알고리즘 과제14 컴퓨터소프트웨어공학과 2-YA 20202296 전채린

[수업참여문제]

수업시간에 배운 7개의 알고리즘의 개념을 간단하게 요약하시오.  
n개의 자료를 갖는 배열에서 순차검색과 이진검색

(1) 순차검색

배열에서 찾고자 하는 값을 맨 앞에서부터 끝까지 차례대로 찾아 나가는 방법

(2) 이진검색

정렬된 배열에서 거듭 이분하여 원하는 자료를 찾는 방법

n개의 자료를 갖는 배열에서 오름차순으로 정렬하기 위한 정렬방법(버블, 삽입, 선택, 퀵, 합병)

(3) 버블 정렬(Bubble Sort)

인접한 두 자료(a[i]와 a[i+1])을 비교하여 오름차순으로 저장되어 있지 않으면 교환하는 방법

(4) 삽입 정렬(Insert Sort)

i개의 데이터(a[0]~a[i-1])가 이미 정렬되어 있다면 다음 데이터 a[i]를 오름차순을 유지하도록 삽입하는 방법

(5) 선택 정렬(Selection Sort)

정렬되지 않은 정수들 중에서 가장 작은 값을 찾아서 이미 정렬된 리스트의 다음 자리에 놓는 방법

(6) 퀵 정렬(Quick Sort)

분할정복 방법으로 어떤 제어 값을 중심으로 두 개의 데이터 집합으로 분할하여, 두개의 데이터 집합에 대하여 다시 재귀적으로 quick\_sort함수를 불러 처리하는 방법으로, 위치의 값을 교환하여 작은 값은 앞으로 모으고 큰 값은 뒤로 모은다.

(7)합병 정렬(Merge Sort)

분할정복 방법으로 문제를 두 개의 문제로 나누고 각 각 해결한 후 두 개의 결과를 합해 원래의 문제를 해결하는 방법으로, 두 개의 배열에서 데이터를 비교해 새로운 배열에 오름차순으로 저장하는 방법으로 재귀기법을 이용하여 문제를 해결한다.

[기말예상문제]

상태도 분석을 통한 알고리즘의 이해

1. 다음 실수 단순 계산기 프로그램의 함수 real\_calculator()와 get\_real() 코드를 완성하고 사용한 status 변수의 역할을 설명하고, 다음 실행 화면의 입력을 예로 이에 해당하는 STD(상태전이도)를 그리시오.

(1) 함수 real\_calculator() 코드 완성

void real\_calculator(void)

{

double operand1 = 0, operand2 = 0, result;

int op, c = '0';

int current\_stage = STAGE1, status = 0;

while (c != 'x') {

c = getchar();

if (current\_stage == STAGE1) {

if (is\_digit(c)) {

operand1 = get\_real(operand1, c, status);

if (status != 0) status++;

}

else if (is\_operator(c)) {

op = c;

current\_stage = STAGE2;

status = 0;

operand2 = 0;

}

else if (c == '.') status = 1;

} // STAGE1

else if (current\_stage == STAGE2) {

if (is\_digit(c)) {

operand2 = get\_real(operand2, c, status);

if (status != 0) status++;

}

else if (c == '.') status = 1;

else if (c == '\n') {

result = real\_operation(op, operand1, operand2);

printf("= %.3f", result);

current\_stage = STAGE3;

status = 0;

}

} // STAGE2

else if (current\_stage == STAGE3) {

if (c == '\n') {

operand1 = 0;

current\_stage = STAGE1;

}

else if (is\_operator(c)) {

op = c;

current\_stage = STAGE2;

operand1 = result;

operand2 = 0;

}

} // STAGE3

} // while

}

(2) 함수 get\_real() 코드 완성

double get\_real(double current\_operand, char c, int status)

{

int part1;

double part2;

if (status == 0) {

part1 = c - '0';

current\_operand \*= 10;

current\_operand += part1;

}

else {

part2 = c - '0';

part2 = part2 \* pow(10, -status);

current\_operand += part2;

}

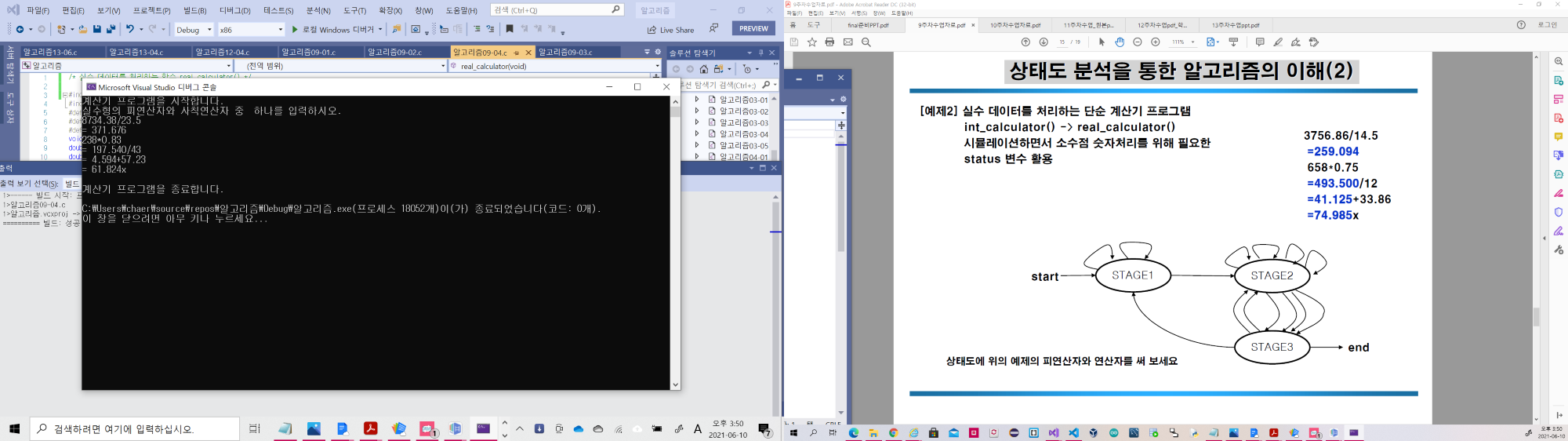
return current\_operand;

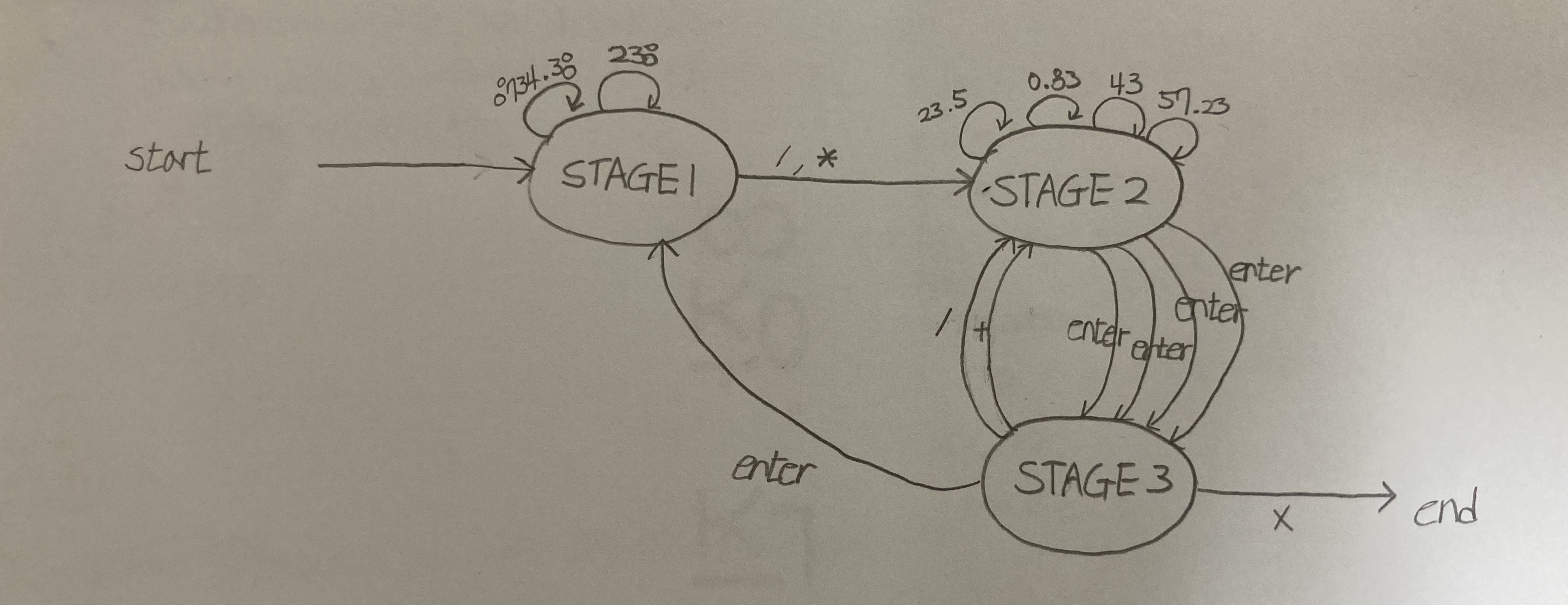
}

(3) status 변수의 역할

소수점 숫자처리를 위해 status 변수를 활용하는데 status는 소수점의 자릿수를 알려준다. status는 0으로 초기화되는데 소수점을 만나면 status를 1로 변경한다. 그 후엔 반복처리 하면서 소수점의 자릿수만큼 status의 값을 증가시켜준다.

(4) STD(상태전이도)





스택을 이용한 알고리즘

2. 다음의 중위표기식을 스택의 변화를 나타내며 후위표기식으로 변경하고, 변경된 후위표기식을 배열 pexpr에 읽어들여 계산하는 함수 cal() 코드를 완성하고 표를 채우시오.

중위표기식 : 5\*(((9+7)+6/3)\*5-2)/2

(1) 스택의 변화를 나타내며 후위표기식으로 변경

후위표기식 : 5 9 7 + 6 3 / + 5 \* 2 - \* 2 /

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| + | / |  |  |  |  |
| ( | + |  |  |  |  |
| ( | ( | \* | - |  |  |
| ( | ( | ( | ( |  |  |
| \* | \* | \* | \* | \* | / |

(2) 함수 cal() 코드 완성

int cal(void)

{

char symbol;

int op1, op2, n = 0;

top = -1;

symbol = pexpr[n++];

while (symbol != '\0') {

if (is\_operator(symbol)) {

// 피연산자를 삭제하여 연산을 수행한 후, 그 결과를 스택에 삽입한다.

op2 = delete\_stack();

op1 = delete\_stack();

switch (symbol) {

case '+':

add\_stack(op1 + op2);

break;

case '-':

add\_stack(op1 - op2);

break;

case '\*':

add\_stack(op1 \* op2);

break;

case '/':

add\_stack(op1 / op2);

} // switch

}

else

add\_stack(symbol - '0'); // 스택에 삽입

symbol = pexpr[n++];

}

return delete\_stack(); // 결과 반환

}

(3) 위의 cal() 함수를 통해 변경된 후위표기식을 계산하는 과정을 표로 완성

|  |  |  |  |
| --- | --- | --- | --- |
| n | symbol | cal()에서 수행한 함수 | top |
| **0, 1, 2** | **‘5’, ‘9’, ‘7’** | **add\_stack(5), add\_stack(9), add\_stack(7)** | **2** |
| **3** | **‘+’** | **2delete\_stack() => 7, 9, add\_stack(16)** | **1** |
| **4, 5** | **‘6’, ‘3’** | **add\_stack(6), add\_stack(3)** | **3** |
| **6** | **‘/ ’** | **2delete\_stack() => 3, 6, add\_stack(2)** | **2** |
| **7** | **‘+’** | **2delete\_stack() => 2, 16, add\_stack(18)** | **1** |
| **8** | **‘5’** | **add\_stack(5)** | **2** |
| **9** | **‘\*’** | **2delete\_stack() => 5, 18, add\_stack(90)** | **1** |
| **10** | **‘2’** | **add\_stack(2)** | **2** |
| **11** | **‘-’** | **2delete\_stack() => 2, 90, add\_stack(88)** | **1** |
| **12** | **‘\*’** | **2delete\_stack() => 88, 5, add\_stack(440)** | **0** |
| **13** | **‘2’** | **add\_stack(2)** | **1** |
| **14** | **‘/ ’** | **2delete\_stack() => 2, 440, add\_stack(220)** | **0** |

결과 값 : 220

자기참조구조체를 이용한 이진탐색트리 구축 및 활용

3. 다음은 사원번호(정수형), 입사년도(정수형), 인사등급(문자형)을 자료로 가지는 이진검색트리의 노드 구조(ENODE)이다. 입력된 정보를 사원번호를 기준으로 내림차순으로 이진 검색트리를 구축하고, 함수 search(), insert\_node(), inorder(), year\_search() 코드를 완성하시오.

typedef struct node {

struct node \*left;

int eid;

int syear;

char grade;

struct node \*right;

} ENODE;

(1) 함수 serch() 코드 완성

ENODE\* search(ENODE\* root, int key)

{

ENODE\* tptr = root;

while (tptr) {

if (key < tptr->eid)

tptr = tptr->left;

else if (key > tptr->eid)

tptr = tptr->right;

else // found

return tptr;

}

return NULL; // not found

}

(2) 함수 insert\_node() 코드 완성

int insert\_node(ENODE\* root, int id, int year, char score)

{

ENODE\* tptr = root, \* before;

while (tptr) {

if (id < tptr->eid) {

before = tptr;

tptr = tptr->left;

}

else if (id > tptr->eid) {

before = tptr;

tptr = tptr->right;

}

else // found

return 0;

}

tptr = (ENODE\*)malloc(sizeof(ENODE));

tptr->eid = id;

tptr->syear = year;

tptr->grade = score;

tptr->left = tptr->right = NULL;

if (id < before->eid) before->left = tptr;

else before->right = tptr;

return 1;

}

(3) 함수 inorder() 코드 완성

void inorder(ENODE\* ptr)

{

if (ptr) { // 내림차순으로 구축된 사원정보

inorder(ptr->right);

printf("%d\t%d\t%c\n", ptr->eid, ptr->syear, ptr->grade);

inorder(ptr->left);

}

}

(4) 함수 year\_search() 코드 완성

void year\_search(ENODE\* ptr, int year)

{

if (ptr) {

year\_search(ptr->left, year);

if ((ptr->syear) == year)

printf("%d\t%c\n", ptr->eid, ptr->grade);

year\_search(ptr->right, year);

}

}

정렬알고리즘(버블, 삽입, 퀵, 합병)

4. 정렬 알고리즘인 버블, 삽입, 퀵, 합병 함수의 코드를 완성하시오.

(1) 다음은 각 행에 도서코드와 그 도서의 재고개수를 임의의 순으로 저장하고 있는 파일(bitem.d)로부터 n\*2의 2차원배열에 데이터를 읽어 들인다. 버블 정렬에 의하여 도서코드의 오름차순으로 정렬하도록 함수 bubble()을 완성하시오.

void bubble(unsigned a[][2], int n)

{

int i = n - 1, j, flag = 1;

unsigned tmp0, tmp1;

while (flag && i != 0) {

flag = 0;

for (j = 0; j <= i - 1; j++) {

if (a[j][0] > a[j + 1][0]) {

flag = 1;

tmp0 = a[j][0];

a[j][0] = a[j + 1][0];

a[j + 1][0] = tmp0;

tmp1 = a[j][1];

a[j][1] = a[j + 1][1];

a[j + 1][1] = tmp1;

}

}

i--;

}

}

(2) 다음 삽입 정렬 함수 insertion()을 완성하고 표를 채우시오.

a = {10, 25, 30, 44, 45, 15, 7, 32, 62}, n=9

void insertion(int a[], int n)

{

int i, j, key;

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

key = a[i];

j = i-1;

while (a[j] > key && j >= 0) {

a[j+1] = a[j]; j--;

}

a[j+1] = key;

}

}

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n=9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| a | 10 | 25 | 30 | 44 | 45 | 15 | 7 | 32 | 62 |
| i=4 | 10 | 15 | 25 | 30 | 44 | 45 | 7 | 32 | 62 |
| i=5 | 7 | 10 | 15 | 25 | 30 | 44 | 45 | 32 | 62 |
| i=6 | 7 | 10 | 15 | 25 | 30 | 32 | 44 | 45 | 62 |
| i=7 | 7 | 10 | 15 | 25 | 30 | 32 | 44 | 45 | 62 |

(3) 퀵 정렬에 의하여 사원번호를 기준으로 오름차순으로 정렬하는 함수 quick\_sorta()와 영어성적을 기준으로 내림차순으로 정렬하는 함수 quick\_sortd()를 완성하시오.

typedef struct employee {

long eid;

char name[20];

int escore;

char dname[5];

} edatatype;

void quick\_sorta(edatatype a[], int left, int right)

{

long pivot;

int i, j;

edatatype tmp;

if (left < right) {

i = left; j = right + 1;

pivot = a[left].eid;

while (i < j) {

do

i++;

while ((a[i].eid < pivot) && (i < right));

do

j--;

while ((a[j].eid > pivot) && (j > left));

if (i < j) {

tmp = a[i]; a[i] = a[j]; a[j] = tmp;

}

}

if (j != left) {

tmp = a[j]; a[j] = a[left]; a[left] = tmp;

}

quick\_sorta(a, left, j - 1);

quick\_sorta(a, j + 1, right);

}

}

void quick\_sortd(edatatype a[], int left, int right)

{

int pivot;

int i, j;

edatatype tmp;

if (left < right) {

i = left; j = right + 1;

pivot = a[left].escore;

while (i < j) {

do

i++;

while (a[i].escore > pivot) && (i < right));

do

j--;

while ((a[j].escore < pivot) && (j > left));

if (i < j) {

tmp = a[i]; a[i] = a[j]; a[j] = tmp;

}

}

if (j != left) {

tmp = a[j]; a[j] = a[left]; a[left] = tmp;

}

quick\_sortd(a, left, j - 1);

quick\_sortd(a, j + 1, right);

}

}

(4) 다음 mege2()는 합병정렬 알고리즘의 개념을 이용하여 2개의 데이터 리스트 a1과 a2를 합쳐 새로운 리스트 a를 얻고 구현한 함수 merge에 다음과 같은 a1과 a2의 데이터가 주어진 경우 while (i < n1 && j < n2) 문장을 빠져 나왔을 때, 출력되는 i, j, k의 값을 쓰시오.

a1[]={121, 322, 455, 525, 550, 600, 650} a2[]={100, 200, 400, 500, 640, 700, 900, 950}

int merge2(int a1[], int a2[], int a[], int n1, int n2)

{

int i=0, j=0, k=0;

**while (**i < n1 && j < n2**)** {

if (a1[i] < a2[j])

a[k++] = a1[i++];

else

a[k++] = a2[j++];

}

**printf("%d == %d == %d\n", i, j, k);**

if (i == n1)

while (j < n2) a[k++] = a2[j++];

else

while (i < n1) a[k++] = a1[i++];

return k;

}

7 == 5 == 12

i = 7, j = 5, k = 12

알고리즘의 성능분석 big-O 개념

5. 다음 7개의 알고리즘에 대응되는 big-O를 적으시오.

(1) 순차검색 : O(n)

(2) 이진거색 : O(logn)

(3) 버블정렬 : O(n^2)

(4) 삽입정렬 : O(n^2)

(5) 선택정렬 : O(n^2)

(6) 퀵정렬 : O(nlogn)

(7) 합병정렬 : O(nlogn)