**02. 재귀(Recursion)**

**02-1. 재귀 함수(Recursive Fuction)**

**# 함수 호출시 동작 과정**

: 스택 구조 - LIFO(Last In First Out)

**# 재귀함수**

: 자기 자신의 함수를 호출함으로싸, 반복적인 처리를 하는 함수

* 재귀함수 안에서 사용하는 변수는 지역변수
* Call by value
* **반드시 탈출(종료) 조건 명시!!(Stack Overflow 발생 주의)**

**# 계승(Factorial) 구하기**

* **반복적 정의 :** 함수 정의는 매개변수를 포함하나, 함수 자체는 포함하지 않음.

int Factorial(int num) {

int res = 1;

for (int i = 0; i <= num; i++)

res = res \* i;

return res;

}

int SUM(int num) {

int tot = 0;

for(int i = 1; i< num + 1; i++)

tot += i;

return tot; // O(n)

return num\*(num+1)/2 // O(1)

}

* **재귀적 정의 :** 함수가 자기 자신을 포함한다.

int Factorial(int num) {

if (num == 0)//재귀함수 탈출(종료) 조건

return 1;

return num \* Factorial(num - 1);

}

int SUM(int num) {

if (num < 0)

return 0;

return num + SUM(num - 1);

}

**2-2. 피보나치 수열**

폰트, 친필, 텍스트, 화이트이(가) 표시된 사진

자동 생성된 설명

int main(void) {

int num;

cout << "### 피보나치 수열 구하기###" << endl;

cout << "몇번째 수열까지 출력할까요? : "; cin >> num;

for (int i = 1; i <=num; i++) {

cout.width(8);

if (i % 5)cout << Fibo(i);

else cout << Fibo(i) << endl;

}

}

**# 피보나치 수열**

-> O(2^n) -> 문제점 : 중복 호출!! -> 재귀적 알고리즘은 지수함수에 비례하는 시간이 든다.

**- 피보나치수열 : 재귀적 용법**

int Fibo(int num) {

// 재귀 함수: 탈출 조건

if(num == 1 || num == 2)

return 1;

return Fibo(num - 1) + Fibo(num - 2);

}

**- 피보나치수열 : 비재귀적 용법**

int Fibo(int num) {

if(num == 1 || num == 2)

return 1;

int first, second, res = 0;

first = second = 1;

for(int i=3; i<=num; i++) {

res = first + second;

first = second;

second = res;

}

return res;

}

* **해결책 : 동적 프로그래밍**

**# 피보나치 수열 : 동적 프로그래밍**

-> Big theta n

int Fibo(int num) {

if (num == 1 || num == 2)

return 1;

//동적 메모리 할당

int\* pArr = new int[num];

if (pArr == NULL) {

cout << "메모리 할당 실패!!!" << endl;

exit(100);

}

int i, temp;

\*pArr = \*(pArr + 1) = 1;

for (i = 2; i <num ; i++) {//for문 범위 수정 <=/<, num+1->num

\*(pArr + i) = \*(pArr + i - 1) + \*(pArr + i - 2);

}

temp = \*(pArr + i - 1);

delete []pArr;

//동적 메모리 할당 해제가 루프안에 들어가 있던 것을, for문에 중괄호를 추가해서 구분

return temp;

}

**2-3. 실습보고서2\_연습문제 #01\_피보나치 수열 : 성능평가**

#include <iostream>

#include <ctime> // clock 또는 time 함수를 이용하여 코드의 실행 시간을 측정

using namespace std;

int Fibo(int num);

int main(void)

{

clock\_t start, finish;

double seconds;

int num;

cout << "20215229 이채훈" << endl;

cout << endl;

cout << "### 피보나치 수열 구하기 ###" << endl;

cout << endl;

cout << "몇 번째 수열까지 출력할까요? : "; cin >> num;

start = clock();

for (int i = 1; i <= num; i++) {

cout.width(8);

if (i % 5) cout << Fibo(i);

else cout << Fibo(i) << endl;

}

finish = clock();

seconds = (double)(finish - start) / CLOCKS\_PER\_SEC;

cout << num << "번째 피보나치 수열 계산 시간: " << seconds << endl;

return 0;

}

//재귀적 용법

int Fibo(int num) {

if (num == 1 || num == 2)

return 1;

return Fibo(num - 1) + Fibo(num - 2);

}

//비 재귀적 용법

int Fibo(int num) {

if(num == 1 || num == 2)

return 1;

int first, second, res = 0;

first = second = 1;

for(int i=3; i<=num; i++) {

res = first + second;

first = second;

second = res;

}

return res;

}

**# 연구조사**

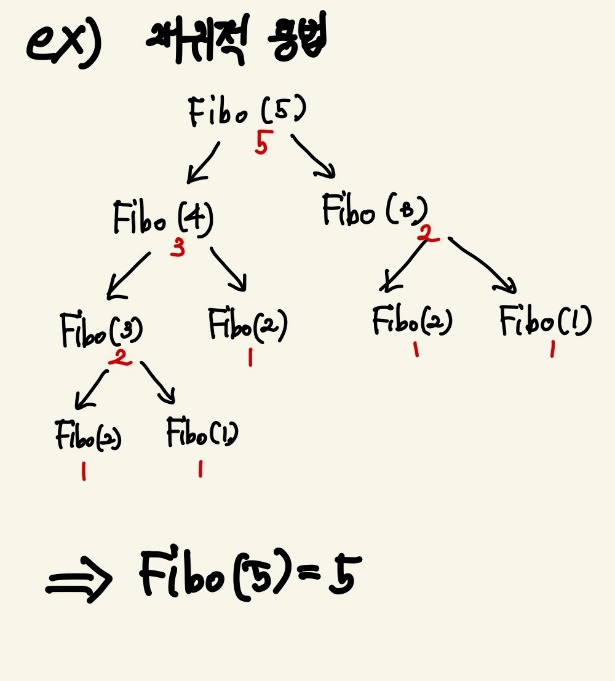
**피보나치 수열**

: n번째 항이 n-1번째와 n-2번째 항의 합인 수열

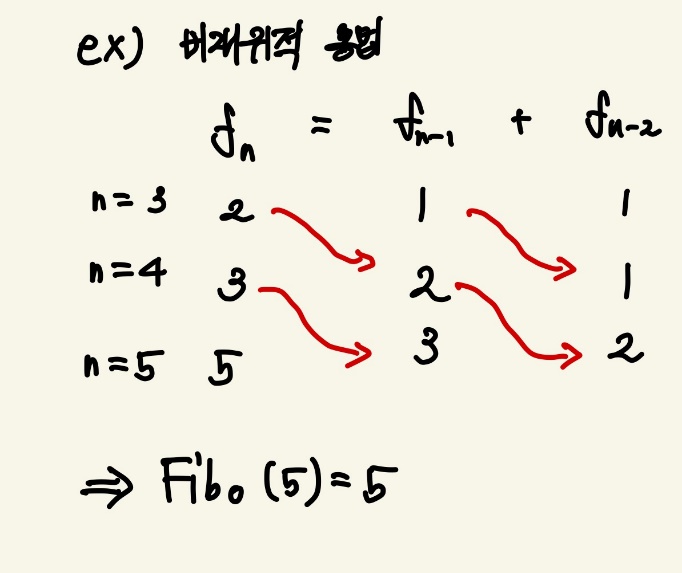
**폰트, 친필, 타이포그래피, 텍스트이(가) 표시된 사진

자동 생성된 설명**

**1. 재귀적 용법**



**2. 비재귀적 용법**



**결론**

두 가지 방법으로 구현한 피보나치 수열을 비교했을 때, 비재귀적 용법의 속도가 더 빠른 것을 확인할 수 있다. 이는 재귀적 용법을 사용해, 피보나치 수열을 구현할 경우 함수의 중복호출로 인해 O(2^n)의 지수함수에 비례하는 시간이 들기 때문이다.

**2-4. 중간고사 족보**

**Test1**

#include <iostream>

#define arrMaxSize 1024

using namespace std;

/\*

\* 마지막 항 안나옴;

\*/

int Fibo\_repeat(int num) {

int\* fn = new int[arrMaxSize];

fn[0] = 1;

fn[1] = 1;

for (int i = 2; i < num; i++) {

fn[i] = fn[i-1] + fn[i-2];

}

return fn[num-1];

}

int main() {

int n;

cout << " 몇 번째 수열까지 출력할까요? : ";

cin >> n;

for (int i = 1; i < n+1; i++) {

cout << Fibo\_repeat(i);

cout << " ";

if (i % 5 == 0) cout << '\n';

}

}