**REPORT**

**[Prac 3]**



**과 목 : 알고리즘06**

**담당교수 : 주종화 교수님**

**학 과 : 컴퓨터공학과**

**학 번 : 2021111971**

**이 름 : 이재혁**

텍스트, 클립아트이(가) 표시된 사진

자동 생성된 설명

**프로그램 구현 환경**

프로그래밍언어 : C++

텍스트 에디터 : Visual Studio Code

운영체제 : MAC OS

**문제 1. 회문**

**의사코드**

|  |  |
| --- | --- |
| **비순환적 확인** | **순환적 확인** |
| **텍스트, 친필, 폰트, 문서이(가) 표시된 사진  자동 생성된 설명** | **텍스트, 친필, 폰트, 문서이(가) 표시된 사진  자동 생성된 설명** |

**프로그램 설계**

회문을 판별하는 Palindrome 클래스를 작성해 비순환적, 순환적 멤버 메소드로 파일에 존재하는 문자열이 회문인지 판별한다.

|  |  |
| --- | --- |
| 텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진  자동 생성된 설명 | 파일 : 최대 100개의 문자가 쓰여져 있으며 문자열은 줄 바꿈으로 구분되어있다. |

회문 판단 기준: 문자열을 뒤집어 “읽어도” 같은 문자열

* 반대로 읽어도 같은 알파벳 순서인지 확인
* 대소문자 구분 X, 공백 무시

**멤버 변수**

**content** : 파일에서 모든 문자를 읽어와 저장한 배열 배열의 크기는 파일의 최대크기인 100이다.

**멤버 메소드**

**생성자** : 입력 스트림 객체로 파일의 모든 문자를 읽어 content 배열에 저장한다.

**print()** : 현재 배열을 출력하는 메소드

**isSentence(int n)** : content 배열의 n번 index의 값이 문자 혹은 공백인지 확인

* 문자열을 구분하기위해 사용

**it\_check()** : 비순환적 확인,

content에 저장 된 문자를 전부 확인

줄 바꿈을 만났다면 이전까지의 문자들이 하나의 문자열

문자열을 복사해 회문인지 확인

문자열 가장 왼쪽과 오른쪽에 있는 알파벳부터 확인하며 중간 지점 까지 같은 문자인지 반복확인

**re\_ check (int n)** : 순환적 확인

n : content 배열에서 새로운 문자열의 시작 index

n번 index부터 줄 바꿈 문자까지 하나의 문자열로 처리

후 회문인지 판별

다음 문자열의 시작 index로 메소드 재귀호출

**소스코드**

// 2021111971 이재혁 문제 1

#include <fstream>

#include <iostream>

using namespace std;

class Palindrome {

public:

Palindrome() { // 파일로 부터 문자열을 불러와 배열에 저장한다.

ifstream fin;

fin.open("실습3.txt");

int i = 0;

char c;

while ((c = fin.get()) != EOF) {

content[i] = c;

i++;

}

// 파일의 내용을 전부 읽은 후 줄바꿈을 기준으로 문자열을 나눠

// 회문을 판별

}

void it\_check() { // 비 순환적 확인 (반복문)

// 회문 : 문자열을 뒤집어 "읽어도" 같은 문자열

// -> 반대로 읽어도 같은 알파벳 순서인지 확인 (대소문자 구분 X, 공백

// 무시)

int start = 0; // 문자열의 시작 index

while (1) {

char str[100]; // 파일에 있는 문자열

int i;

for (i = 0; this->isSentence(start + i); i++) {

str[i] = content[start + i];

} // 문자열의 시작 index부터 줄바꿈까지 하나의 문자열로 저장

for (int j = 0; j < i; j++) {

if ((int)str[j] > 64 && (int)str[j] < 91) {

str[j] = (char)((int)str[j] + 32);

}

} // 복사한 문자열을 모두 소문자로 통일

int isPalindrome = 1; // 회문인지 판별할 변수

// 문자열을 절반으로 나눴을 때

int left = 0; // 왼쪽을 확인할 index

int right = i - 1; // 오른쪽을 확인할 index

while (1) {

if (str[left] != str[right]) {

isPalindrome = 0;

break;

} // 문자열의 가장 왼쪽과 가장 오른쪽이 다르다면 회문 X

while (str[++left] == ' ') {

}

while (str[--right] == ' ') {

}

// 다음 문자를 확인할 때, 다음번 문자만 비교 -> 공백 무시

if (left >= right) {

break;

}

// 왼쪽 오른쪽 index가 교차되었다면 확인 완료

// left == right -> 문자열의 길이 홀수

// left > right -> 문자열의 길이 짝수

}

if (isPalindrome) { // 변수의 값이 여전히 1 이면 회문

// 저장한 문자열을 전부 소문자로 통일했기 때문에 원본의 문자열

// 출력

for (int j = start; j < start + i; j++) {

cout << content[j];

}

cout << " is Palindrome" << endl;

} else {

for (int j = start; j < start + i; j++) {

cout << content[j];

}

cout << " is not Palindrome" << endl;

}

start = start + i + 1;

// i가 잘라낸 문자열의 끝

// 끝 다음문자는 다음 문자열의 시작,

// 다음 반복을 위해 시작점 수정

if (!(this->isSentence(start))) {

// 시작점이 알파벳이나, 공백문자가 아니라면, 파일의 끝

// 회문 판별 종료

break;

}

}

}

void re\_check(int n) { // 순환적 확인 (재귀호출)

// n번째 index의 값부터 문자열 판별 시작

// n번째 index 부터 '\n'까지가 하나의 문자열

if (!(this->isSentence(n))) {

// n번째 index의 값이 알파벳 또는 공백이 아니라면, 파일의 끝

// 회문 판별 종료

return;

}

// 비 순환적 확인과 같은 방식, n = start

while (1) {

char str[100];

int i;

for (i = 0; this->isSentence(n + i); i++) {

str[i] = content[n + i];

}

int isPalindrome = 1;

int left = 0;

int right = i - 1;

while (1) {

if (str[left] != str[right]) {

isPalindrome = 0;

break;

}

while (str[++left] == ' ') {

}

while (str[--right] == ' ') {

}

if (left >= right) {

break;

}

}

if (isPalindrome) {

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

cout << content[j];

}

cout << " is Palindrome" << endl;

} else {

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

cout << content[j];

}

cout << " is not Palindrome" << endl;

}

n = n + i + 1; // 지금 확인한 다음 index로 시작점 조정

if (!(this->isSentence(n))) {

break;

}

}

re\_check(n); // 다음 index를 시작으로 재귀호출

}

void print() {

// 문자열이 비어있지 않은 동안 값을 출력

for (int i = 0; content[i]; i++) {

cout << content[i];

}

cout << endl;

}

int isSentence(int n) {

if ((content[n] > 64 && content[n] < 91) ||

(content[n] > 96 && content[n] < 123) || content[n] == ' ') {

return 1;

} else

return 0;

}

private:

// 파일로 부터 최대 100개의 글자를 읽어 회문 판별

// 문장은 줄바꿈으로 구분

char content[100];

};

int main() {

Palindrome p;

cout << "------- 읽어들인 문자열 -------" << endl;

p.print();

cout << "-------- 순환적 확인 --------" << endl;

p.it\_check();

cout << "------ 비 순환적 확인 -------" << endl;

p.re\_check(0);

}

**결과 분석**

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

파일 (실습.txt)의 문자들을 모두 읽어 들인 것을 확인할 수 있다.

순환적, 비순환적 확인 모두 회문을 판별할 수 있다.

회문의 판별기준에 따라

ABbA -> abba

stepon no pets -> steponnopets

로 문자열을 수정해 회문을 적절히 판별했다.

**문제 2 : 힙 생성**

**힙 생성 과정**

|  |
| --- |
| **1 2**  **텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진  자동 생성된 설명 텍스트, 스크린샷, 도표이(가) 표시된 사진  자동 생성된 설명** |
| **3 4**  **텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진  자동 생성된 설명 텍스트, 스크린샷, 원, 번호이(가) 표시된 사진  자동 생성된 설명** |

**힙 정렬 과정**

|  |  |  |
| --- | --- | --- |
| **텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진  자동 생성된 설명** | **텍스트, 스크린샷이(가) 표시된 사진  자동 생성된 설명** | **텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진  자동 생성된 설명** |

힙의 최댓값을 가장 마지막 원소와 교환 후 배열의 크기를 1 줄인다.

마지막에 있었던 원소가 최댓값의 위치에 들어가 있으므로, 힙을 조정한다. (자식 노드 중 큰 값과 교환해 간다)

**프로그램 설계**

힙을 다루는 Heap class를 정의해 힙 생성, 힙 정렬을 수행한다.

**멤버 변수**

**arr** : 길이가 10인 배열, 10개의 값을 입력 받아 저장한다.

**멤버 메소드**

**생성자** : 사용자로부터 10개의 값을 입력 받는다.

**print()** : 현재 배열 출력

**buildHeap ()** : 현재 저장하고 있는 배열을 최대 힙으로

구성한다.

원소를 앞에서부터 하나씩 선택해 힙의 형태를

만족시키도록 배열에 삽입한다.

배열의 N번 index의 원소를 삽입할 때

N-1번까지의 배열은 MAX힙으로 구성되어있다.

현재 삽입하는 노드의 위치가 왼쪽인지 오른쪽인지를 판별해 적절한 부모 노드와 값을 비교하고 교환한다.

**HeapSort()** : 저장하고 있는 최대 힙으로 정렬을 시행한다.

현재 힙의 최댓값을 힙의 맨 마지막 원소와

교환한다.

맨 마지막 원소가 힙의 최댓값 자리에 위치한다.

자식 노드 중 큰 값과 비교해 자식 노드가 크다면

값을 교환한다. 자식 노드의 값이 작을 때 까지

반복한다.

**소스코드**

// 2021111971 이재혁 문제 2

#include <iostream>

using namespace std;

class Heap {

public:

// 배열의 값 10개를 입력받아 arr 배열에 저장

Heap() {

cout << "배열의 값 10개 입력" << endl;

for (int i = 0; i < 10; i++) {

cin >> arr[i];

}

}

// 배열의 현재 상태를 출력

void print() {

for (int i = 0; i < 10; i++) {

cout << arr[i] << " ";

}

cout << endl;

}

// 주어진 배열을 힙 구조로 만드는 메소드

void buildHeap() {

// 0부터 n-1까지의 배열을 힙 구조로 변환

// 앞에서 부터 노드를 삽입하며 현재 삽입한 노드가 부모 노드보다 크다면

// 원소교환

for (int insert = 0; insert < 10; insert++) {

int child = insert;

while (child > 0) {

switch (child % 2) // 현재 삽입할 노드의 위치 확인

{

case 1: // 현재 삽입할 노드가 왼쪽 : parent\_index = child/2

if (arr[child / 2] < arr[child]) {

int tmp = arr[child / 2];

arr[child / 2] = arr[child];

arr[child] = tmp;

}

child /= 2;

break;

case 0:

// 현재 삽입할 노드가 오른쪽 : parent\_index = child/2 - 1

if (arr[child / 2 - 1] < arr[child]) {

int tmp = arr[child / 2 - 1];

arr[child / 2 - 1] = arr[child];

arr[child] = tmp;

}

child = child / 2 - 1;

break;

}

}

this->print();

}

}

void adjustHeap(int n) {

// 최대 값을 제거한 후 Heap을 조절하는 메소드

// 자식 노드의 값 중 큰 값과 비교해 원소를 조정

int root = 0;

while (root \* 2 + 1 < n) {

int child = root \* 2 + 1;

if (arr[child] < arr[child + 1])

child++;

if (arr[root] < arr[child]) {

int tmp = arr[root];

arr[root] = arr[child];

arr[child] = tmp;

}

root = child;

}

}

void HeapSort() {

// 힙에서 최대값을 계속해서 꺼내 배열의 맨 마지막으로 이동

// 배열의 크기를 하나씩 줄여가며 그 다음 큰 값은

// 이전 정렬 때 뽑은 가장 큰 값 앞에 위치

for (int i = 9; i > 0; i--) {

// 현재 배열의 최댓값인 처음 원소를 마지막으로 이동

int tmp = arr[0];

arr[0] = arr[i];

arr[i] = tmp;

// 배열의 마지막 원소를 제외한 부분을 다시 힙 재구성

this->adjustHeap(i - 1);

this->print(); // 힙을 다시 재구성 후 배열 확인

}

// 원소 2개가 남았을 때 0번 index의 값이 더 크므로 (MAX HEAP)

// 자식 원소와 교환 후 정렬 종료

int tmp = arr[0];

arr[0] = arr[1];

arr[1] = tmp;

cout << "------- 정렬완료 --------" << endl;

this->print(); // 정렬이 완료된 배열 확인

}

private:

int arr[10]; // 배열을 저장하는 멤버 변수

};

int main() {

Heap h; // Heap 객체 생성 및 배열 입력

cout << "------- 힙 구조 생성 -------" << endl;

h.buildHeap(); // 입력된 배열을 힙 구조로 변환

cout << "---------- 힙 정렬 --------" << endl;

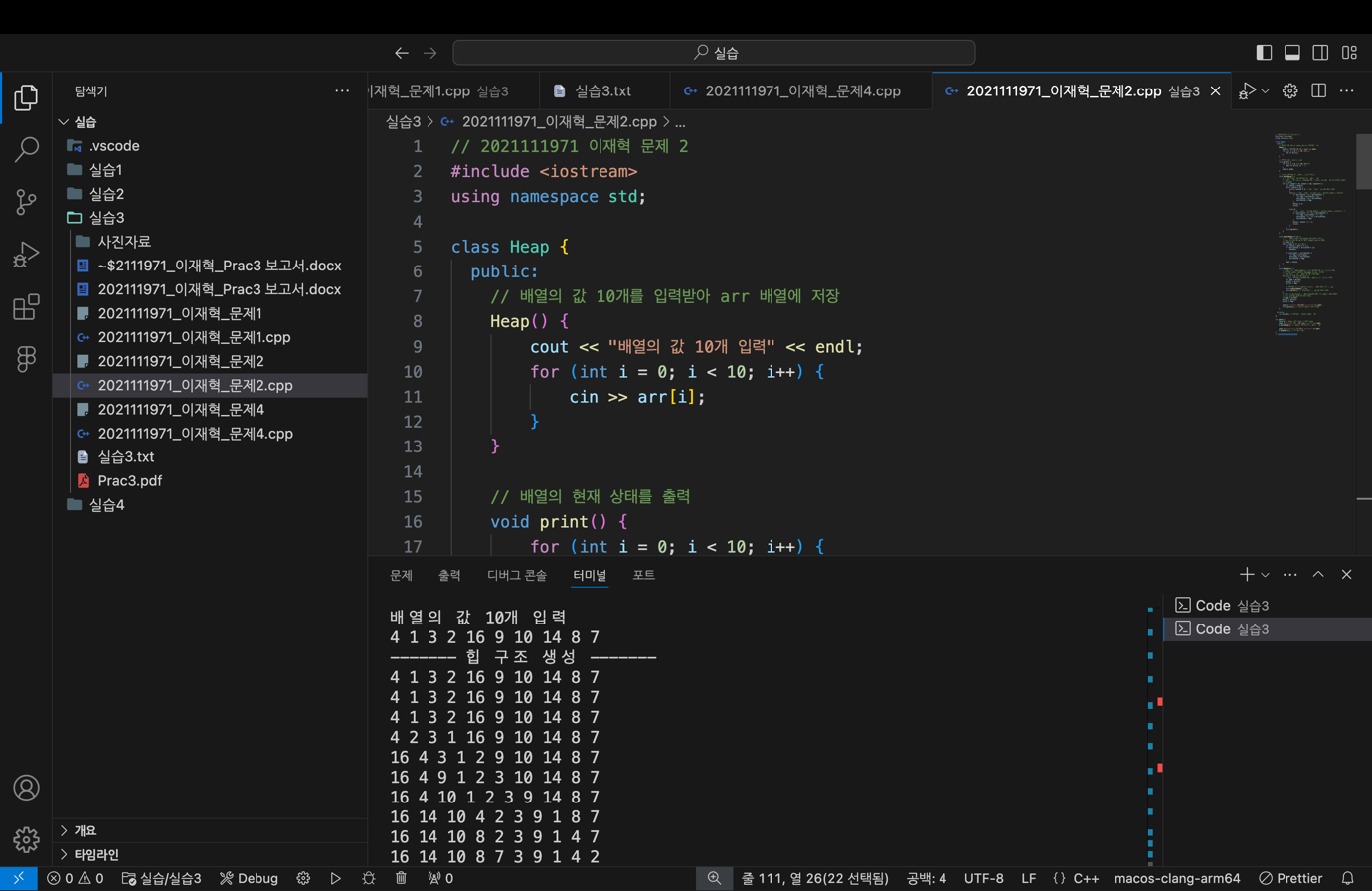
h.HeapSort(); // 힙 정렬 수행

}

// 4 1 3 2 16 9 10 14 8 7

**결과 분석**

힙 생성



현재 주어진 원소의 순서대로 앞쪽 index부터 힙을 구성한다.

4 1 3 까지는 힙의 성질을 만족하므로 배열에 변화가 없다.

2가 들어왔을 때 부모 노드인 1보다 큰 값이므로 원소교환이

발생한다.

16이 들어왔을 때 부모 노드인 2보다 큰 값이므로

원소를 교환 후, 여전히 부모 노드인 4보다 큰 값이므로 원소를 교환해 최대값 노드에 위치한다.

위와 같은 과정으로 최대 힙을 생성한다.

힙 정렬

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

최대 힙에서 가장 큰 값을 배열의 맨 마지막 원소와 교환한다.

맨 처음 정렬에서 16과 2의 값이 교환된다.

2의 위치를 조절한다.

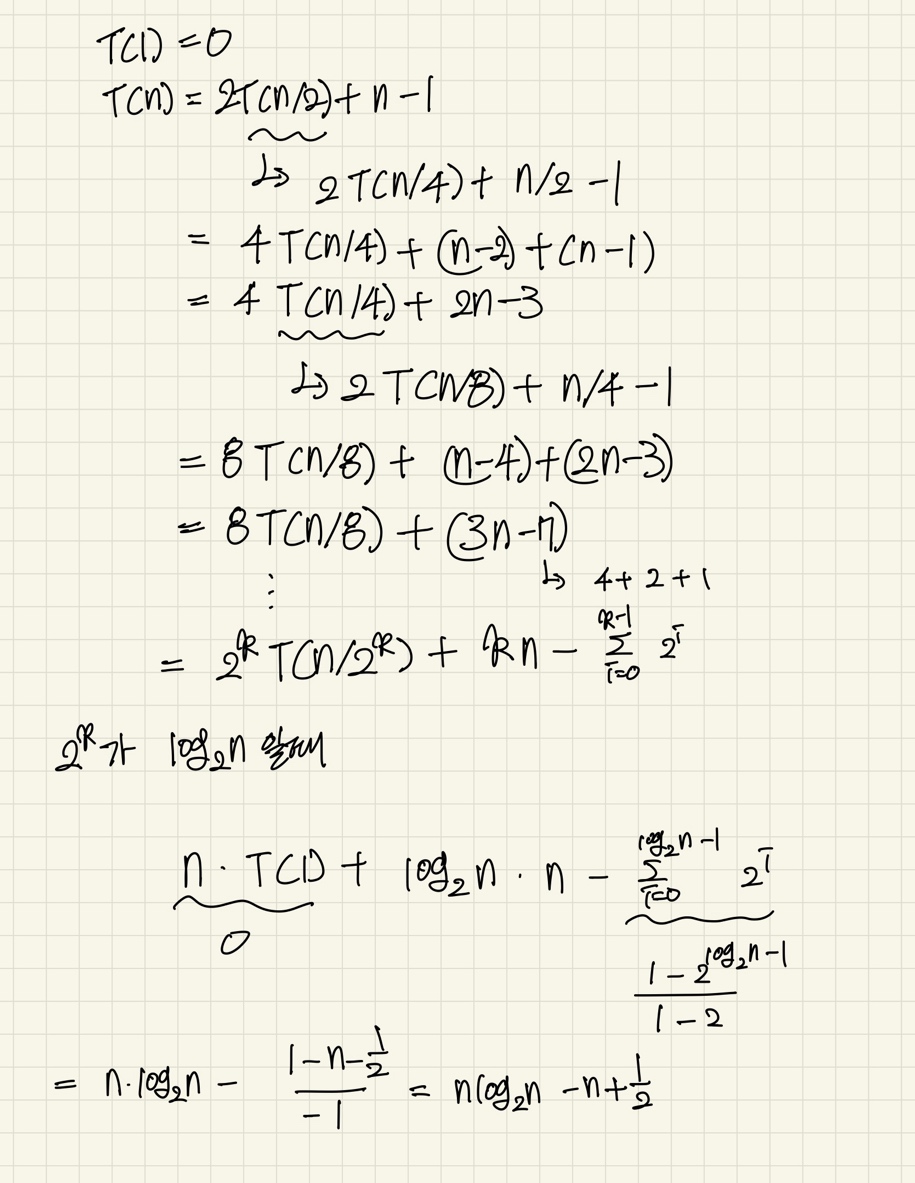
1. 2를 자식 노드 중 큰 값인 14와 교환한다.
2. 2를 자식 노드 중 큰 값인 8과 교환한다.
3. 2를 자식 노드 중 큰 값인 4와 교환한다.
4. 더이상 교환할 값이 없어 조정을 종료한다.

위 과정을 반복하며 정렬을 시행한다.

* 10개의 원소이므로 9번의 정렬이 시행된다.
* 9번의 정렬 후 원소 2개가 남았을 때 두 원소만 교환한다.

**문제 3 : 합병 정렬 최악의 실행시간 계산**

텍스트, 친필, 폰트, 잉크이(가) 표시된 사진

자동 생성된 설명텍스트, 친필, 폰트, 편지이(가) 표시된 사진

자동 생성된 설명

**문제 4 : 합병 정렬**

**의사코드**

|  |  |  |
| --- | --- | --- |
| 합병 알고리즘 | 순환 & 비 순환 정렬  (배열을 분활 후 합병 하는 과정) | 순환적 합병 정렬  그림으로 표현 |
|  | 텍스트, 친필, 폰트, 문서이(가) 표시된 사진  자동 생성된 설명 | 텍스트, 폰트, 도표, 스크린샷이(가) 표시된 사진  자동 생성된 설명 |

**프로그램 설계**

순환, 비 순환적 합병은 배열을 분할하는 방식으로 나뉜다.

즉 합병하는 방식은 동일하다.

합병 정렬을 수행하는 Merge class를 정의한다.

배열을 합병하는 메소드를 하나 정의하고,

순환적, 비 순환적 메소드에서 합병 메소드를 호출해 사용한다.

순환적 합병 정렬 메소드는 재귀적으로 배열을 분할하고

비 순환적 합병 정렬 메소드는 반복문으로 합병해야 하는 배열의 사이즈를 조정하며 합병을 실행한다.

**멤버변수**

**arr** : 길이가 10인 배열, 10개의 값을 입력 받아 저장한다.

**멤버 메소드**

**생성자** : 사용자로부터 10개의 값을 입력 받아 배열에

저장한다.

**setArr()** : 사용자로부터 10개의 값을 입력 받아 배열에

저장한다.

**print()** : 현재 배열을 출력한다.

**merge (int left, int mid, int right)** : 부분 배열 합병 메소드

부분배열1 : index left ~ mid

부분배열2 : index mid+1 ~ right

right – left + 1 크기의 임시배열을 생성한다.

부분배열 1과 부분배열 2를 비교하며 작은 수부터

임시배열에 저장한다.

한쪽의 배열의 값이 모두 들어갔다면 나머지 배열의 값을

순서대로 넣는다.

임시배열에 원소의 값들이 정렬 되어있다.

부분배열의 left ~ right까지의 index에 정렬된 값을

저장한다.

**re\_MergeSort(int left, int right)** :

index left ~ right 까지의 배열을 반으로 나눈 뒤

합병한다.

left와 right의 중간 index인 mid를 계산해

mid를 기준으로 부분배열을 반으로 나눈다.

배열의 범위가 1이 될 때 까지 즉 1개의 원소만

남을 때 까지 배열을 분할 후

merge 메소드로 합병해 간다.

**it\_MergeSort()** : 비 순환적 합병 정렬

순환적 합병은 배열의 길이를 절반 씩 줄여가지만

비 순환적 합병은 배열의 길이를 1부터 2배씩

늘려간다.

두 부분 배열은 인접해 있기 때문에,

부분배열 1 : 시작 index ~ 시작index + size – 1

부분배열 2 :

시작 index + size ~ 시작index + 2\*size – 1

임을 알 수 있다.

size를 1부터 2배씩 늘려가며 부분배열들을

합병한다.

**소스코드**

// 2021111971 이재혁 문제 4

#include <iostream>

using namespace std;

class Merge {

public:

// 10개의 값을 가지는 배열 생성

Merge() {

cout << "배열원소 10개 입력 >> ";

for (int i = 0; i < 10; i++) {

cin >> arr[i];

}

}

void merge(int left, int mid, int right) {

// index: left ~ mid와 mid+1 ~ right의 병합

int \*list;

list = new int[right - left + 1];

int start1 = left; // 왼쪽 부분배열의 시작 index

int start2 = mid + 1; // 오른쪽 부분배열의 시작 index

int index = 0; // 정렬된 원소가 들어갈 배열의 index

while (start1 <= mid && start2 <= right) {

if (arr[start1] < arr[start2]) {

list[index++] = arr[start1++];

} else {

list[index++] = arr[start2++];

}

} // 두개의 부분배열들을 비교하면서 작은 것들을 먼저 임시 배열에 저장

if (start1 > mid) { // 왼쪽 부분배열의 값들이 먼저 다 들어갔다면

// 오른쪽 부분배열의 값들을 그대로 이어 붙이기

for (int i = start2; i <= right; i++) {

list[index++] = arr[i];

}

} else { // 오른쪽 부분배열의 값들이 먼저 다 들어갔다면

// 왼쪽 부분배열의 값들을 그대로 이어 붙이기

for (int i = start1; i <= mid; i++) {

list[index++] = arr[i];

}

}

for (int i = 0; i < index; i++) {

arr[left + i] = list[i];

} // 임시로 저장한 배열의 순서를 원래 배열에 저장

this->print(); // 병합 후 현재 배열상태 출력

delete[] list;

}

void re\_MergeSort(int left, int right) {

// 순환적 병합정렬

// 정렬할 배열의 시작과 끝 index를 매개변수로 전달받는다.

if (left < right) { // 배열의 범위가 1이상 일때 정렬한다.

int mid = (left + right) / 2; // 배열을 2부분으로 나눌 중간 index

this->re\_MergeSort(left, mid); // 왼쪽 부분배열을 다시 병합정렬

this->re\_MergeSort(mid + 1, right); // 오른쪽 부분배열 다시 병합정렬

this->merge(left, mid, right); // 나누었던 부분배열을 병합

}

}

void it\_MergeSort() {

// 비순환적 병합 정렬

// 병합할 부분배열의 크기를 1부터 2배씩 늘려가며 (분할을 n/2로 하기때문)

// 병합시행

for (int size = 1; size <= 9; size += size) {

// 부분배열의 크기를 1부터 전체배열의 크기까지 2배씩 늘려간다

for (int start = 0; start <= 9 - size; start += 2 \* size) {

// start : 2개의 부분배열 중 가장 왼쪽에 있는 index

// 인접한 부분배열이기 때문에 시작과 끝 index를 알면 병합가능

int mid = start + size - 1;

// start 부터 시작해 2\*size 만큼 이동하면

// 다음 정렬해야 하는 부분배열들의 시작

// 따라서 -1을 하면 부분배열들의 마지막 index

int end = start + 2 \* size - 1 < 9 ? start + 2 \* size - 1 : 9;

this->merge(start, mid, end); // 나눈 배열 병합

}

}

}

void print() {

for (int i = 0; i < 10; i++) {

cout << arr[i] << " ";

}

cout << endl;

}

void setArr() {

cout << "배열원소 10개 입력 >> ";

for (int i = 0; i < 10; i++) {

cin >> arr[i];

}

}

private:

int arr[10];

};

int main() {

Merge m;

cout << "정렬 이전 원소" << endl;

m.print();

cout << "-------- 순환적 병합 정렬 --------" << endl;

m.re\_MergeSort(0, 9);

m.print();

m.setArr();

cout << "------ 비 순환적 병합 정렬 -------" << endl;

m.it\_MergeSort();

m.print();

}

// 30 20 40 35 5 50 45 10 25 15

**결과분석**

**순환적 합병 정렬**

**텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

30 20 40 35 5 50 45 10 25 15

왼쪽부터 합병 시작

부분 배열의 길이가 1이 됐을 때

30, 20 -> 20 30

(20 30), 40 -> 20 30 40

35, 5 -> 5 35

(20 30 40), (5 35) -> 5 20 30 35 40

45, 10 -> 10 45

25, 15 -> 15 25

50, (10 45) -> 10 45 50

(10 45 50), (15 25) -> 10 15 25 45 50

(5 20 30 35 40), (10 15 25 45 50)

-> 5 10 15 20 25 30 35 40 45 50

정렬 완료

**비 순환적 합병 정렬**

**텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

size -> 부분 배열의 크기, 1부터 2배씩 증가시킨다.

(부분배열 2개씩 합치기 때문)

size 1

30, 20 -> 20 30

40, 35 -> 35 40

5, 50 -> 5 50

45, 10 -> 10 45

25, 15 -> 15, 25

size 2

(20 30), (35, 40) -> 20 30 35 40

(5 50), (10 45) -> 5 10 45 50

(15 25) -> 15 25

size 4

(20 30 35 40), (5 10 45 50)

* 5 10 20 30 35 40 45 50

(5 10 20 30 35 40 45 50), (15 25)

* 5 10 15 20 25 30 35 40 45 50

정렬 완료