집합 영역에 있어야 한다.
④ X: 이 도식은 P를 NP-하드 내부에 두었다. $P \neq NP$이면 NP-하드 문제 중 다항시간에 풀리는 문제가 존재할 수 없으므로 P가 NP-하드 안에 들어가면 안 된다. 틀린 표현은 P의 위치이며, 올바르게는 P는 NP 안에만 있고 NP-하드와는 겹치지 않아야 한다.

오답 포인트

많이 틀리는 지점은 NP-완전을 그냥 NP의 부분집합으로만 보고 교집합에 두지 않거나, P도 어려운 문제 집합이라는 인상 때문에 NP-하드와 겹친다고 보는 점이다. 이 문항은 특히 ①과 ④처럼 P의 위치를 흔들어 혼동을 유도한다.

다시 나오면 이렇게 판단

먼저 $P \subset NP$를 잡고, 다음으로 NP-완전은 NP이면서 동시에 NP-하드라는 점에서 교집합에 둔다. 마지막으로 $P \neq NP$ 조건이 보이면 P와 NP-완전, P와 NP-하드가 겹치지 않는지 확인하면 된다.

암기 포인트

P는 NP 안, NP-완전은 NP와 NP-하드의 겹침, NP-하드는 NP 밖으로도 뻗는 집합이다.

NoteBase

문 11. 다음 설명에 해당하는 알고리즘은?

◯ 문제의 입력 사례 분할 후, 분할된 문제의 해답을 우선 구한다.
◯ 문제의 입력 사례를 분할할 때, 재귀 관계식을 사용할 수 있다.
◯ 분할된 문제의 해답을 저장하여 분할 전 문제의 해답을 구하는 상향식 방법이다.

① 브루트포스(bruteforce)
② 백 트래킹(backtracking)
③ 분기 한정법(branch-and-bound)
④ 동적 프로그래밍(dynamic programming)

정답 4번

출제 포인트

분할정복과 동적 프로그래밍의 차이, 특히 부분 문제의 해를 저장해 재사용하는지와 상향식 접근을 쓰는지를 구별하는 문제이다.

핵심 개념

분할정복은 문제를 더 작은 부분 문제로 나누어 각각 해결한 뒤 결합하는 방식이다. 동적 프로그래밍은 중복되는 부분 문제의 해를 저장해 재사용하면서 전체 해를 구하는 방식이다. 백트래킹은 조건에 맞지 않는 경로를 되돌아가며 탐색하는 방식이고, 분기 한정법은 경계값으로 유망하지 않은 분기를 제거하는 최적화 탐색 방식이다.

NoteBase

선지별 해설

① X
브루트포스는 가능한 경우를 전부 시도하는 완전탐색이다. 제시문처럼 문제를 부분 문제로 나누고 그 해를 저장해 재사용하는 방식이 아니다. 틀린 표현은 "브루트포스"이다. 올바르게는 "브루트포스는 가능한 해를 전부 대입해 보는 완전탐색 기법이다"이다.

② X
백트래킹은 탐색 중 조건에 맞지 않으면 그 경로를 포기하고 되돌아가는 방식이다. 제시문의 핵심인 재귀 관계식과 부분 문제 해의 저장은 백트래킹의 본질이 아니다. 틀린 표현은 "백트래킹"이다. 올바르게는 "백트래킹은 유망하지 않은 경로를 중간에 버리며 탐색 공간을 줄이는 기법이다"이다.

③ X
분기 한정법은 최적화 문제에서 bound를 이용해 더 좋은 해가 나올 수 없는 분기를 제거하는 방식이다. 제시문 3의 "분할된 문제의 해답을 저장"과 "상향식 방법"은 분기 한정법이 아니라 동적 프로그래밍의 특징이다. 틀린 표현은 "분기 한정법"이다. 올바르게는 "분기 한정법은 경계값을 이용해 불필요한 분기를 잘라 내는 최적화 기법이다"이다.

④ O
동적 프로그래밍은 부분 문제의 해를 저장해 중복 계산을 피하면서 전체 문제를 푸는 방식이다. 제시문 1, 2는 분할정복과도 겹칠 수 있지만, 제시문 3의 "해답 저장"과 "상향식 방법"이 동적 프로그래밍을 확정한다.

오답 포인트

문제의 앞부분만 보고 "문제를 분할한다"는 표현에 끌리면 분할정복 계열로 오인하기 쉽다. 이 문항의 결정적 단서는 ③ 설명의 "분할된 문제의 해답을 저장"과 "상향식"이며, 이 두 표현이 나오면 동적 프로그래밍 쪽으로 판단이 기운다.

다시 나오면 이렇게 판단

알고리즘 설명형 문항에서는 먼저 핵심 동사를 본다. "전부 시도"이면 브루트포스, "되돌아감"이면 백트래킹, "bound로 가지 제거"이면 분기 한정법, "부분 문제 해 저장·재사용"이면 동적 프로그래밍으로 판단할 수 있다.

암기 포인트

동적 프로그래밍은 저장과 재사용, 백트래킹은 되돌아감, 분기 한정법은 가지치기, 브루트포스는 전수조사이다.

NoteBase

문 12. 분할 정복(divide-and-conquer) 알고리즘에 대한 설명으로 옳은 것은?

① 이진 탐색(binary search)은 분할 정복 알고리즘에 속한다.

② 분할 정복에 기반을 둔 알고리즘의 시간 복잡도는 최악의 경우 $O(\log_2(n))$이다.

③ 반복문(iteration)만을 사용하여 큰 문제를 작은 문제들로 분할한다.

④ n개의 정수들 중 k번째로 큰 정수를 찾는 선택(selection) 문제에서 문제의 크기는 재귀적으로 절반씩 분할된다.

정답 1번

출제 포인트

이 문제는 분할 정복의 핵심 성질과 대표 예시를 정확히 구별하는지를 묻는 문제이다.

핵심 개념

분할 정복은 큰 문제를 같은 성질의 작은 부분 문제로 나누고, 각 부분 문제를 해결한 뒤 결과를 결합하여 원문제를 푸는 방법이다. 키워드는 분할, 재귀적 해결, 결합이다.
이진 탐색은 정렬된 구간을 절반으로 나누며 답이 있는 쪽만 계속 탐색하는 대표적인 분할 정복 알고리즘이다.
선택 문제는 피벗 기준으로 한쪽 부분 문제만 재귀적으로 처리할 수 있지만, 부분 문제의 크기가 항상 절반으로 줄어드는 것은 아니다.

선지별 해설

① O: 이진 탐색은 탐색 구간을 절반으로 분할하고 필요한 한쪽 구간만 재귀적으로 해결하므로 분할 정복의 대표 예시에 해당한다.
② X: 틀린 부분은 "시간 복잡도는 최악의 경우 $O(\log_2 n)$이다"이다. 분할 정복은 하나의 고정된 시간 복잡도를 가지는 기법이 아니며, 문제와 결합 비용에 따라 $O(\log n)$, $O(n\log n)$, $O(n)$ 등으로 달라진다.
③ X: 틀린 부분은 "반복문만을 사용하여"이다. 분할 정복은 보통 재귀를 사용하여 문제를 계속 쪼개고 해결하는 방식으로 설명된다. 반복문만 사용해야 하는 기법이 아니다.
④ X: 틀린 부분은 "문제의 크기는 재귀적으로 절반씩 분할된다"이다. 선택 문제에서는 피벗에 따라 한쪽 부분 문제만 남기므로 크기가 줄어들기는 하지만, 항상 정확히 절반으로 줄어드는 것은 아니다.

오답 포인트

②는 이진 탐색의 시간 복잡도 $O(\log n)$를 분할 정복 전체의 공통 성질처럼 일반화한 함정이다. ③은 분할 정복과 단순 반복 처리를 혼동하게 만들고, ④는 선택 문제의 부분 문제 축소가 "항상 절반"이라고 과장한 표현이 함정이다.

NoteBase

다시 나오면 이렇게 판단

분할 정복 문제를 보면 먼저 큰 문제를 부분 문제로 나누는지, 재귀적으로 해결하는지, 마지막에 결합하는지를 확인한다. 그다음 시간 복잡도나 부분 문제 크기를 하나의 값으로 단정하는 표현이 나오면 알고리즘별로 달라지는지 다시 점검하면 된다.

암기 포인트

분할 정복은 "분할-재귀 해결-결합"이고, 이진 탐색은 대표 예시이지만 시간 복잡도와 분할 비율은 알고리즘마다 다르다.

NoteBase

문 13. 다음 알고리즘에 대한 시간 복잡도는?

```
sum ← 0;
for j ← 1 to n
   for i ← 1 to j
      sum ← sum + 1;
for k ← 0 to n - 1
   sum ← sum + k;
```

① $O(n)$

② $O(n\log_2(n))$

③ $O(n^2)$

④ $O(2^n)$

정답 3번

출제 포인트

이 문제는 중첩 반복문의 총 실행 횟수를 합으로 계산하고, 별도의 단일 반복문 비용까지 합친 뒤 지배항으로 시간 복잡도를 판단하는 문제이다.

조건 정리

주어진 값은 바깥 반복문이 $j = 1$부터 n까지 돌고, 안쪽 반복문이 각 j마다 $i = 1$부터 j까지 돈다는 점이다.
구해야 할 값은 전체 알고리즘의 시간 복잡도이다.
주의할 조건은 `sum <- sum + k;`에서 k의 값이 커져도 문장 1회 실행 시간은 상수 시간이라는 점이다.

사용할 공식

중첩 반복문의 실행 횟수는 $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$이다.
이 공식을 쓰는 이유는 안쪽 반복문이 j번째 바깥 반복에서 정확히 j번 실행되기 때문이다. 뒤의 `for k <- 0 to n - 1` 반복문은 총 n번 실행되므로 $O(n)$이다.

NoteBase

계산 과정

1. 첫 번째 이중 반복문의 실행 횟수는 $1 + 2 + \cdots + n$이므로 $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}$이다.
2. 따라서 첫 번째 부분의 시간 복잡도는 $O(n^2)$이다.
3. 두 번째 반복문은 $k = 0$부터 $n - 1$까지 총 n번 실행되므로 $O(n)$이다.
4. 전체 시간 복잡도는 $O(n^2) + O(n) = O(n^2 + n)$이고, 지배항 기준으로 $O(n^2)$이다.

선지별 해설

① X

`O(n)`은 뒤의 단일 반복문만 따로 볼 때의 복잡도이다. 틀린 표현은 전체 알고리즘을 $O(n)$으로 본 점이다. 올바르게는 앞의 이중 반복문이 $O(n^2)$이므로 전체는 $O(n^2)$이다.

② X

`O(n \log_2 n)`이 되려면 반복 횟수가 보통 절반씩 줄거나 두 배 구간 분할 같은 로그 구조가 있어야 한다. 틀린 표현은 이중 반복문의 누적 횟수를 로그형으로 본 점이다. 올바르게는 $1 + 2 + \cdots + n$ 형태이므로 $O(n^2)$이다.

③ O

첫 번째 부분이 $\frac{n(n+1)}{2}$번 실행되어 $O(n^2)$이고, 두 번째 부분은 $O(n)$이다. 전체는 $O(n^2 + n)$이며 지배항은 n^2이므로 ③이 옳다.

④ X

`O(2^n)`은 보통 재귀 분기처럼 경우의 수가 지수적으로 늘어날 때 나온다. 틀린 표현은 이 반복문을 지수 시간으로 본 점이다. 올바르게는 반복 횟수가 삼각수만큼 누적될 뿐이므로 $O(n^2)$이다.

오답 포인트

학생이 `for`문 개수만 보고 막연히 $O(n^2)$ 또는 $O(n)$로 찍는 경우가 많다. 이 문제의 함정은 안쪽 반복문의 상한이 항상 n이 아니라 j라는 점이며, 그래서 실행 횟수를 직접 $1 + 2 + \cdots + n$으로 합산해야 한다는 데 있다. 또 `sum + k`에서 k 값이 커진다고 연산 시간이 커진다고 오해하기 쉽지만, 기본 시간 복잡도 분석에서는 한 번의 덧셈을 상수 시간으로 본다.

다시 나오면 이렇게 판단

반복문 시간 복잡도는 반복문 개수를 세지 말고 각 문장이 실제로 몇 번 실행되는지를 합으로 적어 판단한다. 안쪽 반복문의 상한이 바깥 변수에 따라 달라지면 $\sum$ 형태로 바꾼 뒤, 마지막에는 가장 큰 차수의 항만 남겨 지배항으로 정리한다.

암기 포인트

`1+2+\cdots+n` 형태가 보이면 먼저 $\frac{n(n+1)}{2}$를 떠올리고 시간 복잡도는 $O(n^2)$로 연결한다.

NoteBase

문 14. a^x을 계산하는 다음 알고리즘의 빈칸 ㉠~㉢에 해당하는 코드로 옳게 짝지어진 것은? (단, a와 x는 정수, $x \geq 0$, mod는 나머지 연산이다)

```
int power(a, x) {
   if (x == 1)
      return [ ㉠ ];
   else if (x == 0)
      return 1;
   if (x mod 2 == 0) {
      temp ← power(a, x / 2);
      return [ ㉡ ];
   } else {
      temp ← power(a, (x - 1) / 2);
      return [ ㉢ ] ;
   }
}
```

① ㉠: a * a, ㉡: temp * a, ㉢: temp * temp * a

② ㉠: a * a, ㉡: temp * temp, ㉢: a * a * temp

③ ㉠: a, ㉡: temp * a, ㉢: a * a * temp

④ ㉠: a, ㉡: temp * temp, ㉢: temp * temp * a

정답 4번

출제 포인트

이 문제는 분할 정복 거듭제곱에서 짝수 지수와 홀수 지수를 나누어 a^x를 재귀적으로 계산하는 원리를 묻는 문제이다.

조건 정리

주어진 값: 정수 a, 정수 x, $x \geq 0$이다.
구해야 할 값: 함수 power(a, x)가 항상 a^x를 반환하도록 빈칸 ㄱ, ㄴ, ㄷ을 채우는 것이다.
주의할 조건: $x = 0$이면 1, $x = 1$이면 a가 바로 반환되어야 하고, 짝수와 홀수에서 재귀식이 달라진다.

NoteBase

사용할 공식

사용 원리는 $a^{2k} = (a^k)^2$, $a^{2k+1} = (a^k)^2 \cdot a$이다.
이 공식을 쓰는 이유: 짝수 x는 $x/2$로, 홀수 x는 $(x-1)/2$로 줄여 계산한 뒤 제곱하거나 마지막에 a를 한 번 더 곱하면 원래의 a^x를 복원할 수 있기 때문이다.

계산 과정

1. $x = 1$이면 a^1을 반환해야 하므로 ㄱ=a이다.
2. x가 짝수이면 temp = power(a, x/2)이므로 temp는 $a^{x/2}$이다. 따라서 반환값은 a^x가 되도록 temp * temp여야 하므로 ㄴ=temp * temp이다.
3. x가 홀수이면 temp = power(a, (x-1)/2)이므로 temp는 $a^{(x-1)/2}$이다. 따라서 temp * temp * a는 $a^{x-1} \cdot a = a^x$가 되어야 하므로 ㄷ=temp * temp * a이다.

선지 판단

1번: X. ㄱ=a*a이면 $x = 1$일 때 a가 아니라 a^2를 반환하므로 기본 조건부터 틀린다. 또 ㄴ=temp*a이면 짝수 지수에서 제곱이 아니라 한 번만 곱해 a^x를 만들 수 없다. 올바르게는 ㄱ=a, ㄴ=temp*temp이다.
2번: X. ㄴ과 ㄷ 중 ㄷ=a*a*temp가 틀리다. 홀수 지수에서는 temp가 $a^{(x-1)/2}$이므로 여기에 a^2를 곱하면 $a^{(x+3)/2}$가 되어 일반적으로 a^x가 되지 않는다. 올바르게는 ㄷ=temp*temp*a이다.
3번: X. ㄱ=a는 맞지만 ㄴ=temp*a, ㄷ=a*a*temp가 모두 틀리다. 짝수는 절반 지수를 제곱해야 하고, 홀수는 절반 지수를 제곱한 뒤 a를 한 번만 더 곱해야 한다. 올바르게는 ㄴ=temp*temp, ㄷ=temp*temp*a이다.
4번: O. ㄱ=a, ㄴ=temp*temp, ㄷ=temp*temp*a이면 $x = 1$, 짝수 x, 홀수 x의 세 경우가 모두 a^x와 정확히 일치한다.

오답 포인트

기저 사례와 재귀 사례를 따로 보지 않으면 $x = 1$에서 a를 바로 반환해야 한다는 점을 놓치기 쉽다. 또 홀수 지수 처리에서 temp 자체에 a를 곱하는지, temp를 먼저 제곱한 뒤 a를 곱하는지를 혼동하면 1번, 2번, 3번 같은 오답으로 흐르기 쉽다.

다시 나오면 이렇게 판단

재귀 거듭제곱 문제는 기저 사례가 a^0, a^1을 맞게 반환하는지 먼저 본다. 그다음 짝수는 $a^{2k} = (a^k)^2$, 홀수는 $a^{2k+1} = (a^k)^2 \cdot a$ 형태로 바로 대응시키면 된다.

암기 포인트

분할 정복 거듭제곱은 짝수는 제곱, 홀수는 제곱 후 a 한 번 추가라는 구조로 기억하면 된다.

NoteBase

문 15. 정렬된 정수 배열을 입력으로 하는 이진 탐색 알고리즘에 대한 시간 복잡도는?

① $O(\log_2(n))$

② $O(n)$

③ $O(n\log_2(n))$

④ $O(n^2)$

정답 1번

출제 포인트

이 문제는 이진 탐색의 핵심 성질인 탐색 범위를 절반씩 줄인다는 점에서 시간 복잡도가 $O(\log_2 n)$이 된다는 사실을 묻는 문제이다.

핵심 개념

이진 탐색은 정렬된 배열에서 가운데 원소와 비교한 뒤 탐색 범위를 절반으로 줄여 가는 탐색 방법이다.
선형 탐색은 처음부터 끝까지 순차적으로 확인하므로 시간 복잡도가 $O(n)$이다.
정렬 알고리즘의 대표 복잡도인 $O(n\log_2 n)$이나 이중 반복의 $O(n^2)$와 달리, 이진 탐색은 비교 횟수가 단계마다 절반으로 감소한다.

선지별 해설

① O: 이진 탐색은 한 번 비교할 때마다 남은 탐색 구간이 절반이 되므로 비교 횟수는 대략 $\log_2 n$번이고 시간 복잡도는 $O(\log_2 n)$이다.
② X: 틀린 부분은 $O(n)$이라는 복잡도 표기이다. $O(n)$은 순차적으로 전부 확인하는 선형 탐색의 복잡도이고, 이진 탐색은 실제로 $O(\log_2 n)$이다.
③ X: 틀린 부분은 $O(n\log_2 n)$이라는 복잡도 표기이다. 이는 병합 정렬이나 힙 정렬처럼 정렬 과정에서 자주 보이는 복잡도이며, 이진 탐색의 복잡도는 $O(\log_2 n)$이다.
④ X: 틀린 부분은 $O(n^2)$이라는 복잡도 표기이다. $O(n^2)$는 이중 반복문이나 버블 정렬처럼 비교가 누적되는 경우에 해당하고, 이진 탐색은 매 단계마다 범위를 절반으로 줄이므로 $O(\log_2 n)$이다.

오답 포인트

탐색 문제라는 이유만으로 선형 탐색의 $O(n)$을 떠올리거나, 정렬된 배열이라는 표현 때문에 정렬 알고리즘의 $O(n\log_2 n)$을 섞어 고르기 쉽다. 이 문항의 함정은 알고리즘 이름이 아니라 탐색 범위가 어떻게 줄어드는지를 보지 않으면 복잡도 표기를 혼동한다는 점이다.

NoteBase

다시 나오면 이렇게 판단

정렬된 자료에서 가운데 값을 기준으로 절반씩 버리는 구조가 보이면 먼저 이진 탐색을 떠올리고 시간 복잡도를 $O(\log_2 n)$으로 판단한다. 처음부터 끝까지 순서대로 확인할 때만 $O(n)$이다.

암기 포인트

이진 탐색은 가운데 비교 후 절반 제거이므로 $O(\log_2 n)$, 끝까지 훑는 선형 탐색만 $O(n)$이다.

NoteBase

문 16. 다음 설명에 해당하는 복잡도 함수에 대한 시간 복잡도는?

◯ $T(1) = 27$
◯ $T(n)$은 궁극적으로 감소하지 않는다.
◯ $T(n) = 9T(\frac{n}{3}) + 8n$ ($n > 1$이고, n은 3의 거듭제곱이다)

① $\Theta(1)$
② $\Theta(\log_2(n))$
③ $\Theta(n)$
④ $\Theta(n^2)$

정답 4번

출제 포인트

이 문제는 점화식 $T(n) = 9T(n/3) + 8n$에 마스터 정리를 적용해 시간 복잡도를 판별하는 문제이다.

조건 정리

주어진 값: $T(1) = 27$, $T(n) = 9T(n/3) + 8n$
구해야 할 값: $T(n)$의 점근적 시간 복잡도
주의할 조건: $a = 9$, $b = 3$, $f(n) = 8n$으로 두고 $n^{\log_b a}$와 $f(n)$의 크기를 비교해야 한다.

사용할 공식

마스터 정리의 비교 기준은 $T(n) = aT(n/b) + f(n)$에서 $n^{\log_b a}$와 $f(n)$를 비교하는 것이다.
이 문제에서는 $a = 9$, $b = 3$이므로 $n^{\log_3 9} = n^2$가 되고, $f(n) = 8n$은 n^2보다 한 차수 작다.

계산 과정

1. $a = 9$, $b = 3$, $f(n) = 8n$으로 놓는다.
2. $\log_3 9 = 2$이므로 $n^{\log_3 9} = n^2$이다.
3. $f(n) = 8n = O(n)$이고 이는 $O(n^{2-1})$이므로 마스터 정리 1번 경우에 해당한다.
4. 따라서 $T(n) = \Theta(n^2)$이다.

NoteBase

선지별 해설

1번 X: $\Theta(1)$는 상수 시간인데, 이 점화식은 하위 문제가 9개로 분기되어 입력이 커질수록 연산량이 크게 증가한다. 틀린 표현은 "상수 시간"이고, 올바르게는 "$T(n) = \Theta(n^2)$이다"이다.
2번 X: $\Theta(\log_2 n)$는 문제 크기를 한 번에 절반씩 줄이는 단순 분할 구조에서 자주 나오지만, 이 문제는 매 단계마다 9개의 부분 문제를 푼다. 틀린 표현은 "로그 시간"이고, 올바르게는 "$n^{\log_3 9} = n^2$가 지배하므로 $\Theta(n^2)$이다"이다.
3번 X: $\Theta(n)$이라면 결합 비용 $8n$이 전체를 지배해야 하지만, 실제로는 재귀 호출 수에서 생기는 n^2 차수의 비용이 더 크다. 틀린 표현은 "선형 시간"이고, 올바르게는 "선형항 $8n$보다 재귀항이 더 커서 $\Theta(n^2)$이다"이다.
4번 O: $n^{\log_3 9} = n^2$이고 $f(n) = 8n$은 그보다 낮은 차수이므로 마스터 정리 1번 경우에 따라 전체 시간 복잡도는 $\Theta(n^2)$이다.

오답 포인트

겉에 보이는 $8n$만 보고 $\Theta(n)$으로 고르거나, 재귀 깊이만 보고 $\Theta(\log n)$으로 착각하기 쉽다. 이 문제의 핵심 함정은 "분할 횟수"보다 "각 단계의 부분 문제 개수"가 더 크게 작용한다는 점이다.

다시 나오면 이렇게 판단

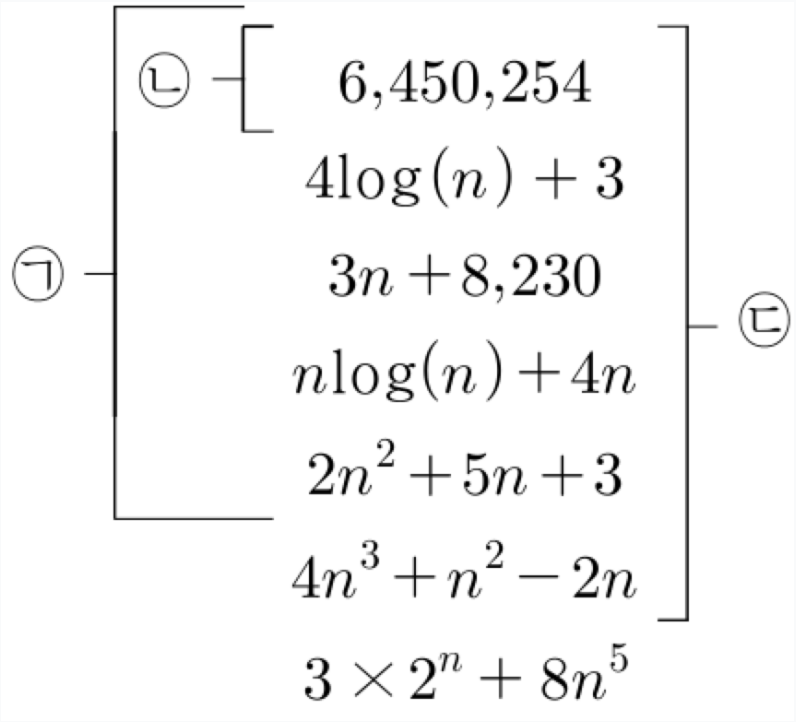
분할정복 점화식이 나오면 먼저 a, b, $f(n)$를 분리하고, 그다음 $n^{\log_b a}$를 계산해 $f(n)$와 차수를 비교하면 된다. $f(n)$보다 $n^{\log_b a}$가 크면 재귀 호출 구조가 전체 복잡도를 지배한다.

암기 포인트

$T(n) = aT(n/b) + f(n)$에서는 먼저 $n^{\log_b a}$를 구하고, $f(n)$가 그보다 낮은 차수이면 답은 $\Theta(n^{\log_b a})$이다.

NoteBase

문 17. 다음 수식들에 대해 ㉠~㉢에 해당하는 빅오(O) 표기가 옳게 짝지어진 것은? (단, k는 상수이다)



① ㉠: $O(2^n)$, ㉡: $O(1)$, ㉢: $O(n^k)$

② ㉠: $O(n^2)$, ㉡: $O(1)$, ㉢: $O(n^3)$

③ ㉠: $O(n^2)$, ㉡: $O(\log(n))$, ㉢: $O(n^k)$

④ ㉠: $O(n^2)$, ㉡: $O(\log(n))$, ㉢: $O(n^3)$

정답 2번

출제 포인트

이 문제는 여러 수식의 최고 지배항을 보고 ㄱ, ㄴ, ㄷ 묶음의 대표 빅오 표기를 고르는 문제이다.

조건 정리

ㄱ은 6,450,254, $4\log n+3$, $3n+8$, 230, $n\log n+4n$, $2n^2+5n+3$을 포함하므로 가장 큰 증가율은 n^2이다.

ㄴ은 상수식 6,450,254만 가리키므로 $O(1)$이다.

ㄷ은 ㄱ의 식들에 $4n^3+n^2-2n$까지 포함하므로 가장 큰 증가율은 n^3이다.

마지막 식 $3\times 2^n+8n^5$은 ㄱ, ㄴ, ㄷ 묶음에 포함되지 않는다.

NoteBase

사용할 공식

빅오 표기에서는 상수계수와 낮은 차수 항을 버리고 가장 빠르게 증가하는 항만 남긴다.
an^d + lower terms = $O(n^d)$이고, 상수식은 $O(1)$이다.
증가율 비교는 대략 $1 < \log n < n < n\log n < n^2 < n^3 < n^5 < 2^n$ 순서이다.

계산 과정

ㄱ: $4\log n + 3$은 $O(\log n)$, $3n + 8$, 230은 $O(n)$, $n\log n + 4n$은 $O(n\log n)$, $2n^2 + 5n + 3$은 $O(n^2)$이다. 이 묶음의 최댓값은 $O(n^2)$이다.
ㄴ: 6,450,254는 n에 따라 변하지 않는 상수이므로 $O(1)$이다.
ㄷ: ㄷ 묶음에서 최고 지배항은 $4n^3 + n^2 - 2n$의 n^3이므로 $O(n^3)$이다.

선지 판단

① X: ㄴ의 $O(1)$은 맞지만, ㄱ을 $O(2^n)$로 둔 부분이 틀린다. ㄱ에는 지수식이 없고 최고 지배항이 n^2이므로 ㄱ은 $O(n^2)$로 고쳐야 한다. ㄷ도 이 문항의 대표 표기로는 $O(n^3)$을 적는다.
② O: ㄱ은 $O(n^2)$, ㄴ은 $O(1)$, ㄷ은 $O(n^3)$이므로 세 묶음이 모두 맞다.
③ X: ㄱ의 $O(n^2)$는 맞지만, ㄴ을 $O(\log n)$로 둔 부분이 틀린다. ㄴ은 n과 무관한 상수식 6,450,254이므로 $O(1)$로 고쳐야 한다. ㄷ도 대표 최고차 표기는 $O(n^3)$이다.
④ X: ㄱ의 $O(n^2)$와 ㄷ의 $O(n^3)$은 맞지만, ㄴ을 $O(\log n)$로 둔 부분이 틀린다. ㄴ은 상수식이므로 $O(1)$로 고쳐야 한다.

오답 포인트

출제자가 노린 함정은 큰 숫자 상수 6,450,254를 로그나 다항식처럼 보거나, 아래의 $3 \times 2^n + 8n^5$를 ㄱ 또는 ㄷ 묶음에 포함시키는 데 있다. 빅오에서는 숫자의 크기가 아니라 n이 커질 때 변하는 항의 증가율이 기준이다.

다시 나오면 이렇게 판단

묶음 표시의 범위를 먼저 확인한 뒤, 각 식에서 상수계수와 낮은 차수 항을 제거하고 묶음 안에서 가장 빠르게 증가하는 항 하나를 고른다. 상수식은 값이 아무리 커도 $O(1)$로 판단한다.

암기 포인트

빅오는 크기 비교가 아니라 증가율 비교이다. 상수는 $O(1)$, 최고차 다항식 n^d는 $O(n^d)$, 지수식 2^n은 다항식보다 빠르게 증가한다.

NoteBase

문 18. 다음 문제를 해결하기 위한 알고리즘 관련 설명으로 옳은 것은?

◯ n개의 물건은 각기 고유의 무게와 가치를 갖는다.
◯ 물건은 나눌 수 없으며, 선택한 물건은 통째로 넣는다.
◯ 배낭 내 물건들의 총 무게는 배낭 용량을 초과할 수 없다.
◯ 최대의 가치를 갖도록 한정된 용량의 배낭에 물건을 넣는다.

① 탐욕 알고리즘을 사용할 경우 시간 복잡도는 $O(n)$이다.
② 탐욕 알고리즘은 문제에 대한 최적해를 계산할 수 없다.
③ 동적 프로그래밍 알고리즘은 가치 ÷ 무게(무게당 가치)를 사용한다.
④ 동적 프로그래밍 알고리즘은 힙(heap)을 구성하여 최적해를 탐색한다.

정답 2번

출제 포인트

이 문제는 0/1 배낭 문제에서 탐욕 알고리즘과 동적 계획법의 차이를 구분할 수 있는지를 묻는다.

핵심 개념

0/1 배낭 문제는 물건을 쪼갤 수 없고 선택한 물건을 통째로 넣어야 하므로, 제한 용량 안에서 전체 조합의 최댓값을 찾아야 하는 최적화 문제이다.
탐욕 알고리즘은 매 단계에서 좋아 보이는 기준으로 하나씩 선택하는 방법이지만, 0/1 배낭 문제에서는 지역적으로 좋은 선택이 전체 최적해를 보장하지 않는다.
동적 계획법은 물건 수와 배낭 용량을 상태로 두고, 물건을 넣는 경우와 넣지 않는 경우를 비교하여 부분 문제의 최적값을 누적한다.

NoteBase

선지별 해설

1번 X: 탐욕 알고리즘의 시간 복잡도를 단순히 $O(n)$이라고 단정할 수 없다. 보통 무게당 가치 같은 기준으로 정렬하면 정렬 비용이 들어가며, 더 근본적으로 0/1 배낭 문제에서 탐욕 선택은 최적해를 보장하는 정확한 알고리즘이 아니다. 틀린 표현은 "시간 복잡도는 $O(n)$이다"이다. 올바르게는 "탐욕 방식은 기준 선택과 정렬 방식에 따라 시간이 달라지고, 0/1 배낭 문제의 최적해를 보장하지 않는다"이다.
2번 O: 0/1 배낭 문제에서는 물건을 나눌 수 없으므로 무게당 가치가 큰 물건부터 고르는 탐욕 선택이 전체 조합의 최댓값으로 이어진다는 보장이 없다. 따라서 탐욕 알고리즘은 이 문제에 대한 최적해를 일반적으로 계산할 수 없다.
3번 X: 동적 계획법은 가치/무게 비율을 선택 기준으로 삼는 알고리즘이 아니다. 가치/무게 비율은 탐욕적 선택 기준으로 쓰일 수 있는 값이고, 동적 계획법은 각 물건을 포함하는 경우와 제외하는 경우의 최댓값을 표로 계산한다. 틀린 표현은 "가치 ÷ 무게를 사용한다"이다. 올바르게는 "동적 계획법은 부분 문제의 최적값과 포함/제외 점화식을 사용한다"이다.
4번 X: 동적 계획법은 힙을 구성하여 최적해를 탐색하는 방식이 아니다. 힙은 우선순위 큐가 필요한 탐색이나 분기 한정법에서 활용될 수 있지만, 배낭 문제의 대표적인 동적 계획법은 상태 테이블을 채운다. 틀린 표현은 "힙을 구성하여"이다. 올바르게는 "동적 계획법은 물건 인덱스와 남은 용량에 따른 DP 테이블을 구성하여 최적값을 계산한다"이다.

오답 포인트

출제자가 노린 함정은 분할 가능한 배낭 문제와 0/1 배낭 문제를 혼동하게 하는 데 있다. 물건을 나눌 수 있으면 무게당 가치 기준의 탐욕법이 최적해를 줄 수 있지만, 이 문제처럼 물건을 통째로 넣어야 하면 탐욕법은 최적해 보장이 없고 동적 계획법으로 조합을 비교해야 한다.

다시 나오면 이렇게 판단

배낭 문제가 나오면 먼저 물건을 나눌 수 있는지 확인한다. 나눌 수 있으면 무게당 가치 기준의 탐욕법을 떠올리고, 나눌 수 없으면 0/1 배낭 문제로 보아 동적 계획법의 포함/제외 비교를 떠올린다.

암기 포인트

0/1 배낭은 탐욕 최적 보장 없음, 동적 계획법은 포함과 제외를 비교하는 테이블 계산이다.

NoteBase

문 19. 각 정렬 알고리즘에 대한 사례별 시간 복잡도가 옳게 짝지어진 것은?

① 삽입(insertion) 정렬: 최선의 경우 $\Omega(n)$, 최악의 경우 $O(n^2)$

② 선택(selection) 정렬: 최선의 경우 $\Omega(n)$, 최악의 경우 $O(n^2)$

③ 힙(heap) 정렬: 최선의 경우 $\Omega(n\log_2(n))$, 최악의 경우 $O(n^2)$

④ 퀵(quick) 정렬: 최선의 경우 $\Omega(n\log_2(n))$, 최악의 경우 $O(n\log_2(n))$

정답 1번

출제 포인트

이 문제는 삽입 정렬, 선택 정렬, 힙 정렬, 퀵 정렬의 최선 및 최악 시간 복잡도를 정확히 구별하는지를 묻는다.

핵심 개념

삽입 정렬은 이미 정렬된 입력에서는 한 번씩만 비교해 $\Omega(n)$이고, 역순에 가까운 입력에서는 이동이 많이 발생해 $O(n^2)$이다.
선택 정렬은 매 단계 남은 원소 전체에서 최솟값을 찾으므로 입력 상태와 관계없이 비교 횟수가 $\Theta(n^2)$이다.
힙 정렬은 힙 구성과 반복 삭제 과정 때문에 최선과 최악 모두 $\Theta(n\log_2 n)$으로 본다.
퀵 정렬은 분할이 균형적이면 $\Theta(n\log_2 n)$, 한쪽으로 치우치면 $\Theta(n^2)$까지 나빠진다.

선지별 해설

① O: 삽입 정렬은 이미 정렬된 경우 각 원소가 거의 제자리에 있어 선형 시간 $\Omega(n)$으로 처리되고, 최악의 경우에는 각 원소가 앞쪽으로 많이 이동해 $O(n^2)$이 된다. 따라서 옳은 짝이다.
② X: 틀린 부분은 선택 정렬의 최선의 경우를 $\Omega(n)$으로 둔 표현이다. 선택 정렬은 입력이 이미 정렬되어 있어도 매 위치마다 남은 원소들을 비교해 최솟값을 찾아야 하므로 최선도 $\Omega(n^2)$ 또는 $\Theta(n^2)$으로 고쳐야 한다.
③ X: 틀린 부분은 힙 정렬의 최악의 경우를 $O(n^2)$으로 둔 표현이다. 힙 정렬은 힙에서 원소를 하나씩 꺼낼 때마다 $O(\log_2 n)$ 조정이 발생하므로 최악도 $O(n\log_2 n)$으로 고쳐야 한다.
④ X: 틀린 부분은 퀵 정렬의 최악의 경우를 $O(n\log_2 n)$으로 둔 표현이다. 퀵 정렬은 피벗이 계속 한쪽 끝으로 치우치면 분할 크기가 $n-1, 0$처럼 불균형해져 최악의 경우 $O(n^2)$으로 고쳐야 한다.

오답 포인트

선택 정렬은 교환 횟수만 보면 적어 보이지만 시간 복잡도 판단의 핵심은 최솟값을 찾는 비교 횟수이다. 또한 힙 정렬과 퀵 정렬을 모두 평균적으로 빠른 정렬로만 기억하면 ③의 힙 정렬 최악과 ④의 퀵 정렬 최악을 서로 혼동할 수 있다.

NoteBase

다시 나오면 이렇게 판단

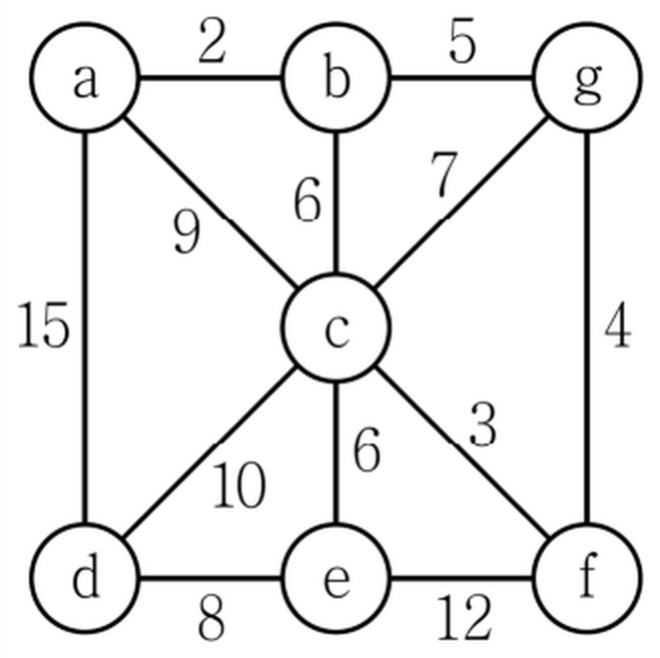
입력 상태에 따라 빨라지는 정렬인지 먼저 본다. 삽입 정렬은 거의 정렬된 입력에서 선형, 선택 정렬은 항상 이차, 힙 정렬은 항상 로그선형, 퀵 정렬은 피벗 분할이 나쁘면 이차로 판단한다.

암기 포인트

삽입은 최선 선형, 선택은 항상 이차, 힙은 항상 $n\log n$, 퀵은 최악 이차이다.

NoteBase

문 20. 다음 그래프에 대해 크루스칼(Kruskal) 알고리즘에 의한 최소 신장 트리(MST, Minimum Spanning Tree)를 계산할 때, 5번째로 MST에 포함되는 간선은? (단, 초기의 MST는 ∅이다)



① (b, c)

② (b, g)

③ (c, e)

④ (d, e)

정답 3번

출제 포인트

이 문제는 크루스칼 알고리즘에서 간선을 가중치 오름차순으로 검사하되, 사이클을 만드는 간선은 버리는 순서를 묻는 문제이다.

올바른 순서

가중치 순서는 (a,b)=2, (c,f)=3, (f,g)=4, (b,g)=5, (b,c)=6, (c,e)=6, (c,g)=7, (d,e)=8, (a,c)=9, (c,d)=10, (e,f)=12, (a,d)=15이다.
MST에 실제로 포함되는 간선은 1번째 (a,b), 2번째 (c,f), 3번째 (f,g), 4번째 (b,g), 5번째 (c,e), 6번째 (d,e)이다. (b,c)는 가중치 6이지만, 그 시점에는 b와 c가 이미 (b,g), (g,f), (f,c)를 통해 같은 연결요소에 있으므로 사이클을 만들어 제외된다.

NoteBase

선지별 해설

① (b,c): X. (b,c)의 가중치는 6이지만, (a,b), (c,f), (f,g), (b,g)가 이미 선택된 뒤에는 b와 c가 같은 연결요소에 속한다. 따라서 (b,c)를 추가하면 사이클이 생기므로 MST에 포함되지 않는다. 틀린 표현은 '(b,c)가 5번째 포함 간선'이라는 점이고, 올바르게는 '(b,c)는 검사되더라도 제외되는 간선'이다.
② (b,g): X. (b,g)의 가중치는 5이고, (a,b), (c,f), (f,g) 다음에 사이클 없이 두 연결요소를 이어 주므로 4번째로 MST에 포함된다. 틀린 표현은 '(b,g)가 5번째 포함 간선'이라는 점이고, 올바르게는 '(b,g)는 4번째 포함 간선'이다.
③ (c,e): O. 가중치 6의 (b,c)는 사이클 때문에 제외되고, (c,e)는 기존 연결요소에 새 정점 e를 붙이므로 사이클을 만들지 않는다. 따라서 (c,e)가 5번째로 MST에 포함된다.
④ (d,e): X. (d,e)의 가중치는 8이고, (c,e)가 5번째로 선택된 뒤 남은 정점 d를 연결하므로 6번째로 MST에 포함된다. 틀린 표현은 '(d,e)가 5번째 포함 간선'이라는 점이고, 올바르게는 '(d,e)는 6번째 포함 간선'이다.

오답 포인트

가중치만 보고 6인 (b,c)를 바로 고르면 틀린다. 크루스칼 알고리즘에서는 간선을 검사하는 순서와 MST에 포함되는 순서가 다를 수 있으며, (b,c)는 검사 대상이지만 이미 형성된 연결요소 안의 간선이어서 제외된다.

다시 나오면 이렇게 판단

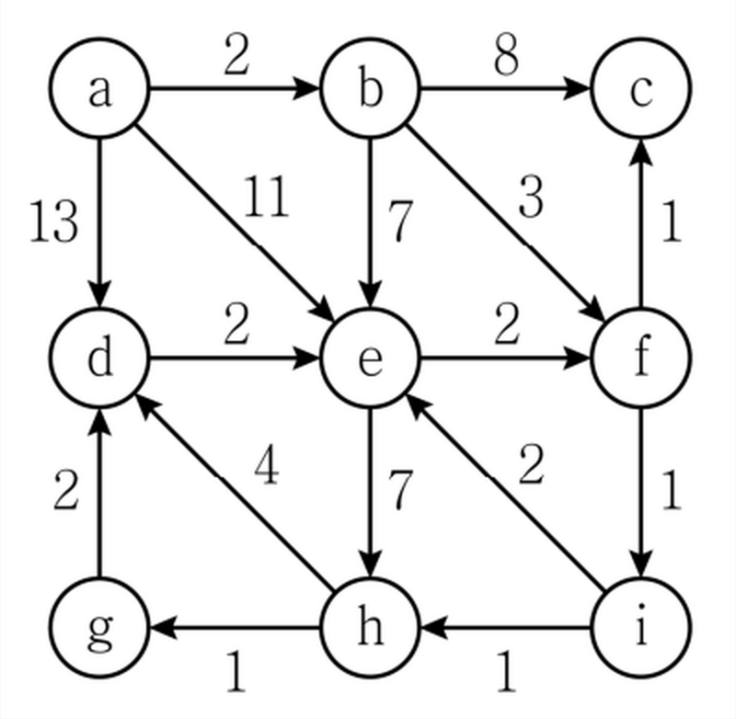
간선을 가중치 오름차순으로 정렬한 뒤, 각 간선의 양 끝 정점이 현재 서로 다른 연결요소에 있는지만 확인한다. 같은 연결요소이면 버리고, 다른 연결요소이면 MST에 포함되는 순서로 센다.

암기 포인트

크루스칼은 '가중치 오름차순, 사이클이면 제외, 포함된 간선만 순서 계산'으로 판단한다.

NoteBase

문 21. 다음 방향 그래프의 a 노드에서 각 노드별 최단 경로를 다익스트라 (Dijkstra) 알고리즘으로 계산할 때, 각 노드별 최단 거리 갱신 횟수로 옳은 것은? (단, a 노드를 제외한 모든 노드의 최단 경로 초기값은 ∞이다)



① b : 1회, c : 1회

② f : 2회, g : 1회

③ d : 2회, i : 2회

④ e : 3회, h : 1회

정답 4번

출제 포인트

이 문제는 방향 그래프에서 다익스트라 알고리즘을 수행할 때 각 노드의 잠정 최단 거리가 몇 번 감소 갱신되는지를 묻는 문제이다.

조건 정리

시작 노드는 a이고, a를 제외한 모든 노드의 초기 최단 거리는 무한대이다. 무한대에서 처음 유한한 값으로 바뀌는 것도 최단 거리 갱신 1회에 포함된다. 간선 방향을 따라 완화가 일어날 때 기존 거리보다 작아지는 경우만 갱신 횟수로 센다.

NoteBase

계산 과정

a 확정 후 b는 2로 1회, d는 13으로 1회, e는 11로 1회 갱신된다.
b 확정 후 c는 10으로 1회, e는 9로 2회, f는 5로 1회 갱신된다.
f 확정 후 c는 6으로 2회, i는 6으로 1회 갱신된다.
i 확정 후 h는 7로 1회, e는 8로 3회 갱신된다.
h 확정 후 g는 8로 1회, d는 11로 2회 갱신된다.
g 확정 후 d는 10으로 3회 갱신된다. 따라서 주요 갱신 횟수는 b 1회, c 2회, d 3회, e 3회, f 1회, g 1회, h 1회, i 1회이다.

선지별 해설

① X. b는 a에서 직접 2로 한 번 갱신되므로 1회가 맞지만, c는 b를 통해 10으로 한 번 갱신된 뒤 f를 통해 6으로 다시 갱신되므로 2회이다. 틀린 표현은 "c: 1회"이고, 올바르게는 "c: 2회"이다.
② X. g는 h를 통해 8로 처음 갱신되므로 1회가 맞지만, f는 b를 통해 5로 한 번만 갱신된다. e를 거친 경로는 10이어서 기존 5보다 크므로 갱신이 아니다. 틀린 표현은 "f: 2회"이고, 올바르게는 "f: 1회"이다.
③ X. d는 a를 통해 13, h를 통해 11, g를 통해 10으로 모두 세 번 감소 갱신된다. i는 f를 통해 6으로 한 번 갱신되고 이후 더 짧아지지 않는다. 틀린 표현은 "d: 2회, i: 2회"이고, 올바르게는 "d: 3회, i: 1회"이다.
④ O. e는 a를 통해 11, b를 통해 9, i를 통해 8로 세 번 갱신된다. h는 i를 통해 7로 한 번 갱신되고 이후 더 짧은 경로가 없으므로 1회이다.

오답 포인트

출제자가 노린 함정은 최종 최단 거리만 구하고 중간 갱신 횟수를 놓치는 데 있다. 특히 c는 최종 거리가 6이지만 10에서 6으로 한 번 더 줄어 2회이고, e는 11에서 9, 다시 8로 줄어 3회이다. 또한 d는 직접 간선 13에서 시작해 h 경유 11, g 경유 10으로 계속 줄어드는 점이 함정이다.

다시 나오면 이렇게 판단

다익스트라 갱신 횟수 문제는 확정된 노드에서 나가는 간선을 순서대로 완화하면서, 각 노드의 잠정 거리가 실제로 작아지는 순간만 표시해 센다. 최종 거리표만 보지 말고 거리값이 바뀐 이력을 함께 적어야 한다.

암기 포인트

다익스트라에서 갱신 횟수는 최종 거리의 개수가 아니라 잠정 거리가 감소한 횟수이다.

NoteBase

문 22. 하노이탑 문제를 해결하기 위한 다음 알고리즘의 빈칸 ㉠, ㉡에 들어갈 코드가 옳게 짝지어진 것은? (단, n은 원판의 개수이다)

```
hanoi(n, from, temp, to){
   if (n == 1)
      move(from, to); // from에서 to로 이동
   if (1 < n){
      hanoi(n - 1, [ ㉠ ]);
      move(from, to); // from에서 to로 이동
      hanoi(n - 1, [ ㉡ ]);
   }
}
```

① ㉠: from, to, temp, ㉡: from, temp, to

② ㉠: from, to, temp, ㉡: temp, from, to

③ ㉠: temp, from, to, ㉡: from, to, temp

④ ㉠: temp, from, to, ㉡: from, temp, to

정답 2번

출제 포인트

이 문제는 하노이탑 재귀 알고리즘에서 작은 원판 묶음을 어느 기둥에서 어느 기둥으로 옮기는지에 따라 재귀호출의 인자 순서가 달라짐을 묻는다.

NoteBase

올바른 순서

1. 첫 번째 재귀호출은 작은 원판 $n-1$개를 from에서 temp로 옮기는 단계이다. 이때 to가 임시 기둥이므로 인자는 from, to, temp가 된다.
2. 가운데 move(from, to)는 가장 큰 원판을 from에서 to로 옮기는 단계이다.
3. 두 번째 재귀호출은 temp에 옮겨 둔 작은 원판 $n-1$개를 to로 옮기는 단계이다. 이때 from이 임시 기둥이므로 인자는 temp, from, to가 된다.

왜 이 순서: 가장 큰 원판을 to로 옮기려면 먼저 작은 원판들이 temp에 모여 있어야 하고, 그 뒤에는 temp의 작은 원판들을 최종 목적지 to로 옮겨야 한다.

선지별 해설

① X: ㄱ의 from, to, temp는 맞지만, ㄴ의 from, temp, to가 틀리다. 두 번째 재귀호출은 작은 원판들을 temp에서 to로 옮겨야 하므로 올바르게는 temp, from, to이다.

② O: ㄱ은 작은 원판 $n-1$개를 from에서 temp로 옮기는 호출이므로 from, to, temp가 맞다. ㄴ은 temp에 둔 작은 원판 $n-1$개를 to로 옮기는 호출이므로 temp, from, to가 맞다.

③ X: ㄱ의 temp, from, to가 틀리다. 첫 번째 재귀호출은 시작 기둥 from의 작은 원판들을 보조 기둥 temp로 옮기는 단계이므로 올바르게는 from, to, temp이다. ㄴ의 from, to, temp도 틀리며, 올바르게는 temp, from, to이다.

④ X: ㄱ의 temp, from, to가 틀리고 ㄴ의 from, temp, to도 틀리다. 올바르게는 ㄱ이 from, to, temp이고 ㄴ이 temp, from, to이다.

오답 포인트

가운데 move(from, to)만 보고 두 재귀호출도 모두 from에서 to로 간다고 판단하면 틀린다. 첫 번째 재귀호출의 목적지는 temp이고, 두 번째 재귀호출의 출발지는 temp라는 점이 함정이다.

다시 나오면 이렇게 판단

하노이탑 재귀는 작은 원판들을 보조 기둥으로 이동, 가장 큰 원판을 목표 기둥으로 이동, 작은 원판들을 보조 기둥에서 목표 기둥으로 이동의 세 단계로 판단한다. 따라서 호출 인자는 첫 호출에서 출발 from, 임시 to, 목적 temp이고, 둘째 호출에서 출발 temp, 임시 from, 목적 to가 된다.

암기 포인트

하노이탑은 먼저 from에서 temp로 작은 원판을 비우고, 큰 원판을 from에서 to로 보낸 뒤, temp에서 to로 작은 원판을 덮는다.

NoteBase

문 23. 라빈-카프(Rabin-Karp) 알고리즘에 대한 설명으로 옳지 않은 것은? (단, 패턴 문자열의 길이는 m, 대상 문자열의 길이는 n, $m \leq n$이다)

① 패턴 문자열에 대한 해시 값 1개, 대상 문자열에 대한 해시 값이 최대 $n - m + 1$개 생성된다.

② 문자열의 해시 값을 계산하는 과정에서 호너의 방법(Horner's rule)을 사용할 수 있다.

③ 해시 값이 컴퓨터 레지스터의 용량을 초과할 경우, 나머지 연산(mod)을 사용한다.

④ 패턴 문자열의 해시 값과 대상 문자열의 일부 문자열에 대한 해시 값이 일치하면 알고리즘을 종료한다.

정답 4번

출제 포인트

이 문제는 라빈-카프 알고리즘에서 해시 값을 이용해 후보 위치를 빠르게 찾되, 해시 충돌 때문에 실제 문자열 비교가 필요하다는 점을 묻는다.

핵심 개념

라빈-카프 알고리즘은 길이 m인 패턴의 해시 값과 대상 문자열의 길이 m 부분 문자열 해시 값을 비교하는 문자열 검색 알고리즘이다. 대상 문자열 길이가 n이면 검사할 시작 위치는 $n - m + 1$개이며, 롤링 해시를 쓰면 다음 부분 문자열의 해시 값을 효율적으로 갱신할 수 있다. 다만 해시 값이 같아도 실제 문자열이 다를 수 있으므로 해시 일치는 후보 판정일 뿐 최종 일치 판정이 아니다.

선지별 해설

1번 O: 패턴 문자열 전체에 대한 해시 값은 1개가 계산된다. 대상 문자열에서는 길이 m인 부분 문자열을 시작 위치별로 보므로 가능한 창의 수가 $n - m + 1$개이고, 각 창에 대한 해시 값이 생성될 수 있다.
2번 O: 문자열 해시를 다항식 형태로 계산할 때 호너의 방법을 쓰면 곱셈과 덧셈을 순차적으로 적용하여 효율적으로 해시 값을 계산할 수 있다.
3번 O: 해시 계산 과정에서 값이 커질 수 있으므로 나머지 연산을 사용해 값을 일정 범위 안에 둔다. 이는 레지스터 용량 초과를 피하고 롤링 해시 계산을 안정적으로 수행하기 위한 처리이다.
4번 X: 틀린 부분은 "해시 값이 일치하면 알고리즘을 종료한다"이다. 해시 충돌이 가능하므로 패턴의 해시 값과 부분 문자열의 해시 값이 같으면 후보로만 보고, 실제 문자를 비교해 완전히 일치하는지 확인해야 한다.
올바르게는 해시 값이 일치하면 실제 문자열 비교를 수행하고, 그 결과가 일치할 때만 패턴 발견으로 판단한다.

오답 포인트

④의 함정은 해시 값 일치를 문자열 일치와 같은 것으로 처리한 점이다. 라빈-카프의 장점은 대부분의 위치를 해시 비교로 빠르게 거르는 데 있지만, 해시 충돌 가능성 때문에 최종 검증 단계가 사라지지는 않는다.

NoteBase

다시 나오면 이렇게 판단

라빈-카프 문항에서는 먼저 창의 개수 $n - m + 1$, 롤링 해시와 나머지 연산의 역할을 확인한다. 그다음 해시 값이 같은 경우를 최종 일치가 아니라 실제 비교가 필요한 후보 상태로 판단한다.

암기 포인트

라빈-카프는 해시로 후보를 찾고, 문자 비교로 확정한다.

NoteBase

문 24. KMP(Knuth-Morris-Pratt) 알고리즘에 대한 설명으로 옳지 않은 것은? (단, 패턴 문자열의 길이는 m, 대상 문자열의 길이는 n, $m \leq n$이다)

① 알고리즘의 전처리(preprocessing) 과정을 포함한 시간 복잡도는 $O(mn)$이다.

② 대상 문자열과 패턴 문자열의 문자들을 각각 비교하며 문자열 매칭 여부를 판단한다.

③ 매칭이 실패할 경우 미리 생성된 위치 정보를 이용하여 비교를 다시 시작한다.

④ 대상 문자열과 패턴 문자열에서 일치하는 부분 문자열을 이용하여 비교 횟수를 감소시킨다.

정답 1번

출제 포인트

이 문제는 KMP 알고리즘의 전처리와 탐색 과정이 각각 어떤 정보를 이용하며 전체 시간 복잡도가 어떻게 되는지를 묻는다.

핵심 개념

KMP는 패턴 문자열의 접두사와 접미사가 일치하는 길이 정보를 미리 계산해, 불일치가 발생해도 대상 문자열의 비교 위치를 되돌리지 않는 문자열 매칭 알고리즘이다.
패턴 길이가 m, 대상 문자열 길이가 n일 때 전처리는 $O(m)$, 탐색은 $O(n)$, 전체 시간 복잡도는 $O(m + n)$이다.
핵심 키워드는 실패 함수, LPS, 접두사와 접미사, 선형 시간이다.

선지별 해설

1번 X: 전처리 과정을 포함한 시간 복잡도를 $O(mn)$이라고 한 부분이 틀렸다. KMP는 패턴에 대한 위치 정보를 $O(m)$에 만들고 대상 문자열을 $O(n)$에 훑으므로 전체 시간 복잡도는 $O(m + n)$이다. 틀린 표현 $O(mn)$은 올바르게는 $O(m + n)$이다.
2번 O: KMP도 문자열 매칭 알고리즘이므로 대상 문자열의 현재 문자와 패턴 문자열의 현재 문자를 비교하면서 매치 여부를 판단한다. 차이는 불일치 후 이동 방법에 있다.
3번 O: 매칭이 실패하면 전처리로 만든 실패 함수 또는 LPS 정보를 이용해 패턴의 비교 위치를 적절히 옮긴다. 정확히는 대상 문자열의 인덱스를 처음으로 되돌리지 않고 패턴 인덱스를 조정해 비교를 이어 간다.
4번 O: KMP는 이미 일치한 부분 문자열의 접두사와 접미사 일치 정보를 이용해 불필요한 재비교를 줄인다. 이 정보가 비교 횟수를 선형 수준으로 줄이는 핵심이다.

오답 포인트

출제자가 노린 함정은 1번의 $O(mn)$이다. 단순 비교 방식의 최악 시간 복잡도와 KMP의 선형 시간 복잡도를 혼동하면 전처리를 포함해도 곱셈형 복잡도가 된다고 착각할 수 있다.

NoteBase

다시 나오면 이렇게 판단

KMP 문제에서는 먼저 실패 함수 또는 LPS를 이용하는지 확인하고, 패턴 전처리는 $O(m)$, 대상 문자열 탐색은 $O(n)$, 전체는 $O(m + n)$으로 판단한다.

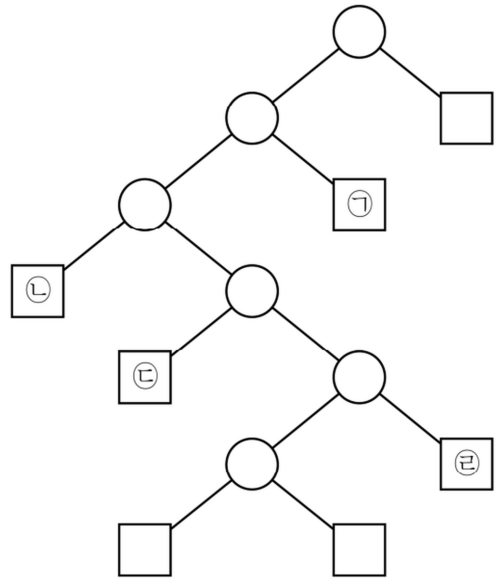
암기 포인트

KMP는 실패 함수로 되돌아가지 않고, 시간 복잡도는 전처리 포함 $O(m + n)$이다.

NoteBase

문 25. 다음 문자와 빈도수에 대해 허프만(Huffman) 알고리즘을 이용하여 트리를 구성했을 때, 빈칸 ㉠ ~ ㉣에 들어갈 문자는? (단, ○는 빈도수 노드, □는 문자 노드이다)

문자	a	b	d	e	g	i	k
빈도 수	50	4	2	8	1	25	6



① ㉠: a, ㉡: i, ㉢: b, ㉣: k

② ㉠: a, ㉡: i, ㉢: k, ㉣: b

③ ㉠: i, ㉡: e, ㉢: d, ㉣: g

④ ㉠: i, ㉡: e, ㉢: k, ㉣: b

정답 4번

출제 포인트

이 문제는 허프만 알고리즘에서 가장 작은 빈도수 두 노드를 반복적으로 결합해 만든 트리 구조를 보고 각 빈칸의 문자를 판단하는 문제이다.

조건 정리

허프만 결합 순서는 빈도 1의 g와 빈도 2의 d를 합쳐 3을 만들고, 3과 b의 4를 합쳐 7을 만든다. 이어서 k의 6과 7을 합쳐 13, e의 8과 13을 합쳐 21, 21과 i의 25를 합쳐 46, 마지막으로 46과 a의 50을 합쳐 루트가 된다.

NoteBase

선지별 해설

① X: ㄱ을 a로 본 점이 틀린다. a는 빈도 50으로 가장 큰 문자이므로 마지막 결합에서 루트의 반대쪽 단독 문자 노드가 된다. ㄱ은 빈도 25의 i, ㄴ은 e, ㄷ은 k, ㄹ은 b가 되어야 한다.
② X: ㄱ을 a로 본 점과 ㄴ을 i로 본 점이 틀린다. i는 빈도 25로 루트 왼쪽 부분트리의 단독 문자 노드인 ㄱ에 해당하고, a는 루트 오른쪽의 단독 문자 노드에 해당한다. 올바르게는 ㄱ=i, ㄴ=e, ㄷ=k, ㄹ=b이다.
③ X: ㄱ=i와 ㄴ=e는 맞지만, ㄷ=d와 ㄹ=g가 틀린다. 빈도 13 노드는 k의 6과 빈도 7 노드가 결합한 것이므로 ㄷ은 k이고, 빈도 7 노드는 빈도 3 노드와 b의 4가 결합한 것이므로 ㄹ은 b이다.
④ O: ㄱ=i, ㄴ=e, ㄷ=k, ㄹ=b가 모두 트리의 결합 순서와 일치한다. 루트 왼쪽의 빈도 46 노드는 빈도 21 노드와 i의 25가 결합한 것이므로 ㄱ은 i이고, 빈도 21 노드는 e의 8과 빈도 13 노드가 결합한 것이므로 ㄴ은 e이다. 빈도 13 노드의 단독 문자 노드는 k, 빈도 7 노드의 단독 문자 노드는 b이므로 ㄷ은 k, ㄹ은 b이다.

오답 포인트

가장 큰 빈도수 a를 트리의 깊은 곳에 넣거나, 빈도 7 노드와 빈도 13 노드의 구성 문자를 뒤바꾸면 오답이 된다. 허프만 트리에서는 빈도가 큰 문자가 루트에 더 가깝고, 빈도가 작은 문자들이 더 깊은 곳에서 먼저 결합된다.

다시 나오면 이렇게 판단

빈도수를 오름차순으로 놓고 가장 작은 두 값을 합친 뒤, 새로 만든 빈도수 노드를 다시 목록에 넣어 같은 과정을 반복한다. 완성된 결합 순서의 합계 노드가 도식의 내부 노드 위치와 대응되는지를 차례로 확인한다.

암기 포인트

허프만 트리는 작은 빈도 둘을 먼저 묶고, 큰 빈도 문자는 루트에 가까운 짧은 경로에 놓인다.

NoteBase