1. <보기> 수식을 전위 표기법(prefix notation)으로 변환한 후, 스택을 이용하여 계산하려고 한다. 계산 과정에서 스택에 다섯 번째로 삽입(push)하는 값은? (단, 전위표기법을 스택으로 계산할 때, 오른쪽에서 왼쪽으로 읽으며 진행한다.)

<보기>

(A−B)*C+D/E

① A ② B
③ D ④ E

2. 이중 연결 리스트(doubly linked list)의 특징에 대한 설명으로 가장 옳은 것은?

① 각 노드가 이전 노드와 다음 노드의 정보를 가진다.
② 노드들이 메모리의 연속된 공간에 저장된다.
③ 모든 노드는 하나의 링크만 가진다.
④ 인덱스를 이용하여 원하는 노드에 직접 접근할 수 있다.

3. <보기>는 피보나치 수열을 계산하는 C 프로그램이다. fib(5)을 호출했을 때, 함수 fib( )가 호출되는 총 횟수는? (단, 첫 호출도 1회로 계산한다.)

<보기>

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

① 7 ② 12
③ 15 ④ 17

4. 탐색 알고리즘의 설명으로 가장 옳지 않은 것은?

① 순차 탐색의 시간 복잡도는 $O(n)$이다.
② 색인 순차 탐색에서 인덱스 테이블의 크기가 줄어들면, 배열의 인덱스를 저장하는 간격이 커지므로 배열에서 탐색해야 하는 범위도 커진다.
③ 색인 순차 탐색의 시간 복잡도는 $O(m+m/n)$이다. (단, n: 배열의 크기, m: 인덱스 테이블의 크기)
④ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21과 같이 정렬되었을 때 이진 탐색으로 찾고자 하는 수가 17일 경우, 비교 횟수는 2번이다.

5. <보기>의 C 프로그램 함수 func에 대해서 배열 arr[] = {1, 2, 3, 4, 5, 6}일 때, func(arr, 6, 2)를 실행한 후 배열 arr[]의 상태는?

<보기>

```
void func(int arr[], int n, int k) {
    int temp[100];
    for (int i = 0; i < n; i++)
        temp[i] = arr[(i + k) % n];
    for (int i = 0; i < n; i++)
        arr[i] = temp[i];
}
```

① [1, 2, 3, 4, 5, 6]
② [2, 3, 4, 5, 6, 1]
③ [3, 4, 5, 6, 1, 2]
④ [5, 6, 1, 2, 3, 4]

6. 그래프에 대한 설명으로 가장 옳지 않은 것은?

① 완전 그래프에서 정점이 n개인 방향 그래프의 최대 간선 수는 $n(n-1)$개이다.
② 방향 그래프의 인접 행렬에서 행 i의 합은 정점 i의 진출 차수이다.
③ 방향 그래프의 인접 리스트에서 각 정점의 헤드에 연결된 노드의 수는 정점의 진출 차수이다.
④ e개의 간선을 가진 무방향 그래프의 인접 리스트에 저장되는 연결 노드의 총 수는 e개이다.

7. <보기>에서 이진 트리(binary tree)의 성질에 대한 설명으로 옳은 것을 모두 고른 것은? (단, 트리의 높이는 루트 노드의 레벨을 1로 하여 계산한다.)

<보기>

ㄱ. 총 n개의 노드를 가진 이진 트리의 높이(h)는 적어도 $h \geq \lceil \log_2(n+1) \rceil$ 이다.
ㄴ. 완전 이진 트리(complete binary tree)에서 모든 단말 노드는 항상 같은 레벨에 존재한다.
ㄷ. 이진 탐색 트리(binary search tree)의 삽입에 드는 접근적 시간은 검색에 드는 시간과 동일하다.
ㄹ. n개의 노드를 가진 이진 탐색 트리의 접근적 평균 검색 시간은 $\Theta(n)$이다.

① ㄱ ② ㄱ, ㄷ
③ ㄴ, ㄹ ④ ㄱ, ㄷ, ㄹ

8. <보기>의 알고리즘 시간 복잡도로 가장 옳은 것은?

<보기>

```
Algorithm(int n) {
    int i, j, k, l;
    int sum = 0;
    for (i = 0; i < 500; i++)
        for (j = 0; j < 100; j++)
            for (k = 1; k < n; k = k * 2)
                for (l = 0; l < 10; l++)
                    sum = sum + i + j + k + l;
}
```

① $O(1)$ ② $O(n)$

③ $O(n^2)$ ④ $O(\log n)$

9. 크기가 6인 원형 큐(circular queue)가 있다. <보기 1>의 설명을 참고하여 <보기 2>의 연산을 모두 수행한 후, rear가 가리키는 배열의 인덱스는?

<보기 1>

- 배열의 인덱스는 0부터 5까지 사용한다.
- 초기 상태에서 front와 rear은 모두 0을 가리키며, 큐는 공백 상태이다.
- 공백 상태와 포화 상태를 구분하기 위해 항상 한 칸을 비워 두는 방식으로 구현한다.
- enqueue는 데이터를 삽입하는 연산이며, rear을 먼저 한 칸 이동시킨 후 해당 위치에 저장한다.
- dequeue는 데이터를 삭제하는 연산이며, front를 먼저 한 칸 이동시킨 후 해당 데이터를 삭제한다.

<보기 2>

enqueue(A) ⇒ enqueue(B) ⇒ enqueue(C) ⇒ dequeue() ⇒ enqueue(D) ⇒ enqueue(E) ⇒ enqueue(F)

① 0 ② 3

③ 4 ④ 5

10. [91, 42, 17, 12, 80, 95, 3, 27] 배열을 셸(shell) 정렬로 오름차순 정렬할 때, 간격을 h에 저장하는데, $h=2$인 단계가 완료된 직후의 배열 상태는? (단, 최초 간격 h는 배열 원소 개수 n에 대하여 $h=\frac{n}{2}$이고 이후 각 단계에서 절반으로 감소한다.)

① [3, 12, 17, 27, 80, 42, 91, 95]

② [3, 12, 27, 17, 80, 42, 91, 95]

③ [80, 42, 3, 12, 91, 95, 17, 27]

④ [80, 42, 3, 12, 95, 91, 17, 27]

11. 힙(heap)에 대한 설명으로 가장 옳지 않은 것은?

① 최대 힙은 키값이 가장 큰 노드를 찾기 위한 완전 이진 트리로서 부모 노드의 키값이 자식 노드의 키값보다 항상 크거나 같고, 최소 힙은 부모 노드의 키값이 자식 노드의 키값보다 항상 작다.

② 삽입 연산 시 노드가 n개인 완전 이진 트리에서 다음 노드의 확장 자리는 $n+1$번의 노드이며 그 자리에 삽입할 원소를 임시 저장한다.

③ 일반적인 힙의 삭제 연산 시 루트 노드에 있는 원소를 삭제한다.

④ 힙을 구현할 때 1차원 배열의 순차 자료구조를 이용하면 인덱스 관계를 이용하여 부모 노드를 찾기가 쉽다.

12. 가중치 무방향 그래프에서 크루스칼(Kruskal) 알고리즘을 이용해 최소 비용 신장 트리를 구성하려고 한다. 간선의 가중치는 <보기>와 같다. 크루스칼 알고리즘으로 간선을 선택하는 과정에서, 세 번째로 신장 트리에 포함되는 간선은?

<보기>

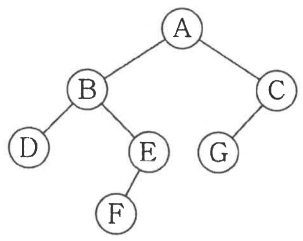
(A, B): 2, (A, C): 3, (B, C): 1, (B, D): 5, (C, D): 4

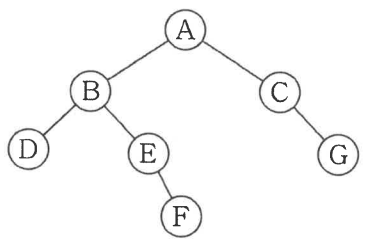
① (A, B) ② (A, C)

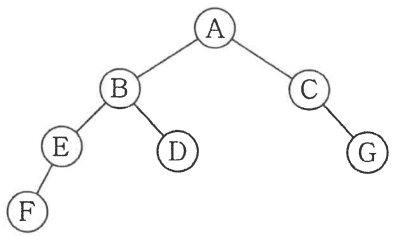
③ (B, D) ④ (C, D)

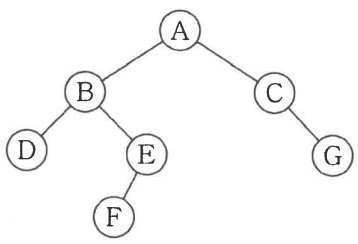
13. <보기>와 같은 순회 결과들을 모두 만족하는 7개의 노드로 이루어진 이진 트리의 구조로 옳은 것은?

<보기>
전위 순회: A − B − D − E − F − C − G
중위 순회: D − B − F − E − A − C − G
후위 순회: D − F − E − B − G − C − A

①


②


③


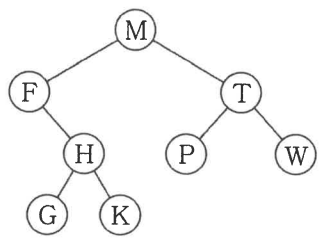
④


14. <보기>의 코드는 원형 연결 리스트(circular linked list)에서 특정 노드 old를 삭제하는 C언어 함수이다. 빈칸 (가), (나)에 들어갈 내용으로 가장 옳은 것은? (단, head는 첫 번째 노드를 가리키며, 삭제 후에도 기존 노드들의 순서는 유지된다.)

<보기>
```
void deleteNode(linkedList_h* CL, listNode* old) {
    listNode* pre;
    if (CL->head == NULL) return;
    if (CL->head->link == CL->head) {
        free(CL->head);
        CL->head = NULL;
        return;
    }
    else if (old == NULL) return;
    else {
        pre = CL->head;
        while (pre->link != old) {
            pre = pre->link;
        }
        (가)
        if (old == CL->head) {
            (나)
        }
        free(old);
    }
}
```

	(가)	(나)
①	old->link = pre->link;	CL->head = old->link;
②	pre->link = old->link;	CL->head = old->link;
③	pre->link = old->link;	CL->head = pre;
④	pre = old->link;	CL->head = old->link;

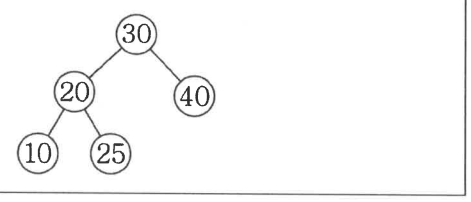
15. <보기>의 트리를 중위 스레드 이진트리로 구성하려고 한다. 이때 실제 자식 노드를 가리키는 링크 수를 a, 중위 선행자 또는 중위 후속자를 가리키는 스레드 수를 b, 스레드 구성 후에도 NULL로 남는 포인터 수를 c라고 할 때, a, b, c의 값으로 가장 옳은 것은? (단, 중위 선행자나 중위 후속자가 없는 경우 해당 스레드 포인터에는 NULL을 저장한다.)

<보기>


① $a=7$ $b=6$ $c=1$
② $a=7$ $b=7$ $c=2$
③ $a=7$ $b=8$ $c=1$
④ $a=9$ $b=7$ $c=2$

16. <보기>의 AVL 트리에 5를 삽입한 후, AVL 트리의 균형을 유지하기 위해 수행해야 하는 회전으로 가장 옳은 것은?

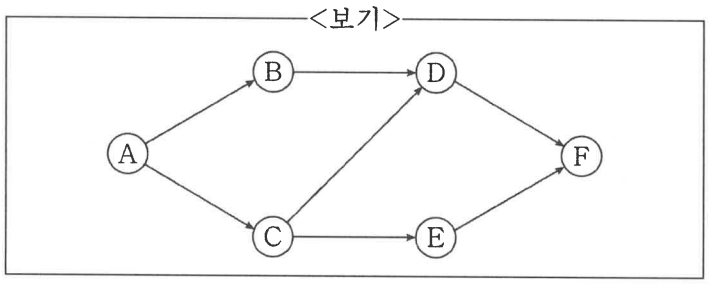
<보기>



① LL 회전
② RR 회전
③ LR 회전
④ RL 회전

17. <보기>의 방향 그래프에 대해 위상 정렬(topological sort)을 수행할 때, 가능한 서로 다른 위상 순서의 총 개수는?

<보기>



① 3
② 4
③ 5
④ 6

18. [691, 423, 17, 124, 80, 95, 3, 227] 배열을 LSD(least significant digit) 기수 정렬로 오름차순 정렬할 때, 2단계가 완료된 직후 배열 상태로 가장 옳은 것은?

① [3, 17, 124, 227, 423, 80, 691, 95]
② [3, 17, 423, 124, 227, 80, 691, 95]
③ [17, 3, 227, 423, 124, 80, 95, 691]
④ [17, 3, 423, 227, 124, 80, 95, 691]

19. <보기>는 무방향 그래프에서 깊이 우선 탐색(DFS, depth first search)을 수행하는 C 프로그램이다. dfs(0)을 호출했을 때, order[2]와 order[3]의 값으로 가장 옳은 것은?

<보기>

```
int adj[5][5] = {
{0, 1, 1, 0, 0},
{1, 0, 0, 1, 1},
{1, 0, 0, 0, 1},
{0, 1, 0, 0, 0},
{0, 1, 1, 0, 0}
};
int visited[5] = { 0, };
int order[5];
int cnt = 0;

void dfs(int v) {
    visited[v] = 1;
    order[v] = ++cnt;
    for (int i = 0; i < 5; i++) {
        if (adj[v][i] == 1 && !visited[i]) {
            dfs(i);
        }
    }
}
```

① order[2]: 2, order[3]: 4
② order[2]: 3, order[3]: 4
③ order[2]: 5, order[3]: 3
④ order[2]: 5, order[3]: 4

20. 크기가 8인 해시 테이블에서 해시 함수 $h(k) = k \bmod 8$을 사용하고, 충돌은 체이닝(chaining)으로 처리한다. <보기>의 키를 순서대로 삽입할 때, 가장 긴 체인이 형성된 버킷의 인덱스와 해당 체인의 길이를 옳게 짝지은 것은? (단, 버킷의 인덱스는 0부터 시작한다.)

<보기>

13, 25, 6, 21, 14, 29, 8

① 인덱스 0, 길이 2
② 인덱스 5, 길이 3
③ 인덱스 5, 길이 4
④ 인덱스 6, 길이 2