

자료구조론

1. 포인터 변수 L이 단순 연결 리스트의 첫 번째 노드를 가리킬 때, (가) ~ (다)를 수행하는 각 알고리즘의 시간 복잡도를 바르게 연결한 것은? (단, n 은 리스트의 노드 개수를 나타낸다)

(가) L이 가리키는 리스트의 노드 순서를 역순으로 변환
(나) 새로운 노드를 L이 가리키는 리스트의 첫 번째 노드가 되도록 삽입
(다) L이 가리키는 리스트의 마지막 노드 삭제

	<u>(가)</u>	<u>(나)</u>	<u>(다)</u>
①	$O(1)$	$O(1)$	$O(1)$
②	$O(1)$	$O(n)$	$O(n)$
③	$O(n)$	$O(1)$	$O(n)$
④	$O(n)$	$O(n)$	$O(1)$

2. 다음은 같은 기능을 수행하는 두 함수를 구현한 C 코드이다. 각 함수의 시간 복잡도를 바르게 연결한 것은?

```
double p1(double x, int n) {
    int i;
    double result = 1.0;

    for(i = 0; i < n; i++)
        result = result * x;
    return result;
}

double p2(double x, int n) {
    if(n == 0)
        return 1;
    else if((n % 2) == 0)
        return p2(x * x, n/2);
    else
        return x * p2(x * x, (n - 1)/2);
}
```

	<u>p1 함수</u>	<u>p2 함수</u>
①	$O(n)$	$O(1)$
②	$O(n)$	$O(\log n)$
③	$O(2^n)$	$O(n^2)$
④	$O(\log n)$	$O(n \log n)$

3. 다음은 이중 연결 리스트(doubly linked list)를 구현한 C 코드이다. 노드 삭제 함수의 빈칸에 들어갈 내용으로 옳은 것은?

```
typedef int element;
typedef struct DListNode {
    element data;
    struct DListNode *llink;
    struct DListNode *rlink;
} DListNode;
DListNode *head = NULL;

void init() {
    head = (DListNode*)malloc(sizeof(DListNode));
    head->llink = head->rlink = head;
}

void dinser(DListNode *inode, DListNode *inserted) {
    inserted->llink = inode;
    inserted->rlink = inode->rlink;
    inode->rlink->llink = inserted;
    inode->rlink = inserted;
}

void ddelete(DListNode *removed) {
    if(removed == head) return;
    removed->llink->rlink = removed->rlink;
    
    free(removed);
}
```

- ① $removed->rlink->llink = removed->llink;$
② $removed->rlink->llink = removed->rlink;$
③ $removed->llink = removed->rlink->llink;$
④ $removed->rlink = removed->rlink->llink;$

4. 2차원 배열 A[1:15][1:20]의 원소 A[10][8]에 대하여, 행 우선 순서 주소와 열 우선 순서 주소를 바르게 연결한 것은? (단, 첫 번째 원소 A[1][1]의 주소는 2000이고, 각 원소의 크기는 1바이트이다)

	<u>행 우선 순서 주소</u>	<u>열 우선 순서 주소</u>
①	2114	2265
②	2182	2187
③	2187	2114
④	2265	2182

5. 다음은 연결 리스트를 이용하여 스택을 구현한 C 코드이다. (가), (나)에 들어갈 내용을 바르게 연결한 것은?

```
typedef struct _node {
    int key;
    struct _node *next;
} node;

node *head = NULL;
node *tail = NULL;

void init() {
    head = (node*)malloc(sizeof(node));
    tail = (node*)malloc(sizeof(node));
    head->next = tail;
    tail->next = tail;
}

int push(int k) {
    node *t;
    if((t = (node*)malloc(sizeof(node))) == NULL) {
        printf("\n Out of memory");
        return -1;
    }
    t->key = k;
    (가)

    head->next = t;
    return k;
}

int pop() {
    node *t;
    int i;
    if(head->next == tail) {
        printf("\n Stack underflow");
        return -1;
    }
    t = head->next;
    i = t->key;
    (나)

    free(t);
    return i;
}
```

(가)

(나)

- | | |
|-------------------------|-----------------------|
| ① t->next = head; | head->next = t->next; |
| ② t->next = head->next; | head->next = t->next; |
| ③ t->next = head->next; | head = t->next; |
| ④ t->next = head->next; | head->next = t; |

6. 다음 C 언어로 작성된 함수들의 시간 복잡도에 대한 설명으로 옳은 것은?

```
void func1(int n) {
    for(int i = 0 ; i < 2 ; i++)
        for(int j = 0 ; j < n ; j++)
            printf("%d\n", j);
}

void func2(int n) {
    for(int i = 0 ; i < 100 ; i++)
        printf("%d\n", i);
}

void func3(int n) {
    if(n == 0)
        return;
    printf("%d\n", n);
    func3(n/2); ..... (가)
}
```

- ① func1 함수는 이중 for 반복문으로 구성되어 시간 복잡도는 $O(n^2)$ 이다.
 ② func2 함수는 for 반복문으로 구성되어 시간 복잡도는 $O(n)$ 이다.
 ③ func3 함수는 for 반복문이 없어 시간 복잡도는 $O(1)$ 이다.
 ④ (가) 문장을 `func2(n/2);`으로 수정할 경우, func3 함수의 시간 복잡도는 $O(1)$ 이다.

7. 다음은 원형 연결 리스트의 메모리 구성이다. 메모리 주소 1500인 위치에 data와 link 항목 내용이 각각 F와 1400인 새 노드를 삽입할 때, 변경되는 내용으로 옳은 것은?

메모리 주소	data	link
1000	A	1300
1100	B	1400
1200	C	1000
1300	D	1100
1400	E	1200

- ① data 값이 A인 노드의 link 값을 1400으로 변경
 ② data 값이 B인 노드의 link 값을 1500으로 변경
 ③ data 값이 C인 노드의 link 값을 1400으로 변경
 ④ data 값이 E인 노드의 link 값을 1500으로 변경

8. 다음은 크기가 7인 배열로 구현된 원형 덱(deque)의 상태이다. 현재 상태에서 연산 1부터 연산 5까지 순서대로 수행하였을 때, 수행이 완료된 후 원형 덱의 상태는? (단, 현재 상태는 front = 0, rear = 3이다)

[0]	[1]	[2]	[3]	[4]	[5]	[6]
5	2	6	8			

연산 1: 맨 뒤에 위치한 데이터 삭제
 연산 2: 맨 앞에 위치한 데이터 삭제
 연산 3: 맨 뒤에 데이터 3을 삽입
 연산 4: 맨 앞에 데이터 1을 삽입
 연산 5: 맨 앞에 데이터 4를 삽입

①

[0]	[1]	[2]	[3]	[4]	[5]	[6]
1	2	6	3			4

②

[0]	[1]	[2]	[3]	[4]	[5]	[6]
4	1	2	6	3		

③

[0]	[1]	[2]	[3]	[4]	[5]	[6]
1	2	6	3	4		

④

[0]	[1]	[2]	[3]	[4]	[5]	[6]
4	1	2	6			3

9. 다음 C 프로그램의 실행 결과는?

```
#include <stdio.h>

void function(int array[], int idx, int size) {
    for(int i = idx; i < size - 1; i++)
        array[i] = array[i + 1];
}

int main() {
    int array[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    function(array, 5, 10);
    function(array, 5, 10);

    for(int i = 0; i < 10; i++)
        printf("%d ", array[i]);
    printf("\n");

    return 0;
}
```

- ① 0 1 2 3 4 5 6 7 8 9
 ② 0 1 2 3 4 7 8 9 9 9
 ③ 0 1 2 3 5 5 6 7 8 9
 ④ 0 1 2 3 5 6 7 8 9 9

10. 이진 탐색 트리의 모든 키를 오름차순으로 정렬한 결과를 일차원 배열 형식으로 출력하기 위해 사용할 수 있는 순회 알고리즘은? (단, 순회 알고리즘은 1회만 수행하고 별도의 정렬 알고리즘은 수행하지 않는다)
- ① 중위 순회
 ② 전위 순회
 ③ 후위 순회
 ④ 레벨 순회

11. 정점의 개수가 n인 단순 그래프가 트리아기 위한 필요충분조건으로 옳지 않은 것은? (단, n은 3 이상의 정수이다)
- ① 사이클이 없고 연결요소의 개수가 1이다.
 ② 임의의 두 정점 사이에 존재하는 경로의 개수는 1이다.
 ③ 간선의 개수가 n-1이고, 차수가 1인 정점의 개수가 2 이상이다.
 ④ 사이클이 없지만, 인접하지 않은 임의의 두 정점 사이에 간선을 추가하면 사이클의 개수가 1이 된다.

12. 다음과 같이 6개의 키값으로 구성된 4개의 입력 배열 데이터 (가) ~ (라)를 각각 순서대로 삽입하여 4개의 AVL 트리를 생성하였다. 생성된 모든 트리에 공통으로 존재하는 리프 노드 키값으로 옳은 것은?

(가)	4	1	2	8	5	9
(나)	1	2	8	5	9	4
(다)	9	8	4	2	1	5
(라)	9	8	1	5	4	2

- ① 2
 ② 4
 ③ 5
 ④ 9

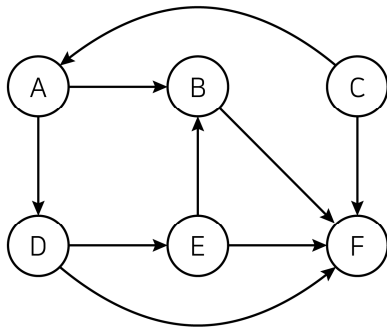
13. 선형 조사법(linear probing)을 사용하는 해시 테이블에서 버킷(bucket)을 삭제할 때, 빈 버킷(empty bucket)으로 비워 두지 않고 삭제했음을 표시하는 이유는? (단, 버킷 내 슬롯(slot) 수는 1이다)
- ① 1차 군집 현상이 발생하지 않게 한다.
 ② 해시 테이블의 전체 크기를 자동으로 기존 크기의 두 배로 확장시킨다.
 ③ 충돌로 인해 원래 해시 주소와는 다른 주소에 저장된 레코드에 대한 탐색이 가능하게 한다.
 ④ 해시 테이블에 저장된 모든 레코드의 키값이 항상 오름차순으로 정렬된 상태가 되도록 유지시킨다.

14. 다음 C 언어로 구현한 sort 함수에서 (가) 문장이 수행되는 횟수가 가장 작은 배열 arr의 입력내용은? (단, $n = 5$ 이다)

```
void sort(int arr[], int n) {
    int i, key, j;
    for(i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while(j >= 0 && arr[j] < key) {
            arr[j + 1] = arr[j]; .....(가)
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

- ① 1, 2, 3, 4, 5
 ② 5, 3, 4, 7, 6
 ③ 8, 3, 10, 7, 2
 ④ 10, 9, 8, 7, 6
15. 단순 그래프의 인접 행렬 표현이 인접 리스트 표현보다 점근적으로 빠른 연산으로 옳은 것은? (단, 인접 행렬에서 임의의 정점 v 의 행 또는 열의 접근 시간과 인접 리스트에서 임의의 정점 v 의 인접 리스트의 접근 시간은 모두 상수이다)
- ① 특정 정점의 차수 계산
 ② 그래프 전체의 간선 개수 세기
 ③ 특정 정점에서 시작하는 너비 우선 탐색
 ④ 특정 두 정점 사이의 간선 존재 여부 확인

16. 다음 그래프에 대한 깊이 우선 탐색(depth first search)을 수행할 때 가능한 정점들의 방문 순서는?



- ① A, B, F, C, D, E
 ② C, A, B, D, E, F
 ③ D, F, E, B, A, C
 ④ E, F, A, B, D, C

17. 2-3 트리에 대한 설명으로 옳은 것은?

- ① 2-3 트리가 항상 완전 균형 트리를 유지하는 것은 아니다.
 ② 2-3 트리의 삽입은 B 트리와 달리 리프 노드 외에서도 발생한다.
 ③ 3-노드에 저장된 키값들은 그 노드의 모든 자식 노드의 키값보다 크다.
 ④ 3-노드를 2개의 2-노드로 표현하여 2-3 트리를 레드-블랙 트리로 만들 수 있다.

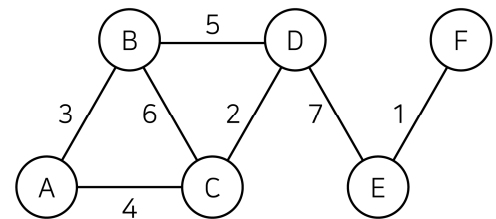
18. n 개의 버킷(bucket)으로 구성된 비어있는 해시 테이블에 다음 <조건>에 따라 해시 함수 $h(x) = x \bmod n$ 을 사용하여 키값 <1, 20, 2, 35, 18>을 차례대로 삽입할 때, 발생하는 전체 충돌 횟수가 가장 많은 n 값은?

— <조 건> —

- n 은 자연수이다.
- 버킷 내 슬롯(slot) 수는 1이다.
- 버킷의 인덱스는 0부터 시작한다.
- 충돌 해결 방법으로 선형 조사법(linear probing)을 사용한다.
- 각 키값의 삽입과정에서 발생하는 충돌 횟수가 k (k 는 0 이상의 정수)일 때 전체 충돌 횟수가 k 만큼 증가한다.

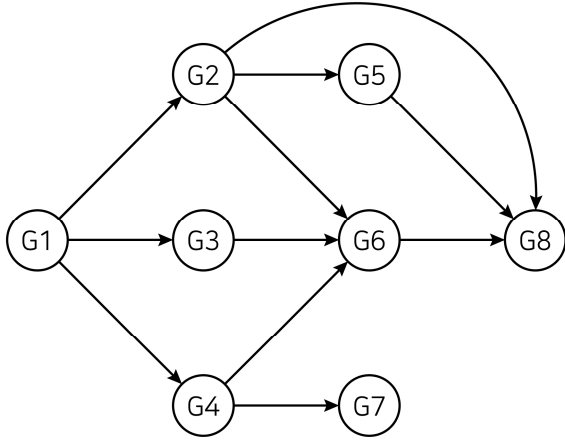
- ① 5
 ② 7
 ③ 13
 ④ 17

19. 다음 그래프의 최소 비용 신장 트리(minimum cost spanning tree)에 대한 설명으로 옳은 것은?



- ① 사이클이 있어 최소 비용 신장 트리가 존재하지 않는다.
 ② 가중치가 가장 큰 간선은 최소 비용 신장 트리에 포함되지 않는다.
 ③ 모든 간선 가중치가 서로 달라 최소 비용 신장 트리의 개수는 1이다.
 ④ 두 정점 B와 D 사이의 최단 경로는 최소 비용 신장 트리에 포함된다.

20. 다음 그래프에 대해 위상 정렬(topological sort)을 수행할 때 생성되는 위상 순서로 옳지 않은 것은?



- ① G1, G2, G3, G4, G5, G6, G7, G8
 ② G1, G2, G3, G4, G5, G8, G6, G7
 ③ G1, G3, G2, G4, G6, G5, G8, G7
 ④ G1, G4, G3, G2, G7, G6, G5, G8
21. 다음 C 언어로 구현한 arraySort 함수에서 재귀 호출 횟수의 최댓값이 가장 큰 배열 arr의 입력내용은? (단, arraySort 함수의 최초 호출 시 low 변수와 high 변수는 배열 arr의 첫 번째와 마지막 데이터의 인덱스이다)

```

#define SWAP(x,y) {int temp = x; x = y; y = temp;}

int splitArray(int arr[], int low, int high) {
    int pi = (low + high)/2;
    SWAP(arr[pi], arr[high]);
    int pivot = arr[high];

    int i = low - 1;
    for(int j = low; j < high; j++) {
        if(arr[j] < pivot) {
            i++;
            SWAP(arr[i], arr[j]);
        }
    }
    SWAP(arr[high], arr[i + 1]);
    return i + 1;
}

void arraySort(int arr[], int low, int high) {
    if(low < high) {
        int pi = splitArray(arr, low, high);
        arraySort(arr, low, pi - 1);
        arraySort(arr, pi + 1, high);
    }
}
  
```

- ① 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
 ② 2, 4, 6, 8, 10, 5, 3, 7, 1, 9
 ③ 3, 6, 2, 8, 5, 10, 7, 1, 9, 4
 ④ 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

22. 다음은 후위 표기식을 스택을 이용하여 사칙연산을 하는 알고리즘을 구현한 C 프로그램이다. 출력 결과가 12인 프로그램의 실행에 대한 설명으로 옳은 것만을 모두 고르면?

```

#include <stdio.h>
#include <stdlib.h>

#define MAX_STACK_SIZE 50

typedef int Item;
Item stack[MAX_STACK_SIZE];
int top;

void reset_stack() { top = -1; }

void push(Item elm) { stack[++top] = elm; }

Item pop() { return stack[top--]; }

Item calc_exp(char *exp) {
    int x, y, z, current, i = 0;
    reset_stack();
    while(exp[i] != '$') {
        current = exp[i++];
        if(current >= '0' && current <= '9')
            push((가));
        else if(current == '+' || current == '-' ||
                current == '*' || current == '/') {
            (나) = pop();
            (다) = pop();
            switch(current) {
                case '+': z = x + y; break;
                case '-': z = x - y; break;
                case '*': z = x * y; break;
                case '/': z = x / y;
            }
            push(z);
        }
    }
    return pop();
}

int main() {
    char exp[20] = "2 3 4 + * 6 3 / - $";
    printf("%d\n", calc_exp(exp));
    return 0;
}
  
```

- ㄱ. (가)에 들어갈 내용은 current이다.
 ㄴ. (나)에 들어갈 내용은 y이고 (다)에 들어갈 내용은 x이다.
 ㄷ. 프로그램을 실행하는 동안 pop 함수의 총호출 횟수는 8회이다.
 ㄹ. 프로그램을 실행하는 동안 push 함수의 총호출 횟수는 9회이다.

- ① ㄱ, ㄴ ② ㄱ, ㄷ
 ③ ㄴ, ㄹ ④ ㄱ, ㄴ, ㄹ

23. 다음 C 프로그램의 실행 결과는?

```
#include <stdio.h>

typedef struct {
    int id;
    char name[10];
} Student;

void func(Student *dest, Student *src){
    dest = src;
}

int main() {
    Student st_a = {1, "AAA"}, st_b = {2, "BBB"};
    Student *st_arr[3] = {&st_a, &st_b, &st_b};

    func(st_arr[2], st_arr[0]);
    st_arr[2]->name[0] = 'X';

    for(int i = 0; i < 3; i++)
        printf("%s ", st_arr[i]->name);

    return 0;
}
```

- ① AAA XBB XBB
 ② AAA BBB XBB
 ③ AAA BBB XAA
 ④ XAA BBB XAA

24. 다음 C 프로그램의 실행 결과는?

```
#include <stdio.h>

int count = 0;

void new_hanoi(int n, char from, char tmp, char to) {
    if (n <= 2)
        count += n;
    else {
        new_hanoi(n - 2, from, to, tmp);
        count += 2;
        new_hanoi(n - 2, tmp, from, to);
    }
}

int main() {
    new_hanoi(9, 'A', 'B', 'C');
    printf("%d", count);
    return 0;
}
```

- ① 31
 ② 46
 ③ 62
 ④ 511

25. 다음은 입력 배열 데이터 data 내에 찾고자 하는 값 key와 같은 값이 저장된 모든 배열 원소의 인덱스를 배열로 반환하는 search 함수를 포함한 Java 프로그램이다. 출력 결과가 (12:2,13:0,23:3,43:1)인 프로그램의 실행에 대한 설명으로 옳은 것만을 모두 고르면?

```
public class SearchAlgorithm {
    private static int[] index_list;
    private static int size;
    private static void reset() {
        index_list = new int[1];
        size = 0;
    }
    private static void resize(int newsize) {
        int[] nlist = new int[newsize];
        for(int i = 0; i < size; i++)
            nlist[i] = index_list[i];
        index_list = nlist;
    }
    private static void insert_index(int index) {
        if(size == index_list.length) resize(2 * size);
        index_list[ (가) ] = index;
    }
    public static int[] search(int[] data, int key) {
        (나) ;
        for(int i = 0; i < data.length; i++)
            if(data[i] == key) insert_index(i);
        int[] result_list = new int[size];
        for(int i = 0; i < result_list.length; i++)
            result_list[i] = index_list[i];
        return result_list;
    }
    public static void main(String[] args) {
        int[] data = {12, 45, 23, 43, 23, 54, 12, 23};
        int[] keys = {12, 13, 23, 43};
        String index_f = "(";
        for(int i = 0; i < keys.length; i++) {
            int key = keys[i];
            int[] indexs = search(data, key);
            index_f += key + ":" + indexs.length;
            if(i != keys.length - 1) index_f += ",";
        }
        System.out.println(index_f + ")");
        return;
    }
}
```

- ㄱ. (가)에 들어갈 내용은 ++size이다.
 ㄴ. (나)에 들어갈 내용은 reset()이다.
 ㄷ. 프로그램을 실행하는 동안 resize 함수의 총호출 횟수는 4회이다.
 ㄹ. 프로그램을 실행하는 동안 insert_index 함수의 총호출 횟수는 6회이다.

- ① ㄱ, ㄷ
 ② ㄴ, ㄷ
 ③ ㄴ, ㄹ
 ④ ㄱ, ㄴ, ㄹ