알고리즘

2주차(17. 03. 13)

알고리즘 복잡도

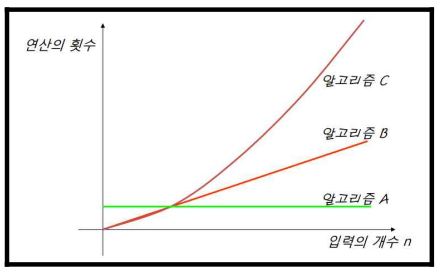
○ **알고리즘** : 컴퓨터로 문제를 풀기 위한 단계적인 절차

* **알고리즘 효율성**
  - 시간복잡도(수행시간) : 연산의 수행횟수는 고정된 숫자가 아니라 입력의 개수 n에 대한 함수
    // ex) 연산의 수 = 8 -> 3n+2
  - 공간복잡도(메모리)
* **유한성** : 한정된 수의 단계 후에는 반드시 종료되어야 한다. // 알고리즘과 프로그램의 차이

* **복잡도 분석의 예 (n을 n번 더하는 문제)**

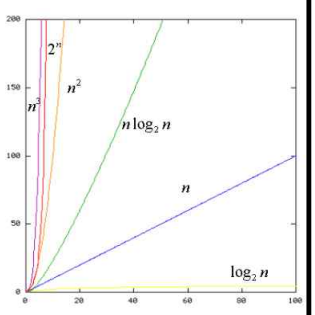
	알고리즘A	알고리즘B	알고리즘C	알고리즘D
코드	sum <- n*n;	sum <- 0; for i <- 1 to n do sum <- sum+n;	sum <- 0; for i <- 1 to n do for i <- 1 to n do sum <- sum+1;	sum <- 0; for i <- 1 to n for j <- 1 to n k <- A[1..n]에서 임의 n/2개 최대값 sum <- sum + k;
대입연산	1	1 + n - 1	1 + (n-1)*(n-1)	n*n*n/2 + 1
덧셈연산		n	n + n*(n-1)	n*n
곱셈연산	1			
나눗셈연산				
return	1	1	1	1
전체연산수	2 + 1(return)	2n + 1	$2n^2$ - 2n + 2 + 1(return)	$n^3/2 + n^2 + 1$ + 1(return)
빅오표기법	O(3) [상수시간대]	O(n)	$O(n^2)$	$O(n^3)$

* **상수시간대** : 입력 개수와 연산의 횟수가 고정되어 있는 것 // 알고리즘A



* **빅오 표기법** : 연산의 횟수를 대략적으로 표기한 것 / 데이터의 값에 따라 수행시간이 달라짐
  - 두 개의 함수 f(n)과 g(n)이 주어졌을 때, 모든 n≥n0에 대하여 |f(n)| ≤ c|g(n)|을 만족하는 2개의 상수 c와 n0가 존재하면 f(n)=O(g(n))이다.
  - 함수의 상한을 표시한다.(최고차항만 표시) // ex) n≥5 이면 2n+1 < 10n 이므로 2n+1 = O(n)
  - 빅오의 데이터 이상의 시간이 걸리지 않는다. // ex) $O(n^2)$

- O(1) : 상수형
- O(logn) : 로그형
- O(n) : 선형
- O(nlogn) : 로그선형
- $O(n^2)$: 2차형
- $O(n^3)$: 3차형
- $O(n^k)$: k차형
- $O(2^n)$: 지수형
- O(n!) : 팩토리얼형



시간복잡도	n					
	1	2	4	8	16	32
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
n!	1	2	24	40326	20922789888000	26313×10^{33}

* 빅오메가 표기법 // 점근적 시간 복잡도 : 많은 양의 데이터를 계산

- 모든 n≥n0에 대하여 |f(n)| ≥ c|g(n)|을 만족하는 2개의 상수 c와 n0가 존재하면 f(n)=Ω(g(n))이다
- 빅오메가는 함수의 하한을 표시한다.
- ex) n≥ 1 이면 2n+1 > n 이므로 n = Ω(n)

* 점근 성능의 효율성

- $1 < \log n < n < n^2 < n^2 \log n < n^3 < \ldots < 2^n$

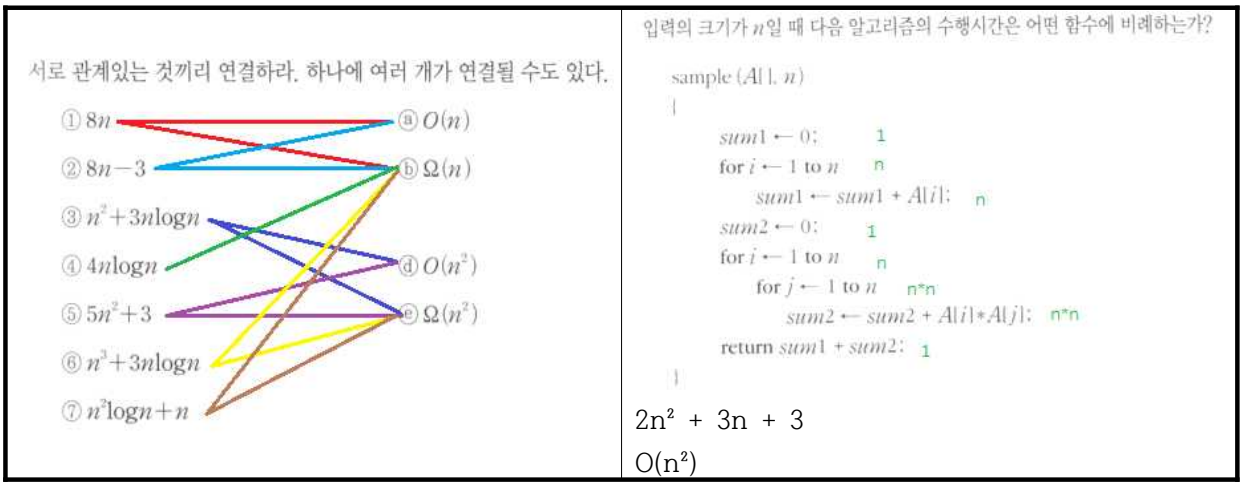
* 연습문제 0 : 코드의 각 명령문의 수행 횟수를 계산하여 시간 복잡도 함수를 계산하시오

시간 복잡도 구하기	시간 복잡도 구하기2	시간 복잡도 구하기3

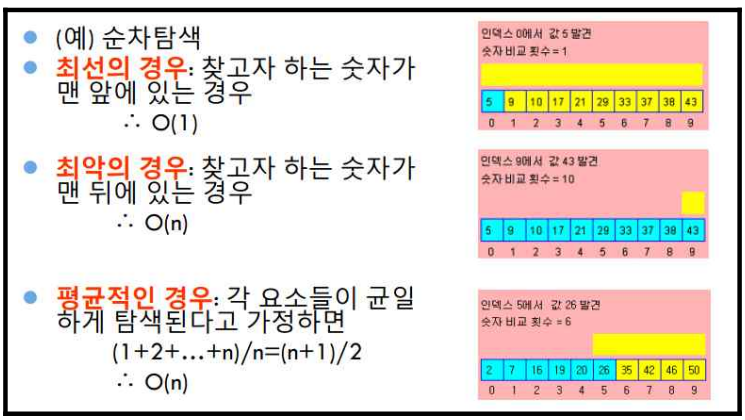
int test(int n){
  int i;
  int total;
 	
  total=1; // 1번
  for(i=1; i<n; I++){
  total *=n; // n-2번
  }
  return n; // 1번
}
	float sum(float list[], int n){
	  float tempsum;
	  int i;
	  tempsum=0; // 1번
	
	  for(i=0;i<n;i++){
	  tempsum+=list[i]; // n-1번
	  }
	  tempsum += 100; // 1번
	  tempsum += 200; // 1번
	  return tempsum; // 1번
	}
		int test(int n){
		  int i,b;
		
		  b=1; // 1번
		  i=1; // 1번
		  while(i <= n)
		  i = i*b; // logn 번
		}

1+n-2+1 = n 시간 복잡도 : O(n)번	1 + n-1 + 1 + 1 + 1 = n+3 시간 복잡도 : O(n)번	1+1+logn = logn+2 O(logn)번

* 연습문제



* 최선, 평균, 최악의 경우



순환 (순환(재귀) 알고리즘(함수)) : 자기 자신을 호출하는 함수(매개변수만 달라짐)

* 점화식 : 어떤 함수를 자신보다 더 작은 변수에 대한 함수와의 관계로 표현한 것
// $a_n = a_{n-1} + w$ -> f(n) = n f(n-1) -> f(n) = f(n-1) + f(-2)

○ 점화식 정리

(1) T(n) = T(n-1)+O(1), T(1) = O(1) => T(n) = O(n) //ex) 팩토리얼

(2) T(n) = T(n-1)+O(n), T(1) = O(1) => T(n) = $O(n^2)$

(3) T(n) = T(n/2)+O(1), T(1) = O(1) => T(n) = O(logn)

(4) T(n) = T(n/2)+O(n), T(1) = O(1) => T(n) = O(n) //ex) 가짜 동전 찾기

(5) T(n) = T(n/2)+O(1), T(1) = O(1) => T(n) = O(n)

(6) T(n) = T(n/2)+O(n), T(1) = O(1) => T(n) = O(nlogn) //ex) 병합 정렬 알고리즘

* T(값) 은 다음 함수에 들어가는 데이터의 개수
* O(1) : for문을 사용하지 않을 경우(마지막 return 값이 1일 경우)
* O(n) : for문을 이용하여 데이터 전체를 한 번씩 처리해야할 경우
* $O(n^2)$: for문을 이용하여 데이터 전체를 두 번씩 처리해야할 경우

○ 병합 정렬(merge sort) 알고리즘

* 데이터를 반으로 나누어 계산

(6) 병합정렬 수행시간

- 점화식으로 표현 : T(n) = 2T(n/2) + O(n), T(1) = O(1) / T(n) = O(nlogn)
- T(n) : 데이터의 개수가 n개 일 때, mergeSort 함수의 수행 시간
- T(1) = 1
- T(n) = 2T(n/2) + n

$= 2(2T(n/2^2) + n/2) + n = 2^2T(n/2^2) + 2 * n$

$= 2^2(2T(n/2^3) + n/2^2) + 2*n = 2^3T(n/2^3) + 3 * n$

...

$= 2^iT(n/2^i) + i*n$ // T(1) = 1을 알기 때문에 $n/2^i$을 1로 만들어야함.

// $n/2^i = 1 \Rightarrow n = 2^i \Rightarrow \log_2 n = \log_2 2^i = k$ // $\log_2$은 2와 나누어진다.

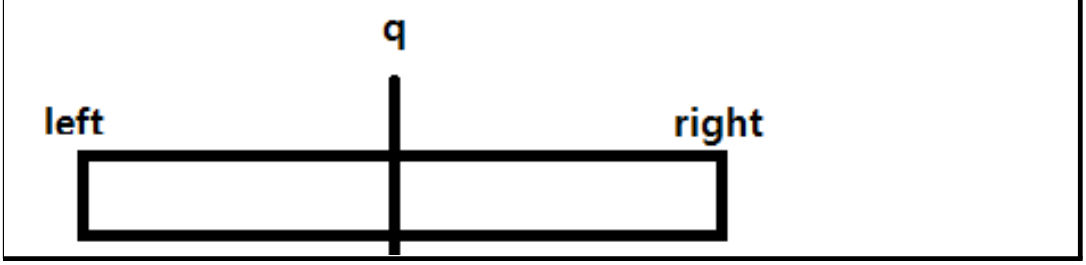
$= n*T(n/n) + \log_2 n * n$

$= n*T(1) + \log_2 n * n$

$= n+n\log n$

병합정렬 코드

```
mergeSort(A[ ], left, right) {
    if (left < right) then
    {
        q ← (left+right)/2; ---------- ① 중간 지점 계산
        mergeSort(A, left, q); ---------- ② 전반부 정렬
        mergeSort(A, q+1, right); ------- ③ 후반부 정렬
        merge(A, left, q, right); --------- ④ 병합
    }
}
```



(4) 가짜 동전 찾기 수행시간

- 점화식으로 표현 : T(n) = T(n/2) + O(n), T(1) = O(1) / T(n) = O(n)
- T(n) : 데이터의 개수가 n개 일 때, findFakeCoin(n) 함수의 수행 시간
- T(n) = T(n/2) + n + 1 => T(n/2) + n
  = (T(n/4) + n/2) + n = T(n/4) + n/2 + n
  = (T(n/8) + n/4) + n/2 + n = $T(n/2^3) + n/2^2 + n/2 + n$
  ...
  = $T(n/2^i) + n/2^{i-1} + n/2^{i-2} + \dots + n/2 + n$ // $n/2^{i-1}$에 2를 곱하면 $n/2^{i-2}$ 가 된다.
  = $T(1) + n * \{1/2^{i-1} + 1/2^{i-2} + \dots + 1/2 + 1\}$ // 공비수열
  // 공식 : 초항(r^n-1) / r-1
  = $T(1) + n * \{ 1/2^{i-1}(2^i-1) / 2-1 \}$
  = $T(1) + n * \{1/2^{i-1}(2^i-1)\}$ // $2^{i-1} = n/2$ 가 된다.
  = $1 + n * \{2/n(n-1)\}$
  = $1 + n * \{2-2/n\}$
  = $1 + 2n - 2$ => $2n - 1$
  = O(n)

(1) 팩토리얼 : 순환 알고리즘은 멈추는 부분과 순환 호출하는 부분으로 나누어진다.

- T(n) = T(n-1) + O(1)
  = (T(n-2) + 1) + 1
  = (T(n-3) + 1) + 1 + 1
  ...
  = (T(n-k))+k
  // n-k = 1 => n=k+1
  = (T(1+k-k)) + k
  = T(1) + k
  = T(1) + n - 1
  = n

(1) 하노이탑 : 기둥 세 개에 원반 옮기기 (작은 것이 큰 것 밑에 오면 안됨) // 2^n-1 번이 경우의 수

- T(n) = 2 * T(n-1) + O(1)
  = 2 * (2*T(n-2) +1) + 1
  = 2 * (2*(2*T(n-3) +1) + 1) + 1
  = 2^3T(n-3) +2^2 + 2 + 1
  ...
  = $\{2^i * T(n-i)\} + \{1 + 2 + \dots + 2^{i-1}\}$
  // n-i = 1 => n = i+1
  = $\{2^i * T(i+1-i)\} + \{1 * (2^i-1) / 2-1\}$ //초항(r^n-1) / r-1
  = $2^i * T(1) + (2^i-1)$
  = $2^{n-1} + 2^{n-1} - 1$
  = $2*2^{n-1} - 1$
  = $2^n - 1$
  = $O(2^n)$

정렬

○ 선택 정렬(Selection Sort)

* Max 값을 -1 하면서 하나하나 비교하여 스왑한다.

선택 정렬(Selection Sort)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void Swap(int &a, int &b){
        int tmp = a;
        a = b;
        b = tmp;
}

void Selection_Sort(int a[], int max){
        int maxindex;
        for (int i = n - 1; i >= 0; i--){
                maxindex = i;
                for (int j = i - 1; j >= 0; j--){
                        if (a[maxindex] < a[j])
                                maxindex = j;
                }
                Swap(a[i], a[maxindex]);
        }
}
```

```
void main(){
        srand((unsigned)time(NULL));
        int a[10];
        int max=9;

        printf("[ ");
        for (int i = 0; i<10; i++){
                a[i] = rand() % 101;
                printf("%2.d ", a[i]);
        } printf(" ]\n");

        Selection_Sort(a[], max);

        printf("[ ");
        for (int i = 0; i <= max; i++){
                printf("%2.d ", a[i]);
        } printf(" ]\n");

}
```

○ 버블 정렬(bubble sort)

* 근접한 데이터 크기를 비교하여 스왑한다.

버블 정렬(Bubble Sort)

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

void Swap(int &a, int &b){
        int tmp = a;
        a = b;
        b = tmp;
}

void Bubble_Sort(int a[], int max){
        for (int i = max; i > 1; i--){
                for (int j = 0; j < i; j++){
                        if (a[j] >= a[j + 1]){
                                Swap(a[j], a[j + 1]);
                                printf("[ ");
                        }
                }
        }
}
```

```
void main(){
        srand((unsigned)time(NULL));
        int a[10];
        int max=9;

        printf("[ ");
        for (int i = 0; i<10; i++){
                a[i] = rand() % 101;
                printf("%2.d ", a[i]);
        } printf(" ]\n");

        Bubble_Sort(a[], max);

        printf("[ ");
        for (int i = 0; i <= max; i++){
                printf("%2.d ", a[i]);
        } printf(" ]\n");

}
```

○ 병렬 정렬(merge sort)

* 재귀함수 사용 / Row와 High 가 같아지면 데이터가 하나이므로 종료

병렬 정렬(Merge Sort)

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

#define N 9

void Merge(int A[], int low, int mid, int high){
        int B[N+1];
        int LeftPtr, RightPtr, BufPtr;

        LeftPtr = low; RightPtr = mid + 1; BufPtr = low;
        while (LeftPtr <= mid && RightPtr <= high){
                if (A[LeftPtr] < A[RightPtr])
                        B[BufPtr++] = A[LeftPtr++];
                else B[BufPtr++] = A[RightPtr++];
        }
        if (LeftPtr > mid)
                for (int i = RightPtr; i <= high; i++)
                        B[BufPtr++] = A[i];
        else
                for (int i = LeftPtr; i <= mid; i++)
                        B[BufPtr++] = A[i];

// 합병이 끝나면 함수가 끝나기 때문에 넣어줘야한다.
        for (int i = low; i <= high; i++)
                A[i] = B[i];
}
```

```
void MergeSort(int A[], int low, int high){
        int mid;
        if (low < high){
                mid = (low + high) / 2;
                MergeSort(A, low, mid);
                MergeSort(A, mid+1, high);
                Merge(A, low, mid, high);
        }
}

void main(){
        srand((unsigned)time(NULL));
        int a[10];

        printf("[ ");
        for (int i = 0; i<10; i++){
                a[i] = rand() % 100;
                printf("%2.d ", a[i]);
        } printf(" ] / [START]\n\n");

        MergeSort(a, 0, N);

        printf("[ ");
        for (int i = 0; i <= N; i++){
                printf("%2.d ", a[i]);
        } printf(" ] / [END]\n");

}
```