문제 01 · 객관식

알고리즘이 갖추어야 할 다섯 가지 조건 중 '입력(input)'에 대한 설명으로 옳은 것은?

① 적어도 1개는 주어져야 한다.

② 입력이 없는 절차는 절대 알고리즘이 될 수 없다.

③ 출력의 개수와 같은 수만큼 주어져야 한다.

④ 사용자가 키보드로 직접 입력한 값이어야 한다.

⑤ 0개 이상이면 되므로 반드시 있어야 하는 것은 아니다.

알고리즘이 갖추어야 할 다섯 가지 조건 중 '입력(input)'에 대한 설명으로 옳은 것은?

정답 ⑤ **0개 이상이면 되므로 반드시 있어야 하는 것은 아니다.**

해설

입력은 0개 이상, 출력은 1개 이상이다. "1부터 100까지 더해서 출력하라"처럼 외부 입력이 전혀 없는 알고리즘을 하나라도 떠올릴 수 있으면 ①②③④는 모두 무너진다. 입력과 출력의 조건이 서로 반대라는 점을 묶어서 기억하면 헷갈리지 않는다.

다음 중 알고리즘이라고 볼 수 없는 것은?

① 종료 조건 없이 같은 명령을 계속 반복하는 절차

② 1부터 100까지 더한 값을 출력하는 절차

③ 두 수의 최대공약수를 구해 반환하는 절차

④ 리스트에서 최대값을 찾아 반환하는 절차

⑤ 입력 없이 "Hello"를 한 번 출력하는 절차

정답 및 해설 02

다음 중 알고리즘이라고 볼 수 없는 것은?

정답 ① **종료 조건 없이 같은 명령을 계속 반복하는 절차**

해설

유한성 위반이다. 알고리즘은 유한한 명령을 실행한 뒤 반드시 종료되어야 한다. 참고로 "입력 없이 Hello를 한 번 출력하는 절차"는 입력이 0개지만 출력이 1개 있으므로 알고리즘의 조건을 모두 만족한다.

문제 03 · 객관식

요리 절차에 "소금을 적당히 넣는다"라고 적혀 있다. 이 명령이 위반하는 알고리즘의 조건은?

① 입력

② 출력

③ 유효성

④ 유한성

⑤ 명백성

요리 절차에 "소금을 적당히 넣는다"라고 적혀 있다. 이 명령이 위반하는 알고리즘의 조건은?

정답 ⑤ **명백성**

해설

명백성(clarity) 위반이다. '적당히'는 사람마다 다르게 해석되므로 모호하다. "소금 3g을 넣는다"처럼 해석의 여지가 없어야 한다.

문제 04 · 객관식

어떤 절차에 "x를 0으로 나눈 값을 y에 저장한다"라는 명령이 있다. 이 명령이 위반하는 조건은?

① 입력

② 유효성

③ 명백성

④ 유한성

⑤ 출력

어떤 절차에 "x를 0으로 나눈 값을 y에 저장한다"라는 명령이 있다. 이 명령이 위반하는 조건은?

정답 ② **유효성**

해설

유효성(effectiveness) 위반이다. 각 명령은 실제로 수행 가능한 기본 연산이어야 하는데, 0으로 나누기는 수행할 수 없다. 명령이 모호한 것이 아니라(명백성은 만족) 아예 실행이 불가능하다는 점에서 명백성과 구별된다.

다음 코드를 S = [9, 3, 2, 8, 10, 1] 에 대해 linear_search(S, 8)로 실행했을 때 반환되는 값은?

code.py

```python
def linear_search(S, target):
    for i in range(len(S)):
        if S[i] == target:
            return i
    return -1
```

① 3

② 4

③ 8

④ -1

⑤ 0

정답 및 해설 05

다음 코드를 S = [9, 3, 2, 8, 10, 1] 에 대해 linear_search(S, 8)로 실행했을 때 반환되는 값은?

code.py

```
def linear_search(S, target):
    for i in range(len(S)):
        if S[i] == target:
            return i
    return -1
```

정답 ① 3

해설

8은 사람이 세면 네 번째지만 인덱스는 0부터 시작하므로 3이다. 코드가 반환하는 것은 '몇 번째'가 아니라 '인덱스'라는 점에 주의해야 한다. 시험에서 가장 많이 나오는 실수 유형이다.

알고리즘의 역사에 대한 설명으로 옳지 않은 것은?

① 유클리드는 기원전 300년경 최대공약수를 구하는 방법을 고안했다.

② 에라토스테네스의 체는 소수를 찾는 방법이다.

③ 알고리즘은 컴퓨터가 발명된 이후에 등장한 개념이다.

④ algorithm이라는 단어는 수학자 알콰리즈미의 이름에서 유래했다.

⑤ 유클리드 알고리즘은 오늘날에도 그대로 사용된다.

알고리즘의 역사에 대한 설명으로 옳지 않은 것은?

정답 ③ **알고리즘은 컴퓨터가 발명된 이후에 등장한 개념이다.**

해설

알고리즘은 컴퓨터보다 훨씬 오래되었다. 컴퓨터는 1940년대에 등장했지만 유클리드 알고리즘은 기원전 300년경의 것이다. 문제 해결 절차가 먼저 있었고, 컴퓨터는 그것을 대신 실행해 주는 도구로 나중에 등장했다.

유사코드(pseudocode)의 특징으로 옳지 않은 것은?

① 작성한 그대로 컴파일하여 실행한다.

② 자연어에 비해 논리의 흐름이 명확하게 드러난다.

③ 특정 프로그래밍 언어의 문법에 구애받지 않는다.

④ 대입을 나타낼 때 ← 기호를 쓰기도 한다.

⑤ 논문에서 알고리즘을 설명할 때 널리 쓰인다.

유사코드(pseudocode)의 특징으로 옳지 않은 것은?

정답 ① **작성한 그대로 컴파일하여 실행한다.**

해설

유사코드는 실행을 목적으로 하지 않는다. 실행되지 않기 때문에 문법의 제약에서 자유롭고, 그 덕분에 어떤 언어를 쓰는 독자에게도 논리를 전달할 수 있다. 대입에 =가 아니라 ←를 쓰는 것도 '같다'와 '넣는다'의 혼동을 피하기 위해서다.

문제 08 · 객관식

순서도(flowchart)에서 조건 판단을 나타내는 도형은?

① 타원

② 직사각형

③ 원

④ 평행사변형

⑤ 마름모

순서도(flowchart)에서 조건 판단을 나타내는 도형은?

정답 ⑤ **마름모**

해설

타원은 시작/끝, 직사각형은 처리, 마름모는 조건 판단, 평행사변형은 입출력이다. 마름모에서만 흐름이 둘로 갈라진다.

자연어로 알고리즘을 기술할 때의 가장 큰 단점은?

① 작성하는 데 시간이 오래 걸린다는 점이 가장 큰 문제다.

② 비전공자는 내용을 전혀 이해할 수 없다는 점이 문제다.

③ 표현이 모호하여 읽는 사람마다 다르게 해석한다는 점이 문제다.

④ 반복이나 조건 같은 논리 구조를 절대 표현할 수 없다는 점이 문제다.

⑤ 종이에 옮겨 적을 수 없다는 점이 가장 큰 문제다.

자연어로 알고리즘을 기술할 때의 가장 큰 단점은?

정답 ③ **표현이 모호하여 읽는 사람마다 다르게 해석한다는 점이 문제다.**

해설

자연어는 쓰기 쉽고 누구나 읽을 수 있다는 장점이 있지만, "하나씩 조사한다"가 앞에서부터인지 뒤에서부터인지 알 수 없는 것처럼 해석이 갈린다. 사람은 알아서 좋게 해석하지만 컴퓨터는 그러지 못한다. 나머지 넷은 자연어의 장점을 단점으로 뒤집어 놓은 것이거나 사실이 아니다.

문제 10 · 객관식

다음 알고리즘 기술 방법을 '표현이 엄밀해지고 실제 구현에 가까워지는 순서'로 바르게 나열한 것은?

ㄱ. 순서도　　ㄴ. 자연어　　ㄷ. 프로그래밍 언어　　ㄹ. 유사코드

① ㄱ - ㄴ - ㄹ - ㄷ

② ㄴ - ㄱ - ㄹ - ㄷ

③ ㄴ - ㄹ - ㄱ - ㄷ

④ ㄹ - ㄴ - ㄱ - ㄷ

⑤ ㄴ - ㄱ - ㄷ - ㄹ

다음 알고리즘 기술 방법을 '표현이 엄밀해지고 실제 구현에 가까워지는 순서'로 바르게 나열한 것은?

정답 ② ㄴ - ㄱ - ㄹ - ㄷ

해설

자연어 → 순서도 → 유사코드 → 프로그래밍 언어 순이다. 아래로 갈수록 엄밀해지고 기계에 가까워지지만, 대신 읽기는 불편해지고 언어에 종속된다. 어느 하나가 항상 옳은 것이 아니라 목적에 따라 골라 쓴다.

다음 코드를 gcd_brute_force(1000000, 999999)로 실행할 때 while 반복 횟수에 가장 가까운 것은?

code.py

```python
def gcd_brute_force(a, b):
    t = min(a, b)
    while t > 0:
        if a % t == 0 and b % t == 0:
            return t
        t -= 1
    return 1
```

① 2회

② 20회

③ 1,000회

④ 999,999회

⑤ 10억 회

다음 코드를 gcd_brute_force(1000000, 999999)로 실행할 때 while 반복 횟수에 가장 가까운 것은?

code.py

```python
def gcd_brute_force(a, b):
    t = min(a, b)
    while t > 0:
        if a % t == 0 and b % t == 0:
            return t
        t -= 1
    return 1
```

정답 ④ **999,999회**

해설

연속한 두 자연수는 항상 서로소여서 최대공약수가 1이다. 따라서 t를 999,999부터 1까지 하나씩 줄여가며 전부 검사한 뒤에야 답을 찾는다. 답이 작을수록 오래 걸린다는 점이 완전 탐색의 특징이다.

유클리드 알고리즘의 핵심 관계식으로 옳은 것은? (단, mod는 나머지 연산)

① gcd(a, b) = gcd(a × b, b)

② gcd(a, b) = gcd(b, a ÷ b)

③ gcd(a, b) = gcd(a mod b, a + b)

④ gcd(a, b) = gcd(a + b, a − b)

⑤ gcd(a, b) = gcd(b, a mod b)

유클리드 알고리즘의 핵심 관계식으로 옳은 것은? (단, mod는 나머지 연산)

정답 ⑤ **gcd(a, b) = gcd(b, a mod b)**

해설

큰 수를 작은 수로 나눈 나머지와 작은 수의 최대공약수는 원래 두 수의 최대공약수와 같다. a와 b가 모두 d의 배수이면 a mod b도 d의 배수가 되므로 공약수의 집합이 그대로 보존되기 때문이다.

문제 13 · 객관식

유클리드 알고리즘으로 gcd(48, 18)을 구할 때, 수행되는 나머지 연산의 횟수와 최종 결과를 바르게 짝지은 것은?

① 2회, 6

② 4회, 6

③ 3회, 12

④ 3회, 6

⑤ 2회, 12

유클리드 알고리즘으로 gcd(48, 18)을 구할 때, 수행되는 나머지 연산의 횟수와 최종 결과를 바르게 짝지은 것은?

정답 ④ **3회, 6**

해설

48 mod 18 = 12 → 18 mod 12 = 6 → 12 mod 6 = 0 이므로 나머지 연산 3회 후 종료되고, 나머지가 0이 되는 순간의 나누는 수인 6이 답이다. 완전 탐색이었다면 18부터 6까지 13번을 검사해야 한다.

문제 14 · 객관식

완전 탐색 방식과 유클리드 알고리즘의 비교로 옳지 않은 것은?

① 컴퓨터 성능을 10배 올리면 두 방법의 속도는 비슷해진다.

② 두 코드의 길이는 열 줄 안팎으로 크게 다르지 않다.

③ 완전 탐색은 최대공약수가 작을수록 오래 걸린다.

④ 유클리드는 문제를 더 작은 문제로 계속 바꾸어 나간다.

⑤ 두 방법 모두 항상 정확한 최대공약수를 낸다.

완전 탐색 방식과 유클리드 알고리즘의 비교로 옳지 않은 것은?

정답 ① **컴퓨터 성능을 10배 올리면 두 방법의 속도는 비슷해진다.**

해설

약 100만 배 차이를 하드웨어 10배로 메울 수 없다. 하드웨어 개선은 상수배 효과에 그치지만 알고리즘 개선은 증가의 '차수' 자체를 바꾼다. 코드가 짧다고 빠른 것이 아니며, 두 코드 모두 정확하다는 점(버그가 없는데도 쓸 수 없는 코드)도 중요한 포인트다.

이진 탐색을 사용하기 위한 전제 조건은?

① 리스트에 저장된 값이 모두 양의 정수여야 한다.

② 데이터의 개수가 반드시 2의 거듭제곱이어야 한다.

③ 데이터에 같은 값이 두 번 이상 나타나지 않아야 한다.

④ 데이터가 오름차순이나 내림차순으로 반드시 정렬되어 있어야 한다.

⑤ 데이터가 배열이 아닌 연결 리스트로 저장되어 있어야 한다.

이진 탐색을 사용하기 위한 전제 조건은?

정답 ④ **데이터가 오름차순이나 내림차순으로 반드시 정렬되어 있어야 한다.**

해설

가운데 값을 보고 절반을 버릴 수 있는 근거가 바로 정렬이다. 정렬되어 있지 않으면 가운데보다 큰 값이 반드시 오른쪽에 있다는 보장이 없어 아무것도 버릴 수 없다. 나머지 넷은 그 조건이 깨져도 이진 탐색이 정상 동작하므로 전제 조건이 아니다.

정렬된 데이터 1,000,000개에서 이진 탐색이 수행하는 최대 비교 횟수에 가장 가까운 것은?

① 10회

② 20회

③ 30회

④ 1,000회

⑤ 1,000,000회

정렬된 데이터 1,000,000개에서 이진 탐색이 수행하는 최대 비교 횟수에 가장 가까운 것은?

정답 ② **20회**

해설

약 $\log_2(1{,}000{,}000) \approx 19.9$이므로 최대 20회다. 십억 개여도 30회다. 백만이면 20, 십억이면 30이라는 두 수치는 외워 두면 감을 잡기 좋다.

정렬된 원소가 1,000,000개다. 순차 탐색은 최대 1,000,000회, 이진 탐색은 약 20회 비교한다. 순차 탐색의 비교 횟수는 이진 탐색의 약 몇 배인가?

① 5배

② 20배

③ 1,000배

④ 50,000배

⑤ 1,000,000배

정렬된 원소가 1,000,000개다. 순차 탐색은 최대 1,000,000회, 이진 탐색은 약 20회 비교한다. 순차 탐색의 비교 횟수는 이진 탐색의 약 몇 배인가?

정답 ④ **50,000배**

해설

1,000,000÷20=50,000이다. 같은 입력에서도 탐색 방법에 따라 비교 횟수가 크게 달라진다.

다음 이진 탐색 코드에서 low = mid + 1 대신 low = mid로 작성했을 때 발생할 수 있는 문제는?

code.py

```python
def binary_search(S, x):
    low = 0
    high = len(S) - 1
    while low <= high:
        mid = (low + high) // 2
        if S[mid] == x:
            return mid
        elif S[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

① 구문 오류가 발생하여 컴파일 단계에서 실패할 수 있다.

② 탐색 범위가 줄지 않아 무한 루프에 빠질 수 있다.

③ 값이 리스트에 있어도 -1을 반환할 수 있다.

④ 리스트의 정렬 상태가 깨져 결과가 달라질 수 있다.

⑤ 재귀 호출이 늘어나 메모리 사용량이 두 배가 될 수 있다.

정답 및 해설 18

다음 이진 탐색 코드에서 low = mid + 1 대신 low = mid로 작성했을 때 발생할 수 있는 문제는?

code.py

```
def binary_search(S, x):
    low = 0
    high = len(S) - 1
    while low <= high:
        mid = (low + high) // 2
        if S[mid] == x:
            return mid
        elif S[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

정답 ② **탐색 범위가 줄지 않아 무한 루프에 빠질 수 있다.**

해설

mid는 이미 확인해서 답이 아님을 알고 있으므로 반드시 건너뛰어야 한다. low = mid로 두면 low와 high가 인접했을 때 mid가 계속 같은 값으로 계산되어 범위가 줄지 않는다.

문제 19 · 객관식

"컴퓨터가 빨라졌으니 어떤 알고리즘을 쓰든 상관없다"는 주장에 대한 반박으로 적절하지 않은 것은?

① 다루어야 할 데이터의 크기도 함께 커졌기 때문이다.

② 하드웨어 개선은 상수배 효과지만 알고리즘은 차수를 바꾼다.

③ 하드웨어를 10배 빠르게 하면 $O(n^2)$이 $O(n)$으로 바뀐다.

④ 비효율은 n이 커질수록 급격하게 벌어지기 때문이다.

⑤ 데이터가 클수록 알고리즘의 차이가 더 크게 드러난다.

"컴퓨터가 빨라졌으니 어떤 알고리즘을 쓰든 상관없다"는 주장에 대한 반박으로 적절하지 않은 것은?

정답 ③ **하드웨어를 10배 빠르게 하면 $O(n^2)$이 $O(n)$으로 바뀐다.**

해설

하드웨어를 아무리 빠르게 해도 복잡도의 차수는 바뀌지 않는다. $O(n^2)$은 여전히 $O(n^2)$이고 상수만 작아질 뿐이다. 나머지는 모두 타당한 반박이다.

문제 20 · 객관식

시간 복잡도 분석에서 실제 실행 시간(초)을 측정하지 않는 이유로 가장 적절한 것은?

① 컴퓨터와 언어에 따라 달라져 알고리즘 고유의 성질이 아니기 때문이다.

② 실행 시간을 정확히 측정할 도구가 존재하지 않기 때문이다.

③ 초 단위 값은 자릿수가 너무 커서 서로 비교하기 어렵기 때문이다.

④ 같은 알고리즘의 실행 시간은 환경과 무관하게 항상 일정하기 때문이다.

⑤ 실행 시간보다 메모리 사용량이 훨씬 더 중요한 지표이기 때문이다.

시간 복잡도 분석에서 실제 실행 시간(초)을 측정하지 않는 이유로 가장 적절한 것은?

정답 ① **컴퓨터와 언어에 따라 달라져 알고리즘 고유의 성질이 아니기 때문이다**.

해설

같은 알고리즘도 노트북과 데스크탑에서 다르고 파이썬과 C에서 다르다. 반면 기본 연산의 '횟수'는 어디서 돌려도 같으므로 알고리즘 자체를 비교할 수 있는 기준이 된다. "시간을 재지 않고 횟수를 센다"가 핵심이다.

문제 21 · 객관식

기본 연산(basic operation)에 대한 설명으로 옳지 않은 것은?

① 알고리즘의 핵심 동작 중 반드시 하나를 대표로 고른다.

② 보통 가장 안쪽 반복문에서 자주 실행되는 명령을 고른다.

③ 유일하게 정해지므로 다르게 고르면 틀린 답이 된다.

④ 정렬에서는 비교와 교환 중 어느 쪽을 골라도 근거만 있으면 된다.

⑤ 탐색 알고리즘에서는 보통 비교 연산을 기본 연산으로 삼는다.

기본 연산(basic operation)에 대한 설명으로 옳지 않은 것은?

정답 ③ **유일하게 정해지므로 다르게 고르면 틀린 답이 된다.**

해설

정렬에서 비교를 셀 수도 있고 교환을 셀 수도 있듯이 기본 연산의 선택에는 정답이 하나로 정해져 있지 않다. 중요한 것은 무엇을 골랐는지가 아니라 왜 그것을 골랐는지에 대한 근거다.

다음 코드를 길이가 n인 리스트에 대해 실행할 때 수행되는 비교 연산의 횟수는?

code.py

```
def get_largest(A):
    largest = A[0]
    for i in range(1, len(A)):
        if A[i] > largest:
            largest = A[i]
    return largest
```

① n

② n / 2

③ n + 1

④ n − 1

⑤ n(n−1)/2

정답 및 해설 22

다음 코드를 길이가 n인 리스트에 대해 실행할 때 수행되는 비교 연산의 횟수는?

code.py

```
def get_largest(A):
    largest = A[0]
    for i in range(1, len(A)):
        if A[i] > largest:
            largest = A[i]
    return largest
```

정답 ④ $n - 1$

해설

0번 원소는 이미 largest에 넣었으므로 다시 비교하지 않는다. i는 1부터 n−1까지 돌아 총 n−1회다. 또한 이 알고리즘은 최선·평균·최악이 모두 n−1로 같은데, 최대값을 찾으려면 모든 원소를 반드시 한 번씩 봐야 하므로 조기 종료가 불가능하기 때문이다.

문제 23 · 객관식

target이 리스트에 반드시 존재하고 각 위치에 있을 확률이 모두 1/n로 같을 때, 순차 탐색의 평균 비교 횟수는?

① $n / 2$

② $(n + 1) / 2$

③ $(n - 1) / 2$

④ n

⑤ $\log_2 n$

target이 리스트에 반드시 존재하고 각 위치에 있을 확률이 모두 1/n로 같을 때, 순차 탐색의 평균 비교 횟수는?

정답 ② **(n + 1) / 2**

해설

(1 + 2 + … + n)/n = {n(n+1)/2}/n = (n+1)/2 이다. n이 충분히 크면 대략 n/2이지만 정확한 값은 (n+1)/2다. 직관으로 말한 n/2와 계산으로 얻은 (n+1)/2의 차이를 구분할 수 있어야 한다.

최선·평균·최악의 경우에 대한 설명으로 옳지 않은 것은?

① 최악의 경우는 성능의 상한을 보장하기 위해 주로 쓰인다.

② 최선의 경우는 실무에서 거의 사용되지 않는 기준이다.

③ 평균의 경우를 분석하려면 입력 분포에 대한 가정이 반드시 필요하다.

④ 최대값 찾기는 최선과 최악의 비교 횟수가 항상 n−1로 같다.

⑤ 순차 탐색에서 target이 없는 경우는 최선의 경우에 해당한다.

최선·평균·최악의 경우에 대한 설명으로 옳지 않은 것은?

정답 ⑤ **순차 탐색에서 target이 없는 경우는 최선의 경우에 해당한다.**

해설

존재하지 않으면 끝까지 다 확인해야 하므로 최악의 경우다. '없다'는 결론도 n번 비교해야 나오는 결론이다. 값이 맨 뒤에 있는 경우와 아예 없는 경우는 비교 횟수가 n으로 같다.

다음 코드에서 기본 연산을 덧셈으로 잡을 때 시간 복잡도 함수 T(n)은? (n = len(A))

code.py

```
def add_array_members(A):
    total = 0
    for x in A:
        total = total + x
    return total
```

① $T(n) = 1$

② $T(n) = \log n$

③ $T(n) = n$

④ $T(n) = n - 1$

⑤ $T(n) = n^2$

다음 코드에서 기본 연산을 덧셈으로 잡을 때 시간 복잡도 함수 T(n)은? (n = len(A))

code.py

```
def add_array_members(A):
    total = 0
    for x in A:
        total = total + x
    return total
```

정답 ③ **T(n) = n**

해설

반복문이 원소 하나당 한 번씩 정확히 n번 실행되므로 T(n) = n이다. 참고로 get_largest의 T(n) = n−1과는 다르지만, 빅오로 표기하면 둘 다 O(n)으로 같아진다.

문제 26 · 객관식

$T(n)=3n^2+5n+100$에서 최고차항만 남기고 계수를 생략한 빅오 표기는?

① $O(n^2)$

② $O(n)$

③ $O(1)$

④ $O(n^3)$

⑤ $O(2^n)$

T(n)=3n²+5n+100에서 최고차항만 남기고 계수를 생략한 빅오 표기는?

정답 ① $O(n^2)$

해설

최고차항은 $3n^2$이므로 가장 타이트한 상한은 $O(n^2)$이다. $O(n^3)$, $O(2^n)$도 상한이지만 필요 이상으로 크다.

문제 27 · 객관식

$2n + 1 = O(n)$ 임을 보이기 위한 (c, n_0)의 쌍으로 적절하지 않은 것은?

① $c = 3, n_0 = 1$

② $c = 2, n_0 = 1$

③ $c = 5, n_0 = 1$

④ $c = 10, n_0 = 5$

⑤ $c = 4, n_0 = 2$

$2n + 1 = O(n)$ 임을 보이기 위한 (c, n_0)의 쌍으로 적절하지 않은 것은?

정답 ② $c = 2, n_0 = 1$

해설

$c = 2$이면 $2n + 1 \le 2n$ 이어야 하는데 이는 어떤 n에서도 성립하지 않는다. 나머지는 모두 성립한다. c와 n_0는 하나만 존재하면 되므로 정답이 여러 개일 수 있다는 점도 함께 기억하자.

문제 28 · 객관식

다음 복잡도를 증가 속도가 느린 것부터 빠른 순으로 바르게 나열한 것은?

① $O(1) < O(n) < O(\log n) < O(n \log n) < O(n^2)$

② $O(1) < O(n) < O(n \log n) < O(\log n) < O(n^2)$

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

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

⑤ $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2)$

다음 복잡도를 증가 속도가 느린 것부터 빠른 순으로 바르게 나열한 것은?

정답 ⑤ $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2)$

해설

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$ 순이다. $\log n$은 n보다 느리게 증가하고, $n \log n$은 n과 n^2 사이에 있다.

다음 코드를 길이가 n인 리스트에 대해 실행할 때의 총 비교 횟수와 빅오 표기를 바르게 짝지은 것은?

code.py

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

① n, O(n)

② n^2, $O(n^2)$

③ $n(n-1)/2$, $O(n^2)$

④ $n(n+1)/2$, $O(n)$

⑤ $\log_2 n$ 회, $O(\log n)$

다음 코드를 길이가 n인 리스트에 대해 실행할 때의 총 비교 횟수와 빅오 표기를 바르게 짝지은 것은?

code.py

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

정답 ③ **$n(n-1)/2$, $O(n^2)$**

해설

$(n-1) + (n-2) + \dots + 1 = n(n-1)/2$ 이고, 전개하면 $(n^2 - n)/2$ 이므로 최고차항만 남겨 $O(n^2)$이다. 다만 이중 루프라고 해서 항상 $O(n^2)$인 것은 아니며, 안쪽 반복이 n에 비례할 때만 그렇다.

빅오와 최악의 경우에 대한 설명으로 옳은 것은?

① 빅오는 언제나 최악의 경우를 가리킨다.

② Ω는 최선의 경우와 같은 뜻으로 쓸 수 있다.

③ 최악의 경우는 Θ로 표기하는 것이 원칙이다.

④ 평균의 경우에도 빅오 표기를 쓴다.

⑤ Θ가 성립하면 O는 성립하지 않는다.

빅오와 최악의 경우에 대한 설명으로 옳은 것은?

정답 ④ **평균의 경우에도 빅오 표기를 쓴다.**

해설

최선·평균·최악은 '어떤 입력을 볼 것인가'라는 시나리오이고, O·Ω·Θ는 '증가율을 어떻게 표기할 것인가'라는 수학적 도구다. 서로 다른 축이므로 조합해서 쓴다. 퀵 정렬이 최악의 경우 $O(n^2)$, 평균의 경우 $O(n \log n)$인 것이 대표적인 예다. 또한 Θ가 성립하면 O와 Ω는 자동으로 성립한다.

A컵에는 물이, B컵에는 주스가 들어 있다. 두 컵의 내용물을 서로 바꾸려고 할 때 빈 컵(temp)이 하나 더 필요한 이유를 한 문장으로 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

A컵에는 물이, B컵에는 주스가 들어 있다. 두 컵의 내용물을 서로 바꾸려고 할 때 빈 컵(temp)이 하나 더 필요한 이유를 한 문장으로 쓰시오.

정답 **한쪽을 먼저 옮기면 원래 들어 있던 값이 덮어써져 사라지기 때문이다.**

해설

a = b를 먼저 하면 a의 원래 값이 사라진다. 잃기 전에 다른 곳에 보관해 두어야 한다.

알고리즘이 갖추어야 할 다섯 가지 조건을 모두 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

알고리즘이 갖추어야 할 다섯 가지 조건을 모두 쓰시오.

정답 **입력, 출력, 명백성, 유한성, 유효성**

해설

입력은 0개 이상, 출력은 1개 이상이라는 점을 짝지어 기억하면 좋다.

정렬 문제를 '증가 순서'가 아니라 '비감소 순서'라고 표현하는 이유를 한 문장으로 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

정렬 문제를 '증가 순서'가 아니라 '비감소 순서'라고 표현하는 이유를 한 문장으로 쓰시오.

정답 **같은 값이 이어질 수 있어 '증가'라고 하면 그 경우가 규칙에 어긋나기 때문이다.**

해설

3 다음에 3이 오면 커진 것이 아니다. 그래서 '작아지지만 않으면 된다'로 표현한다.

순차 탐색이 값을 찾지 못했을 때 −1을 반환하도록 만든 이유를 한 문장으로 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

순차 탐색이 값을 찾지 못했을 때 −1을 반환하도록 만든 이유를 한 문장으로 쓰시오.

정답 **인덱스는 0 이상이므로 −1은 정상적인 위치 값과 겹치지 않아 '없음'을 나타낼 수 있기 때문이다.**

순서도에서 ① 시작과 끝, ② 처리, ③ 입출력을 나타내는 도형을 각각 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

순서도에서 ① 시작과 끝, ② 처리, ③ 입출력을 나타내는 도형을 각각 쓰시오.

정답 ① 타원 ② 직사각형 ③ 평행사변형

해설

조건 판단은 마름모이며, 흐름이 둘로 갈라지는 곳은 마름모뿐이다.

유사코드에서 대입을 나타낼 때 = 대신 ← 기호를 쓰는 이유를 한 문장으로 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

유사코드에서 대입을 나타낼 때 = 대신 ← 기호를 쓰는 이유를 한 문장으로 쓰시오.

정답 **=는 '같다'와 '넣는다' 두 가지로 읽혀 혼동되므로, 대입임을 분명히 하기 위해서다.**

문제 37 · 주관식

두 수의 최대공약수를 구할 때, 두 수 중 작은 값부터 1씩 줄여 가며 두 수를 모두 나누어떨어지게 하는 수를 찾는 방법이 있다. 이때 가장 먼저 발견되는 수가 곧 최대공약수인 이유를 한 문장으로 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

두 수의 최대공약수를 구할 때, 두 수 중 작은 값부터 1씩 줄여 가며 두 수를 모두 나누어떨어지게 하는 수를 찾는 방법이 있다. 이때 가장 먼저 발견되는 수가 곧 최대공약수인 이유를 한 문장으로 쓰시오.

정답 **큰 값부터 내려오며 검사하므로 가장 먼저 발견되는 공약수가 가장 큰 공약수이기 때문이다.**

해설

반대로 1부터 올라가며 검사하면 끝까지 다 본 뒤에야 최대값을 알 수 있다.

다음 코드에서 while 반복이 멈추는 조건과, 그때 반환되는 값이 무엇인지 쓰시오.

code.py

```
def gcd(a, b):
    while b != 0:
        temp = a
        a = b
        b = temp % b
    return a
```

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

다음 코드에서 while 반복이 멈추는 조건과, 그때 반환되는 값이 무엇인지 쓰시오.

code.py

```
def gcd(a, b):
    while b != 0:
        temp = a
        a = b
        b = temp % b
    return a
```

정답 **b가 0이 될 때 멈추며, 그때의 a가 반환된다.**

해설

0은 모든 수로 나누어떨어지므로 gcd(a, 0) = a이며, 이 a가 두 수의 최대공약수다.

다음 코드를 S = [1, 3, 5, 7, 9, 11, 13, 15], x = 13 으로 실행할 때 처음 계산되는 mid 값과 최종 반환값을 쓰시오.

code.py

```
def binary_search(S, x):
    low = 0
    high = len(S) - 1
    while low <= high:
        mid = (low + high) // 2
        if S[mid] == x:
            return mid
        elif S[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

다음 코드를 S = [1, 3, 5, 7, 9, 11, 13, 15], x = 13 으로 실행할 때 처음 계산되는 mid 값과 최종 반환값을 쓰시오.

code.py

```
def binary_search(S, x):
    low = 0
    high = len(S) - 1
    while low <= high:
        mid = (low + high) // 2
        if S[mid] == x:
            return mid
        elif S[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

정답 **처음 mid = 3, 반환값 6**

해설

(0+7)//2 = 3 → S[3]=7 < 13 → low=4, mid=5 → S[5]=11 < 13 → low=6, mid=6에서 발견.

이진 탐색에서 데이터의 개수가 2배로 늘어나면 최대 비교 횟수는 몇 회 늘어나는지 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

이진 탐색에서 데이터의 개수가 2배로 늘어나면 최대 비교 횟수는 몇 회 늘어나는지 쓰시오.

정답 **1회**

해설

절반씩 줄이므로 범위를 한 번 더 자르면 된다. 곱하기가 더하기가 되는 것이 로그의 성질이다.

시간 복잡도가 입력 크기에 따라 무엇의 증가 정도를 평가하는 척도인지 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

시간 복잡도가 입력 크기에 따라 무엇의 증가 정도를 평가하는 척도인지 쓰시오.

정답 입력 크기에 따른 기본 연산 횟수의 증가 정도

해설

입력이 커질 때 비교·대입 등 기본 연산 횟수가 어떻게 늘어나는지 평가한다.

알고리즘 안에는 대입·비교·산술 등 여러 종류의 명령이 있는데도 기본 연산 하나만 세어도 되는 이유를 한 문장으로 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

알고리즘 안에는 대입·비교·산술 등 여러 종류의 명령이 있는데도 기본 연산 하나만 세어도 되는 이유를 한 문장으로 쓰시오.

정답 **명령들이 대체로 함께 증가하므로 대표 하나만 세어도 전체의 증가 경향을 알 수 있기 때문이다.**

해설

우리가 알고 싶은 것은 정확한 실행 횟수가 아니라 입력이 커질 때의 증가 양상이다.

안쪽 반복문의 덧셈 실행 횟수 T(n)은? (n=len(A), 다른 연산은 제외)

code.py

```
def f(A):
    n = len(A)
    total = 0
    for i in range(n):
        for j in range(n):
            total = total + A[i] * A[j]
    return total
```

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

정답 및 해설 43

안쪽 반복문의 덧셈 실행 횟수 T(n)은? (n=len(A), 다른 연산은 제외)

code.py

```
def f(A):
    n = len(A)
    total = 0
    for i in range(n):
        for j in range(n):
            total = total + A[i] * A[j]
    return total
```

정답 $T(n)=n^2$

해설
바깥 반복문 n회마다 안쪽 덧셈이 n회 실행되므로 총 $n \times n = n^2$회이다.

다음 코드의 시간 복잡도를 빅오 표기로 쓰시오. (n = len(A))

code.py

```
def h(A):
    n = len(A)
    for i in range(n):
        for j in range(10):
            print(A[i], j)
```

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

다음 코드의 시간 복잡도를 빅오 표기로 쓰시오. (n = len(A))

code.py

```
def h(A):
    n = len(A)
    for i in range(n):
        for j in range(10):
            print(A[i], j)
```

정답 **O(n)**

해설

안쪽이 n과 무관하게 10회 고정이므로 10n이고, 상수를 버리면 O(n)이다. 이중 반복문이라고 항상 $O(n^2)$인 것은 아니다.

다음 코드의 시간 복잡도를 빅오 표기로 쓰시오.

code.py

```
def g(n):
    count = 0
    i = 1
    while i < n:
        count = count + 1
        i = i * 2
    return count
```

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

다음 코드의 시간 복잡도를 빅오 표기로 쓰시오.

code.py

```
def g(n):
    count = 0
    i = 1
    while i < n:
        count = count + 1
        i = i * 2
    return count
```

정답 **O(log n)**

해설

i가 1, 2, 4, 8 ...로 두 배씩 커지므로 약 $\log_2 n$회 만에 종료된다. 반복문이 하나라고 항상 O(n)인 것은 아니다.

정렬 문제와 행렬 곱셈 문제에서 입력 크기 n이 각각 무엇을 가리키는지 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

정렬 문제와 행렬 곱셈 문제에서 입력 크기 n이 각각 무엇을 가리키는지 쓰시오.

정답 **정렬은 리스트의 원소 개수, 행렬 곱셈은 행렬의 차원(한 변의 길이)**

해설

$n \times n$ 행렬의 원소는 n^2개이므로 n을 원소 개수로 착각하지 않아야 한다.

$T(n)=n^2+n+1$에서 n이 매우 커질수록 최고차항 n^2이 전체에서 차지하는 비율은 몇 %에 가까워지는가?

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

$T(n)=n^2+n+1$에서 n이 매우 커질수록 최고차항 n^2이 전체에서 차지하는 비율은 몇 %에 가까워지는가?

정답 100%

해설

n^2으로 나누면 비율은 $1/(1+1/n+1/n^2)$이다. n이 커질수록 1, 즉 100%에 가까워진다. 소수점 나눗셈은 필요 없다.

빅오(O), 빅오메가(Ω), 빅세타(Θ)가 각각 나타내는 것을 순서대로 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

빅오(O), 빅오메가(Ω), 빅세타(Θ)가 각각 나타내는 것을 순서대로 쓰시오.

정답 **상한(천장), 하한(바닥), 상한이자 하한(정확한 차수)**

해설

Θ가 성립하면 O와 Ω는 자동으로 성립한다.

알고리즘에서 로그의 밑을 적지 않고 그냥 log n으로 쓰는 이유를 한 문장으로 쓰시오.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

알고리즘에서 로그의 밑을 적지 않고 그냥 log n으로 쓰는 이유를 한 문장으로 쓰시오.

정답 **밑이 달라도 상수배 차이일 뿐이고 빅오는 상수배를 무시하기 때문이다.**

해설

$\log_2 n$과 $\log_{10} n$은 일정한 상수를 곱한 관계이므로 같은 $O(\log n)$이다.

두 알고리즘의 연산 횟수는 $T_1(n)=n^2$, $T_2(n)=n \log_2 n$이다. $n=1{,}000{,}000$, $\log_2 n \approx 20$일 때 각각의 연산 횟수를 쓰시오. 곱셈식으로 답해도 된다.

답 값 · 용어 · 식으로 한 줄로 답하면 됩니다.

두 알고리즘의 연산 횟수는 $T_1(n)=n^2$, $T_2(n)=n\log_2 n$이다. $n=1{,}000{,}000$, $\log_2 n \approx 20$일 때 각각의 연산 횟수를 쓰시오. 곱셈식으로 답해도 된다.

정답 10^{12}회, 약 2×10^{7}회

해설

$T_1=10^6\times10^6$, $T_2\approx10^6\times20$이다. 빅오만으로 실제 연산 횟수를 알 수는 없으므로 문제에 제시한 T_1, T_2를 이용한다.