자료구조 중간고사 요약 노트

목차

---

1. Big O Notation & Worst Case

1.1 Big O Notation 개념

- 알고리즘의 시간/공간 복잡도를 분석하는 수학적 표기법
- **시간 복잡도**: 실행 시간의 증가율
- **공간 복잡도**: 메모리 사용량
- 최악의 경우(Worst Case)를 기준으로 상한을 표기
- 상수/낮은 차수 항은 무시
    - 예: $n^2 + n + 1$ -> $O(n^2)$

1.2 계산 방법

1. 기본 연산 식별(비교, 할당 등)
2. 반복문 분석
    - 단일: O(n)
    - 중첩: O(n^k)
3. 재귀 함수 분석(호출 깊이)
4. 분할 정복: O(log n), O(n log n)
5. 상수 시간: O(1)

1.3 Worst Case 개념

- 알고리즘이 가장 비효율적으로 동작하는 상황
- 성능 보장을 위해 중요
- 예: 역순 배열(버블 정렬), 값이 없는 검색(선형 검색)

1.4 Big O 차수 크기 비교

- $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$
- 예시 ($n = 10$):

차수	값	예시
$O(1)$	1	배열 인덱스 접근
$O(\log n)$	3.32	이진 검색
$O(n)$	10	선형 검색
$O(n \log n)$	33.2	병합 정렬
$O(n^2)$	100	버블 정렬
$O(n^3)$	1,000	3중 루프
$O(2^n)$	1,024	피보나치 재귀
$O(n!)$	3,628,800	외판원 문제

1.5 Big O 예제 (C 코드 기반)

1. $O(1)$: 변수 덧셈
2. $O(n)$: 선형 검색
3. $O(\log n)$: 이진 검색
4. $O(n^2)$: 버블 정렬
5. $O(n \log n)$: 병합 정렬
6. $O(n^2)$: 행렬 덧셈
7. $O(n)$: 팩토리얼 재귀
8. $O(2^n)$: 피보나치 재귀
9. $O(1)$: 단순 출력
10. $O(n^3)$: 3중 루프

---

2. Array & Pointer

2.1 포인터에 자료형이 존재하는 이유

- 간접 참조(dereference) 시 바이트 크기 결정

```
int *p_int;
char *p_char;
```

→ `*p_int` : sizeof(int) 만큼
→ `*p_char` : sizeof(char) 만큼

- 포인터 산술 연산 시 바이트 이동 단위 결정

```
p_int + 1;  // + sizeof(int)
p_char + 1; // + sizeof(char)
```

- 형 안정성(type safety) 보장
  - 자료형 불일치 시 예기치 않은 결과
  - 예: `double *` → `int *` 로 캐스팅 시 데이터 손상 가능
- 함수 호출 시 포인터 자료형 확인

```
void func(int *p);
```

2.2 배열 vs 포인터 비교

항목	배열(array)	포인터(pointer)
선언	`int arr[5];`	`int *p;`
메모리 할당	컴파일 시, 연속 공간	런타임 malloc 또는 주소 저장
이름 역할	첫 원소 주소로 평가	주소를 담는 변수
sizeof	전체 크기	포인터 크기
대입	불가능(immutable)	가능
주소 연산	`&arr` → 전체 배열 주소	`&p` → 포인터의 주소
산술 연산	`arr + i` = `&arr[i]`	`p + i` = i * sizeof(type)
함수 매개변수	배열 크기 무시 → 포인터로 변환	포인터 그대로 사용

2.3 코드 예시

```
int arr[3] = {10, 20, 30};
int a = 100;

int *p = &a;

printf("%d\n", *(arr + 1)); // 20
printf("%d\n", *p);         // 100
```

3. 자료형(자료구조)

3.1 스택(STACK)

LIFO(Last In First Out)
장점: 간단, 재귀/undo
단점: 중간 접근 불가, 오버플로우
ADT: push, pop, peek, isEmpty, size

3.2 선형 큐(Linear QUEUE)

FIFO, 선형 저장
장점: 간단, 대기열에 유용
단점: 공간 비효율
ADT: enqueue, dequeue, peek, isEmpty, isFull, size

3.3 원형 큐(Circular QUEUE)

FIFO + 원형 구조
장점: 고정 크기 메모리 효율
단점: 상태 판단 복잡
ADT 동일

3.4 덱(DEQUE)

양쪽 삽입/삭제
장점: 유연성
단점: 복잡도
ADT: addFront, addRear, removeFront, removeRear, peekFront, peekRear, isEmpty, size

3.5 단순 연결 리스트(Simple Linked List)

노드: 데이터 + 포인터
장점: 삽입/삭제 용이
단점: 임의 접근 불가
ADT: insert, delete, get, isEmpty, size, search