# 알고리즘 설계

8조
김강우, 손준호, 최재형, 이준혁

# INDEX

# INTRODUCTION

소개

# Introduction (소개)

| 이름(NAME) | 정렬 (Sort) |
|---|---|
| 김강우 | Quick sort |
| 손준호 | Merge sort |
| 이준혁 | Bubble sort |
| 최재형 | Radix sort |

- 정렬을 구현하여 각자 실제 소요시간을 측정하는 것을 중점으로 만들었습니다.

# QUICK SORT

퀵 정렬

# Quick sort (퀵 정렬)

Csv 파일을 문자열 배열 에 저장

```
// csv 파일 배열 생성
void readcsv(char data[MAX_LINE][MAX_TOK]) {
    FILE *file = fopen(CSVFILE, "r");

    if (file == NULL) {
        perror("파일 열기 실패");
        exit(EXIT_FAILURE);
    } else {
        printf("파일 열기 성공\n");
    }

    int cnt = 0;

    // 파일에서 문자열 데이터를 읽어옴
    while (fscanf(file, "%s", data[cnt]) == 1 && cnt
< MAX_LINE) {
        cnt++;
    }


    // 파일 닫기
    fclose(file);
}
```

# Quick sort (퀵 정렬)

배열에서 정상코드,
비정상코드 분류

```
void generation(char data[MAX_LINE][MAX_TOK],
char def[MAX_LINE][MAX_TOK], char
result[MAX_LINE][MAX_TOK] ){

    int cnt;
    int dcnt=0;
    int rcnt =0;

    for(cnt=0; cnt<MAX_LINE; cnt++){
        if(strlen(data[cnt])<MAX_TOK-1 &&
strlen(data[cnt])>0){
            strcpy(def[dcnt], data[cnt]);
            dcnt++;
        }
        if(strlen(data[cnt])==MAX_TOK-1){
            strcpy(result[rcnt], data[cnt]);
            rcnt++;
        }
    }

    d_cnt = dcnt;
    file_cnt = cnt;
    r_cnt = rcnt;
}
```

# Quick sort (퀵 정렬)

- left : 배열의 가장 왼쪽 값
- right : 배열의 가장 오른쪽 값
- Pivot : 정렬할 기준값

1. 배열에서 pivot을 기준으로
   왼쪽 : low < pivot
   오른쪽 : pivot < right
2. 만약 low < high 이면 값 swap(list[low], list[high], t)
3. 마지막으로 pivot 위치와 high 위치를 변경
4. return high;

```
int partition(long long list[MAX_LINE], int left, int
right){
    long long pivot, temp; int low, high;

    low = left;
    high = right +1;
    pivot = list[left];
    do{
        do{
            low++;
        }while(low<=right && list[low]<pivot);

        do{
            high--;
        }while(high>=left && list[high]>pivot);

        if(low<high){
            SWAP(list[low], list[high], temp);
        }
    }while(low<high);

    SWAP(list[left], list[high], temp);
    return high;
}
```

# Quick sort (퀵 정렬)

```
void Quick_sort(long long list[MAX_LINE], int left,
int right){
    if(left<right){
        int q = partition(list, left, right); // 정복

        Quick_sort(list, left, q-1); // 분할
        Quick_sort(list, q+1, right);  // 분할
    }
}
```

1. 배열에서 pivot을 기준으로 배열을 좌우로 재귀정렬

# Quick sort (퀵 정렬)

```
if(INPUT1 == 1){// 전체 코드 정렬 (data)
        int cnt =0;
        long long code[MAX_LINE];
        while(cnt != MAX_LINE){
            code[cnt] = atoll(data[cnt]);
            cnt++;
        }

        std = clock();
        Quick_sort(code, 0, MAX_LINE-1);
        end = clock();

        for (size_t i = 0; i < MAX_LINE; i++) {
            printf("%lld\n", code[i]);
        }
    }
```

Long long 타입 전체 코드 정렬

# Quick sort (퀵 정렬)

```
if(INPUT1==2){
    printf("정렬하고싶은 위치를 말해주세요.\n");
    printf("[ 1 ][ 2 ][ 3 ]\n\n");
    printf("[ %d ] 선택하셨습니다.\n\n", INPUT2);

    for(int i=0;i<r_숏;i++){
        sscanf(result[i], "%3s%3s%8s", con[i].one, con[i].two, con[i].three);
    }
    std = clock();
    quic(0, r_cnt-1);
    end = clock();

    for(int i=0;i<r_cnt;i++){
            printf("%s%s%s\n", con[i].one, con[i].two, con[i].three);
    }
}
```

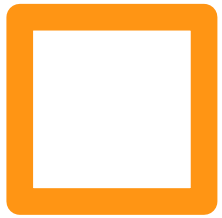
^ Con 이라는 구조체 배열에 정상코드 값 으로 초기화

<<< 구조체를 정렬하는 퀵정렬 : quic

# Quick sort (퀵 정렬)

```
if(INPUT1==3){
        printf("정렬하고싶은 위치를 말해주세요.\n");
        printf("[ 2 ][ 3 ]\n\n");
        printf("[ %d ] 선택하셨습니다.\n\n", INPUT2);

        if(INPUT2 == 1){
            printf("잘못된 입력");
            return 1;
        }

        for(int i=0;i<d_cnt;i++){
            sscanf(def[i], "%3s%8s", con[i].two, con[i].three);
        }
        std = clock();
        quic(0, d_cnt-1);
        end = clock();

        for(int i=0;i<d_cnt;i++){
                printf("%s%s\n", con[i].two, con[i].three);
        }
    }
```

<<< 1이라는 값을 넣으면 비정상 종료



# Quick sort (퀵 정렬)

```
파일 열기 성공          정상코드 2번째 정렬
파일 길이 : 30000
정상 파일 길이 : 26913
오류 코드 갯수 : 3087
---------------------------------|
<-----도서관 분류 코드입니다.----->
---------------------------------
정렬하고싶은 데이터를 말해주세요
1. 전체 2. 정상 코드 3. 비정상 코드]

정렬 [ 2 ] 선택했습니다.

정렬하고싶은 위치를 말해주세요.
[ 1 ][ 2 ][ 3 ]

[ 2 ] 선택하셨습니다.
```

전체 정렬

```
90099020200922
90099020201024
90099020201107
90099020201121
90099020210629
90099020210709
90099020210827
90099020211006
90099020211110
90099020220605
90099020220818
90099020221019
90099020221027
정렬하는데 걸린 시간 : 0.0080000000 초 입니다.
```

* intel i5 – 13400p 기준

| 코드 | 시간(초) |
|---|---|
| **전체 코드 정렬** | 0.01 |
| | |
| **정상 코드 1정렬** | 0.005 |
| **정상 코드 2정렬** | 0.010 |
| **정상 코드 3정렬** | 0.019 |
| | |
| **비정상 코드 1정렬** | 0.0000000000 (측정 불가) |
| **비정상 코드 2정렬** | 0.004 |

# MERGE SORT

합병 정렬

# Merge sort (합병 정렬)

```
readcsv(list, normal, abnormal); // csv파일 읽기 및 정제
```

```
// CSV 파일 읽기 및 정제
void readcsv(char list[MAX_SIZE][MAX_TOK], char normal[MAX_SIZE][MAX_TOK], char abnormal
[MAX_SIZE][MAX_TOK]) {
    FILE *fp = fopen("library.csv", "r");
    int t = 0;
    while (fscanf(fp, "%s", list[t]) == 1 && t < MAX_SIZE) { // 문자열 데이터 읽어오기
        t++;
    }
    fclose(fp);

    for (int i = 0; i < MAX_SIZE; i++) {
        if (strlen(list[i]) == MAX_TOK - 1) {
            strcpy(normal[a], list[i]); // 정상 도서코드를 normal 배열에 복사
            a++;
        } else{
            strcpy(abnormal[b], list[i]); // 비정상 도서 코드를 abnormal 배열에 복사
            b++;
        }
    }
```

- **list = 정상, 비정상 도서코드**
- **normal = 정상 도서코드**
- **abnormal = 비정상 도서코드**

**csv파일 문자열을 list 배열에 읽어오고, 정상 비정상 도서 코드를 담을 배열에 복사**

# Merge sort (합병 정렬)

```
struct BookInfo {
    int category;
    int symbol;
    int date;
};
```

구조체 선언

```
struct BookInfo books[MAX_SIZE];
for (int i = 0; i < a; i++) {
    struct BookInfo bookInfo = convertToBookInfo(normal[i]); // 문자열을 구조체로 변환
    books[i] = bookInfo; // 구조체 배열에 저장
}
```

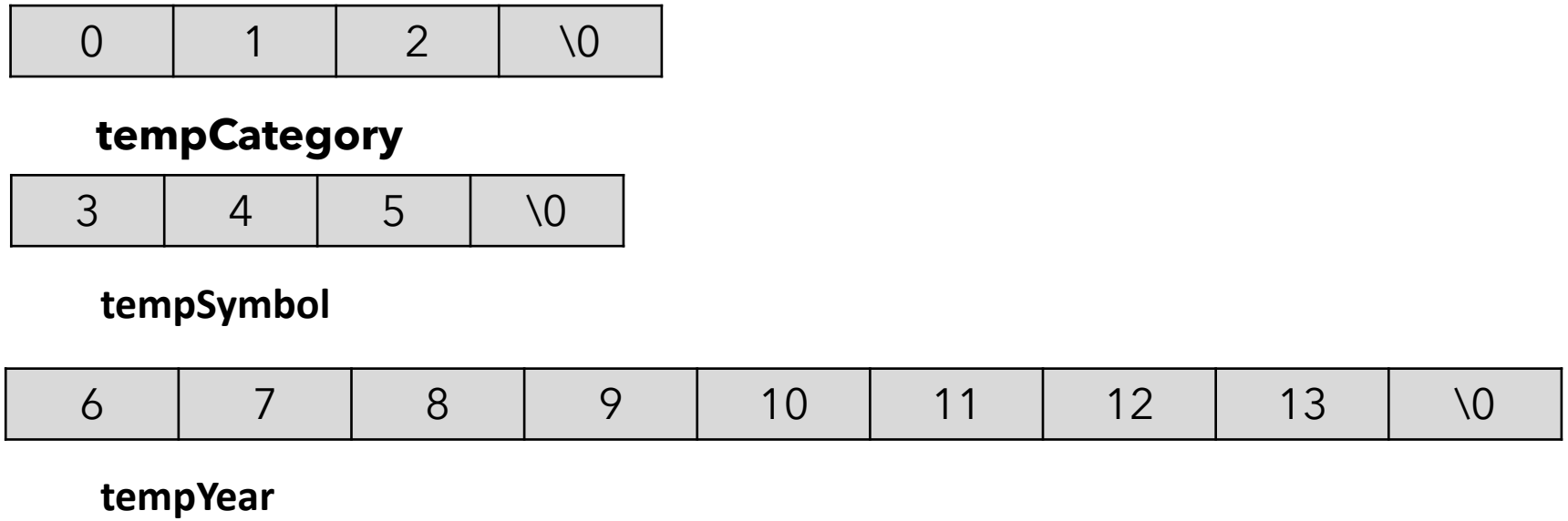
**정상 코드를 담은 배열을 구조체로 변환후 구조체 배열에 저장**

**(BookInfo 형식의 변수 bookInfo에 반환된 구조체를 임시 저장후 books배열의 i번째 인덱스에 복사)**

# Merge sort (합병 정렬)

```
struct BookInfo convertToBookInfo(char normal[]) {
    struct BookInfo bk;

    char tempCategory[4];
    char tempSymbol[4];
    char tempYear[9];

    strncpy(tempCategory, normal, 3); // 문자열 일부분 복사
    tempCategory[3] = '\0';
    strncpy(tempSymbol, normal + 3, 3);
    tempSymbol[3] = '\0';
    strncpy(tempYear, normal + 6, 8);
    tempYear[8] = '\0';

    bk.category = atoi(tempCategory); // 문자열 정수 변환후 구조체에 저장
    bk.symbol = atoi(tempSymbol);
    bk.year = atoi(tempYear);

    return bk;
}
```

# Merge sort (합병 정렬)



# Merge sort (합병 정렬)

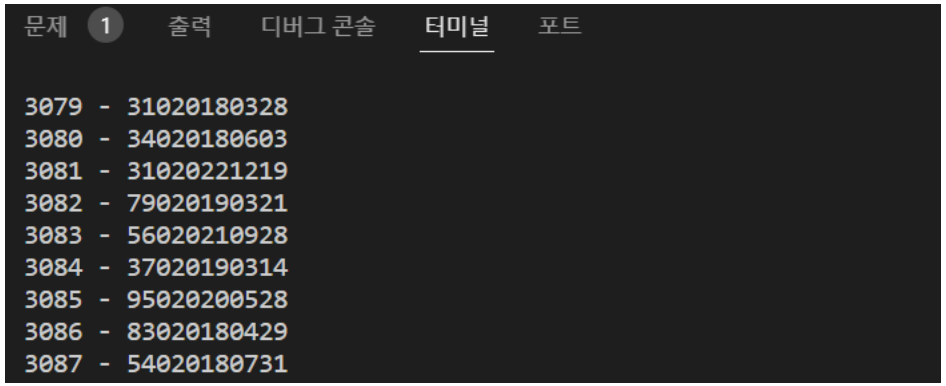
```
    start=clock();
    merge_sort(books, 0, a - 1); // 합병 정렬 수행
```

**시간 측정 시작후 합병정렬 시작 (a는 정상 도서 크기)**

```
printf("\t\t\t\t\t원하시는 작업을 선택해주세요.\n\n");
printf("\t[1] 도서 분류 기준 정렬 [2] 분류 기호 기준 정렬 [3] 연도 기준 정렬 [4] 전체 정렬
[5] 비정상 도서 출력\n입력 : ");
scanf("%d", &sc);
if(sc==5){
    trash(abnormal);
    exit(0);
}
```

**정렬 시작전 정렬 기준을 선택하기 위해 sc값 입력**
**정렬 시작 전 미리 분류 하려는 값을 저장**
**비정상 도서출력일 경우 그냥 비정상 값만 출력 하고 즉시종료(정렬 X)**

# Merge sort (합병 정렬)




```c
void trash(char abnormal[MAX_SIZE][MAX_TOK]){
    for(int i=0; i<b;i++){
        printf("%d - %s\n",i+1, abnormal[i]);
    }
}
```

- **b는 비정상 도서 크기**

# Merge sort (합병 정렬)

```c
void merge_sort(struct BookInfo book[], int left, int right) {
    if (left < right) {

        int mid = (left + right) / 2; //리스트 균등분할
        merge_sort(book, left, mid); // 왼쪽 부분 배열을 재귀적으로 정렬
        merge_sort(book, mid + 1, right); // 오른쪽 부분 배열을 재귀적으로 정렬

        switch (sc) {
            case 1: // 도서 분류 코드로 정렬
                merge_category(book, left, mid, right);
                break;
            case 2: // 분류 기호 코드로 정렬
                merge_symbol(book, left, mid, right);
                break;
            case 3: // 연도로 정렬
                merge_year(book, left, mid, right);
                break;
            case 4: // 분류 기준 전체로 정렬
                merge_all(book, left, mid, right);
                break;
            default:
                printf("잘못된 접근입니다. 프로그램을 정지합니다.");
                exit(0);
        }
    }
}
```

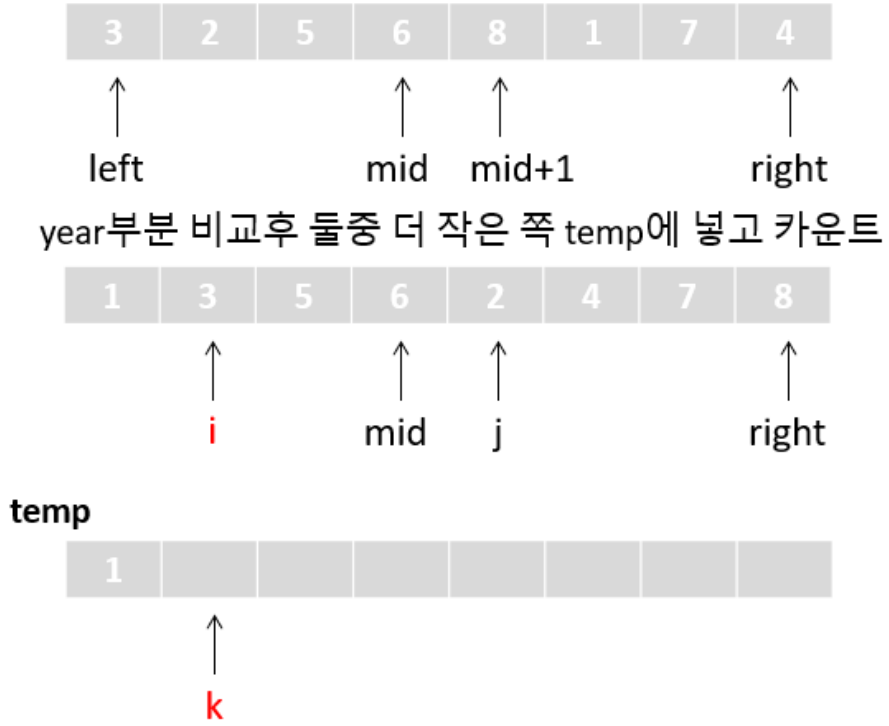
- 입력 받은 인덱스 값으로 합병정렬 시작
- 1,2,3,4,5 이외의 값 입력시 즉시 종료
- 분할 과정후 sc값을기준 으로 알맞은 함수로 호출되어 정복과정을 거침

합병 정렬은 분할 과정에서 같은 크기의 2개의 부분배열을 재귀적 호출을 통해 충분히 작은 수까지 배열을 나눈 후  정복한다 이때 부분배열의 크기가 작지 않으면 다시 분할 정복기법을 적용하며 결합과정을 통해 정렬된 부분배열을 하나의 배열에 통합 시킴

# Merge sort (합병 정렬)

연도 기준 정렬일 경우

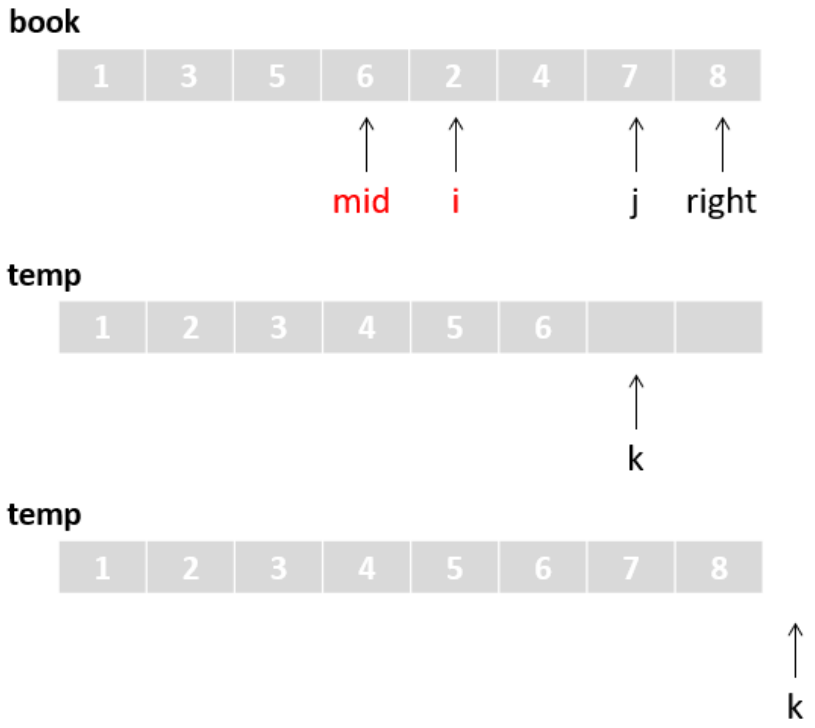
i, j, k에 left, mid+1, left값넣기
예시)




```c
void merge_year(struct BookInfo book[], int left, int mid, int right) {
    int i, j, k, l;
    struct BookInfo temp[MAX_SIZE];
    i = left;
    j = mid+1;
    k = left;
    while (i <= mid && j <= right) { // 분할 정렬된 배열 합병
        if (book[i].year <= book[j].year) {
            temp[k++] = book[i++];
        } else {
            temp[k++] = book[j++];
        }
    }
    if(i>mid){
        for(l=j;l<=right;l++){ // 남은 요소들을 일괄 복사
            temp[k++] = book[l];
        }
    }
    else{
        for(l=i;l<=mid;l++) {
            temp[k++] = book[l];
        }
    }
    for(l=left;l<=right;l++){ // tmpe를 book으로 재복사
        book[l]=temp[l];
    }

    end=clock();
}
```

book값을 temp에 복사해야하기때문에
BookInfo 형식 배열 선언

# Merge sort (합병 정렬)




**도서 분류, 분류 기호 정렬도 정렬 기준(year)만 다르고 나머지는 연도와 똑같다.**

```c
void merge_year(struct BookInfo book[], int left, int mid, int right) {
    int i, j, k, l;
    struct BookInfo temp[MAX_SIZE];
    i = left;
    j = mid+1;
    k = left;
    while (i <= mid && j <= right) { // 분할 정렬된 배열 합병
        if (book[i].year <= book[j].year) {
            temp[k++] = book[i++];
        } else {
            temp[k++] = book[j++];
        }
    }
    if(i>mid){
        for(l=j;l<=right;l++){ // 남은 요소들을 일괄 복사
            temp[k++] = book[l];
        }
    }
    else{
        for(l=i;l<=mid;l++) {
            temp[k++] = book[l];
        }
    }
    for(l=left;l<=right;l++){ // tmpe를 book으로 재복사
        book[l]=temp[l];
    }

    end=clock();
}
```

한쪽이 끝났을 경우 남아있는 쪽을 일괄 복사해 temp에 담은 후 book에 재복사

# Merge sort (합병 정렬)

```
void merge_all(struct BookInfo book[], int left, int mid, int right) {
    int i, j, k, l;
    struct BookInfo temp[MAX_SIZE];
    i = left;
    j = mid+1;
    k = left;


    while (i <= mid && j <= right) {
        if (book[i].category < book[j].category) {
            temp[k++] = book[i++];
        } else if (book[i].category == book[j].category) {
            if (book[i].symbol < book[j].symbol) {
                temp[k++] = book[i++];
            } else if (book[i].symbol == book[j].symbol) {
                if (book[i].year <= book[j].year) {
                    temp[k++] = book[i++];
                } else {
                    temp[k++] = book[j++];
                }
            } else {
                temp[k++] = book[j++];
            }
        } else {
            temp[k++] = book[j++];
        }
```

도서분류(category)를 먼저 비교하고
같을경우 분류기호(symbol)를 비교,
같을경우 연도(year)을 비교한다.

이렇게 각자 비교해서 temp에 복사한다.

그러면

9009902018012 8
9009902018032 4

이런식으로 정렬이 되기 시작한다.

```
26046 - 도서 분류 : 900, 분류 기호 : 740, 연도: 20220928
26047 - 도서 분류 : 900, 분류 기호 : 740, 연도: 20221027
26048 - 도서 분류 : 900, 분류 기호 : 740, 연도: 20221104
26049 - 도서 분류 : 900, 분류 기호 : 740, 연도: 20221226
26050 - 도서 분류 : 900, 분류 기호 : 740, 연도: 20221231
26051 - 도서 분류 : 900, 분류 기호 : 750, 연도: 20180220
26052 - 도서 분류 : 900, 분류 기호 : 750, 연도: 20180302
```

# Merge sort (합병 정렬)

```
        }
    }
    // 남은 요소들을 복사
    if(i>mid){
        for(l=j;l<=right;l++){
            temp[k++] = book[l];
        }
    }
    else{
        for(l=i;l<=mid;l++) {
            temp[k++] = book[l];
        }
    }
    for(l=left;l<=right;l++){
        book[l]=temp[l];
    }

    end=clock();
}
```

**전체 정렬일 경우**

나머지는 다른 정렬과 똑같이 남은 걸 복사하고 book으로 재 복사한다.

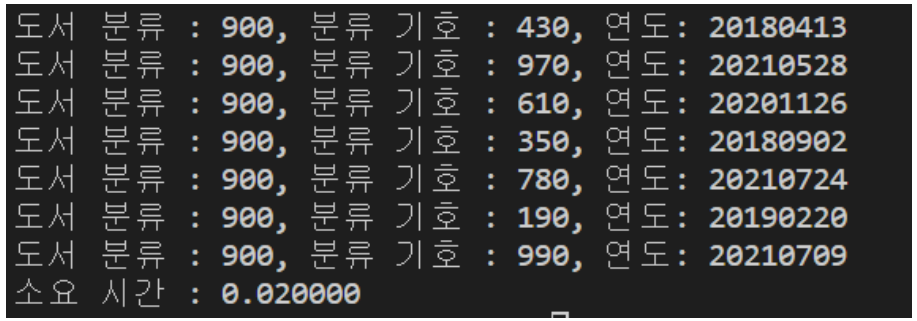
# Merge sort (합병 정렬)

```
printBooks(books, a); // 정렬된 결과 출력
printf("소요 시간 : %f\n", (float)(end - start)/CLOCKS_PER_SEC);
```

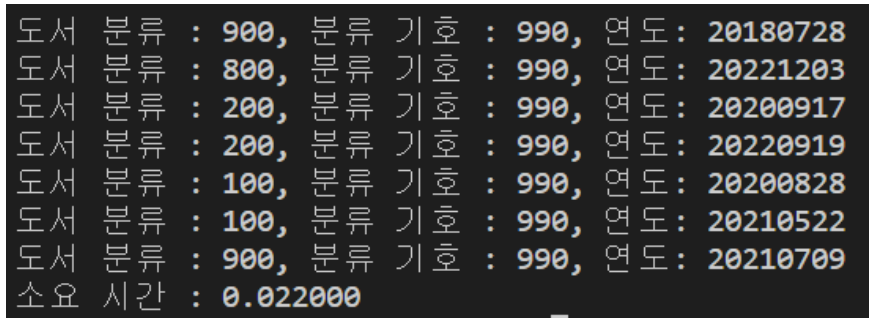
```
void printBooks(struct BookInfo book[], int n) {
    for (int i = 0; i < n; i++) {
        printf("도서 분류 : %d, 분류 기호 : %d, 연도: %d\n", book[i].category, book[i].symbol,
        book[i].year);
    }
}
```

**정렬 완료한 도서분류, 분류기호, 연도를 출력**

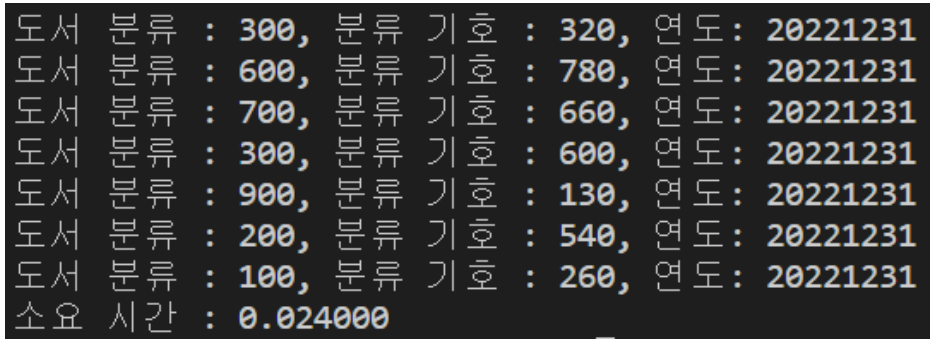
# Merge sort (합병 정렬)



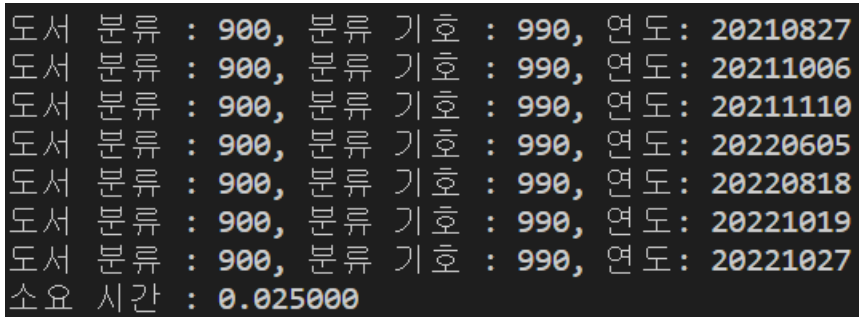

도서 분류 정렬




분류 기호 정렬




연도 정렬




전체 정렬

* INTEL i5 - 1135G7 기준

| 정렬 방법 | 시간 |
|---|---|
| 도서 분류 기준 정렬 | 0.02 |
| 분류 기호 기준 정렬 | 0.022 |
| 연도 기준 정렬 | 0.024 |
| 전체 정렬 | 0.025 |

# BUBBLE SORT

버블 정렬

# Bubble sort (버블 정렬)

```c
void readcsv(char data[MAX_LINE][MAX_TOK]) {
    FILE *file = fopen(CSVFILE, "r");

    if (file == NULL) {
        perror("파일 열기 실패");
        exit(EXIT_FAILURE);
    } else {
        printf("파일 열기 성공\n");
    }

    size_t cnt = 0;

    while (fscanf(file, "%s", data[cnt]) == 1 && cnt < MAX_LINE) {
        cnt++;
    }

    fclose(file);
}
```

- **Csv 파일의 내용을 'data'에 저장**

# Bubble sort (버블 정렬)

```
void separate(char data[MAX_LINE][MAX_TOK], char a[MAX_LINE][MAX_TOK], char b[MAX_LINE][MAX_TOK]) {
    int aIndex = 0;
    int bIndex = 0;

    for (int i = 0; i < MAX_LINE; i++) {
        if (strlen(data[i]) == 14) {
            strcpy(a[aIndex], data[i]);
            aIndex++;
        } else if (strlen(data[i]) == 11) {
            strcpy(b[bIndex], data[i]);
            bIndex++;
        }
    }

    index_a = aIndex;
    index_b = bIndex;
}
```

**'data'배열에 저장된 문자열의 길이에 따라 14자리인경우 a, 11자리인경우 b에 복사**

**Index_a와 index_b에 각각 a,b를 저장하여 나누어진 데이터를 관리**

# Bubble sort (버블 정렬)

```
void bubbleSort(struct BookInfo arr[], int n, int field) {
    start = clock();

    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            switch (field) {
                case 1:
                    if (arr[j].category > arr[j + 1].category) {
                        struct BookInfo temp = arr[j];
                        arr[j] = arr[j + 1];
                        arr[j + 1] = temp;
                    }
                    break;
                case 2:
                    if (arr[j].symbol > arr[j + 1].symbol) {
                        struct BookInfo temp = arr[j];
                        arr[j] = arr[j + 1];
                        arr[j + 1] = temp;
                    }
                    break;
                case 3:
                    if (arr[j].date > arr[j + 1].date) {
                        struct BookInfo temp = arr[j];
                        arr[j] = arr[j + 1];
                        arr[j + 1] = temp;
                    }
                    break;
            }
        }
    }
```

- N : 배열의 크기
- field : 정렬 기준 (도서 분류 코드, 분류 기호 코드, 날짜 중 선택)
- arr : 정렬할 배열

**field가 1인 경우: arr[j].category 값을 기준으로 정렬.**

**2인 경우: arr[j].symbol 값을 기준으로 정렬**
**3인 경우: arr[j].date 값을 기준으로 정렬**

**이중첩된 반복문을 통하여 배열을 반복한후 인접한 값을 비교하고 크기에 따라 값을 교환**

# Bubble sort (버블 정렬)

```c
void trash(char abnormal[MAX_LINE][MAX_TOK]) {
    for (int i = 0; i < index_b; i++) {
        printf("%d - %s\n", i + 1, abnormal[i]);
    }
}

void symbolSort(struct BookInfo arr[], int n) {
    bubbleSort(arr, n, 2);
    printBooks(arr, n, 2);
}

void dateSort(struct BookInfo arr[], int n) {
    bubbleSort(arr, n, 3);
    printBooks(arr, n, 3);
}
```

```c
void overallSort(struct BookInfo arr[], int n) {
    bubbleSort(arr, n, 1);

    // 도서 코드가 같으면 분류 기호 코드로 정렬
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j].category == arr[j + 1].category && arr[j].symbol > arr[j + 1].symbol) {
                struct BookInfo temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    // 분류 기호 코드가 같으면 날짜로 정렬
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j].category == arr[j + 1].category && arr[j].symbol == arr[j + 1].symbol &&
                arr[j].date > arr[j + 1].date) {
                struct BookInfo temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
```

기호코드, 날짜, 비정상 도서 코드들을 정렬, 출력하는 함수

전체 정렬은 void overallSort(struct BookInfo arr[], int n) 함수를 사용하여 도서코드를 기준으로 정렬 후 도서코드가 같은 경우 기호코드 기준으로 정렬 함

이중 for 루프를 사용하여 `arr[j].category == arr[j + 1].category` 조건으로 도서 코드가 같은지 확인하고, `arr[j].symbol > arr[j + 1].symbol` 조건으로 분류 기호 코드를 비교하여 정렬

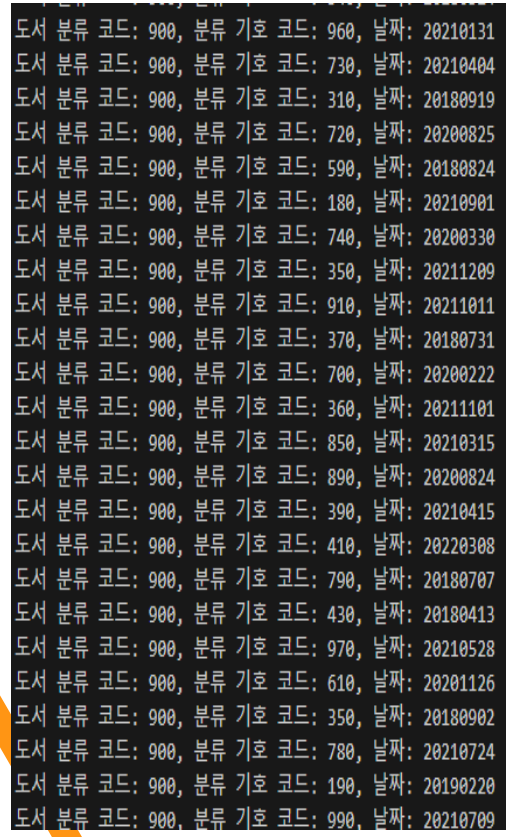
같은방법으로 기호코드가 같으면 날짜로 정렬

# Bubble sort (버블 정렬)

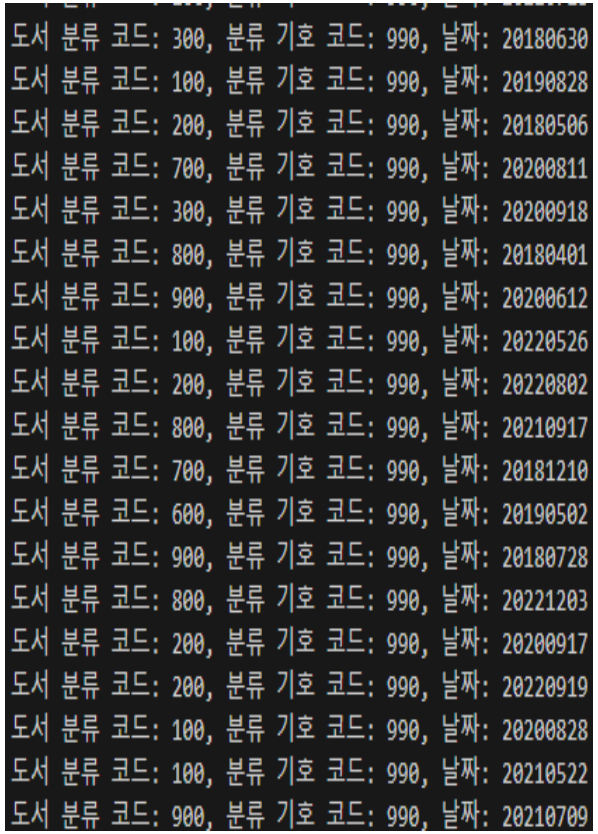
```
switch (sc) {
    case 1:
        // 도서 분류 코드를 기준으로 정렬
        for (int i = 0; i < index_a; i++) {
            books[i] = convert(a[i]);
        }
        bubbleSort(books, index_a, 1);
        printBooks(books, index_a, 1);
        break;
    case 2:
        // 분류 기호 코드를 기준으로 정렬
        for (int i = 0; i < index_a; i++) {
            books[i] = convert(a[i]);
        }
        symbolSort(books, index_a);
        break;
    case 3:
        // 날짜를 기준으로 정렬
        for (int i = 0; i < index_a; i++) {
            books[i] = convert(a[i]);
        }
        dateSort(books, index_a);
        break;
    case 4:
        // 전체 정렬
        for (int i = 0; i < index_a; i++) {
            books[i] = convert(a[i]);
        }
        overallSort(books, index_a);
        break;
    case 5:
        // 비정상 도서 출력
        trash(b);
        break;
    default:
        printf("잘못된 입력입니다.\n");
        break;
}
printf("정렬 소요 시간 : %f 초\n", (float)(end - start) / CLOCKS_PER_SEC);
```

**지정된 값을 입력 할 시 선택한 작업이 수행되고, 범위를벗어난 값을 입력한 경우 예외 처리로 “잘못 된 입력입니다” 출력**

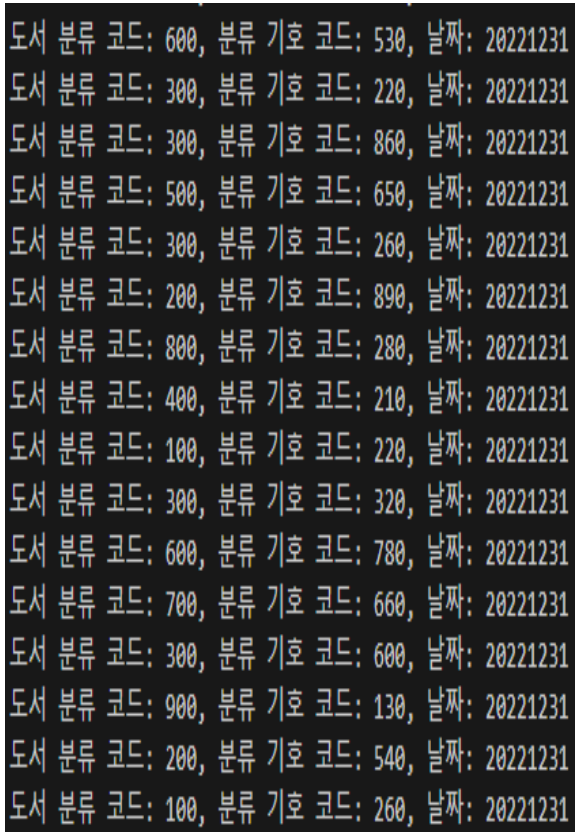
# Bubble sort (버블 정렬)



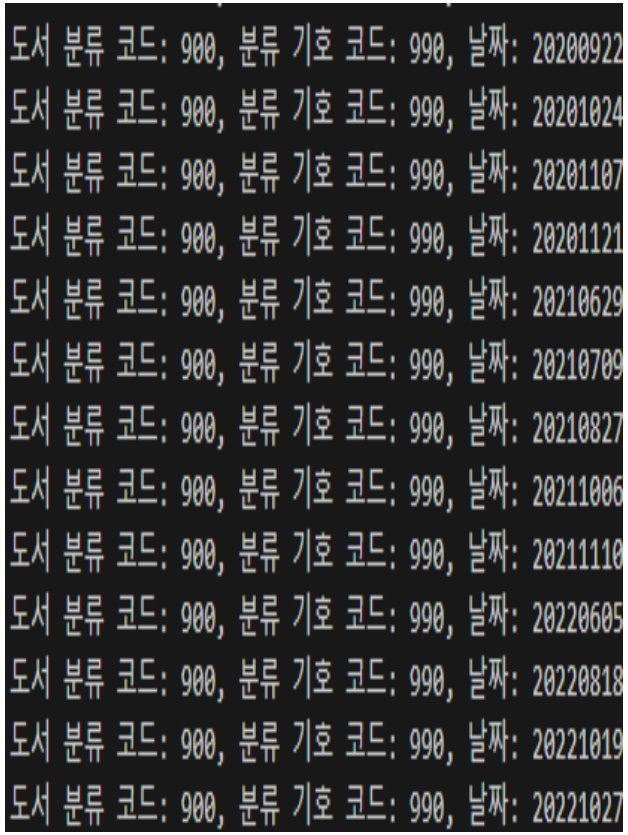

**도서 분류 정렬**




**분류 기호 정렬**




**날짜 정렬**




**전체 정렬**

# Bubble sort (버블 정렬)

* Intel core i5-8259U® CPU기준

| 정렬 방법 | 시간 |
|---|---|
| 도서 분류 기준 정렬 | 5.471sec |
| 분류 기호 기준 정렬 | 5.725sec |
| 날짜 기준 정렬 | 5.728sec |
| 전체 정렬 | 5.414sec |

# RADIX SORT

기수 정렬

```
#define CSVFILE "library.csv"
#define MAX_LINE 30000
#define MAX_TOK 15
#define INPUT 3
```

- MAX_LINE
= 전체 csv파일 데이터
- MAX_TOK
= 데이터 내 숫자열의 최대 길이 설정
- INPUT
= 출력할 기준 설정(1:도서, 2:분류, 3:날짜)

```
struct BookInfo {
    int category;
    int symbol;
    int date;
};
```

구조체 선언

```
int index_a, index_b;
```

- index_a = 정상 코드
- index_b = 오류 코드

```
void separate(char data[MAX_LINE][MAX_TOK], char a[MAX_LINE][MAX_TOK],
 char b[MAX_LINE][MAX_TOK]) {
    int aIndex = 0;
    int bIndex = 0;

    for (int i = 0; i < MAX_LINE; i++) {
        if (strlen(data[i]) == MAX_TOK - 1) {
            // 정상 코드인 경우
            strcpy(a[aIndex], data[i]);
            aIndex++;
        } else if (strlen(data[i]) == MAX_TOK - 4) {
            // 오류 코드인 경우
            strcpy(b[bIndex], data[i]);
            bIndex++;
        }
    }

    index_a = aIndex;
    index_b = bIndex;
}
```

aIndex에 정상 코드, bIndex에 오류 코드를 저장

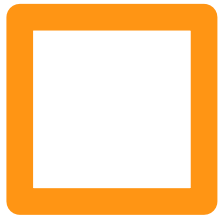
```c
struct BookInfo convertToBookInfo(char number[]) {
    struct BookInfo bookInfo;

    // 문자열을 정수로 변환하여 구조체에 저장
    bookInfo.category = atoi(strncpy((char[4]){}, number, 3));
    bookInfo.symbol = atoi(strncpy((char[4]){}, number + 3, 3));
    bookInfo.date = atoi(strncpy((char[9]){}, number + 6, 8));

    return bookInfo;
}
```

문자열 시작주소(0번째 인덱

문자열 및 널 종료 문자 임시저장 배열 선언
(널 종료 문자가 없을 시 문자열의 끝을 알 수 없음)

- char number[]로부터 0 ~ 2번째 인덱스 값 복사
- char[4]로 복사한 인덱스 값 저장
- atoi함수를 사용하여 복사된 문자열을 정수형으로 변환하여 bookInfo.category에 저장
- Symbol, date도 동일한 방식으로 형 변환

❖ 문자열 및 널 종료 문자 임시저장 배열 선언
(널 종료 문자가 없을 시 문자열의 끝을 알 수 없음)



| 0 | 1 | 2 | \0 |
|---|---|---|---|

**category**

| 3 | 4 | 5 | \0 |
|---|---|---|---|

**symbol**

| 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | \0 |
|---|---|---|---|---|---|---|---|---|

**date**

```c
void radixSort(struct BookInfo arr[], int n, int index, int count[]) {
    const int RANGE = 10; // 기수 정렬에서 사용할 기수, 0부터 9까지
    // 메모리 할당
    struct BookInfo *output = malloc(n * sizeof(struct BookInfo));

    // 기수 정렬을 위한 배열 동적 할당
    int *countArray = (int *)malloc(RANGE * sizeof(int));

    // 각 기수별로 등장 횟수 초기화
    for (int i = 0; i < RANGE; i++)
        countArray[i] = 0;

    // 각 기수별로 등장 횟수를 세기
    for (int i = 0; i < n; i++)
        countArray[(arr[i].symbol / index) % RANGE]++;

    // 전체 정렬에 대한 기수별 등장 횟수 기록
    for (int i = 0; i < RANGE; i++)
        count[i] += countArray[i];

    // 누적 등장 횟수 계산(한 리스트의 n번큐 인덱스만큼의 구역 설정)
    for (int i = 1; i < RANGE; i++)
        countArray[i] += countArray[i - 1];

    for (int i = n - 1; i >= 0; i--) {
        output[countArray[(arr[i].symbol / index) % RANGE] - 1] = arr[i];
        countArray[(arr[i].symbol / index) % RANGE]--;
    }

    // 정렬된 결과 복사
    for (int i = 0; i < n; i++)
        arr[i] = output[i];

    // 메모리 해제
    free(output);
    free(countArray);
}
```

- **기수 정렬:**
**비교 기반 정렬이 아닌 자릿수를 이용하여 정렬하는 정렬 알고리즘.**

**- 각 숫자의 자릿수를 독립적으로 정렬하며, 가장 낮은 자릿수부터 가장 높은 자릿수까지 차례로 적용한다.**

- **radixSort 함수는 특정 자릿수(index)를 기반으로 기수 정렬**
- **주어진 데이터 배열(arr)의 지정 필드에 대해 기수 정렬 수행, 이 함수는 여러 번 호출되어 가장 낮은 자릿수부터 가장 높은 자릿수까지 차례로 정렬을 수행**
- **기수 정렬은 자릿수마다 계수 정렬(Counting Sort)을 사용하여 정렬하는 특징이 있습니다.**
- **따라서 countArray 배열은 각 기수별로 등장 횟수를 계산하고, 이를 누적하여 정렬을 진행합니다.**

```c
void radixSort(struct BookInfo arr[], int n, int index, int count[]) {
    const int RANGE = 10; // 기수 정렬에서 사용할 기수, 0부터 9까지
    // 메모리 할당
    struct BookInfo *output = malloc(n * sizeof(struct BookInfo));

    // 기수 정렬을 위한 배열 동적 할당
    int *countArray = (int *)malloc(RANGE * sizeof(int));

    // 각 기수별로 등장 횟수 초기화
    for (int i = 0; i < RANGE; i++)
        countArray[i] = 0;

    // 각 기수별로 등장 횟수를 세기
    for (int i = 0; i < n; i++)
        countArray[(arr[i].symbol / index) % RANGE]++;

    // 전체 정렬에 대한 기수별 등장 횟수 기록
    for (int i = 0; i < RANGE; i++)
        count[i] += countArray[i];

    // 누적 등장 횟수 계산(한 리스트의 n번큐 인덱스만큼의 구역 설정)
    for (int i = 1; i < RANGE; i++)
        countArray[i] += countArray[i - 1];

    for (int i = n - 1; i >= 0; i--) {
        output[countArray[(arr[i].symbol / index) % RANGE] - 1] = arr[i];
        countArray[(arr[i].symbol / index) % RANGE]--;
    }

    // 정렬된 결과 복사
    for (int i = 0; i < n; i++)
        arr[i] = output[i];

    // 메모리 해제
    free(output);
    free(countArray);
}
```

## 예) 분류 기호 코드 배열:

| 110 | 100 | 630 | 200 | 550 |
|---|---|---|---|---|

## 기수 1의 자리에 대한 정렬:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 110<br>100<br>630<br>200<br>550 | | | | | | |

## 기수 10의 자리에 대한 정렬:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 100(2)<br>200(4) | 110(1) | | 630(3) | | 550(5) | |

## 기수 100의 자리에 대한 정렬:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | 100(1)<br>110(2) | 200(3) | | | 550(4) | 630(3) |

```c
void radixSort(struct BookInfo arr[], int n, int index, int count[]) {
    const int RANGE = 10; // 기수 정렬에서 사용할 기수, 0부터 9까지
    // 메모리 할당
    struct BookInfo *output = malloc(n * sizeof(struct BookInfo));

    // 기수 정렬을 위한 배열 동적 할당
    int *countArray = (int *)malloc(RANGE * sizeof(int));

    // 각 기수별로 등장 횟수 초기화
    for (int i = 0; i < RANGE; i++)
        countArray[i] = 0;

    // 각 기수별로 등장 횟수를 세기
    for (int i = 0; i < n; i++)
        countArray[(arr[i].symbol / index) % RANGE]++;

    // 전체 정렬에 대한 기수별 등장 횟수 기록
    for (int i = 0; i < RANGE; i++)
        count[i] += countArray[i];

    // 누적 등장 횟수 계산(한 리스트의 n번큐 인덱스만큼의 구역 설정)
    for (int i = 1; i < RANGE; i++)
        countArray[i] += countArray[i - 1];

    for (int i = n - 1; i >= 0; i--) {
        output[countArray[(arr[i].symbol / index) % RANGE] - 1] = arr[i];
        countArray[(arr[i].symbol / index) % RANGE]--;
    }

    // 정렬된 결과 복사
    for (int i = 0; i < n; i++)
        arr[i] = output[i];

    // 메모리 해제
    free(output);
    free(countArray);
}
```

**정렬된 분류 기호 코드 배열:**

| 100 | 110 | 200 | 550 | 630 |
|---|---|---|---|---|

**Category, date 또한 동일한 방식으로 숫자열들의 자릿수에 대해 정렬을 수행하며 최종적으로 데이터를 정렬**

# 정렬 완료한 도서분류, 분류기호, 날짜를 출력

```
switch (selectedField) {
    case 1:
        gettimeofday(&start, NULL);
        radixSortCategory(books, index_a, 1, count);
        radixSortCategory(books, index_a, 10, count);
        radixSortCategory(books, index_a, 100, count);
        gettimeofday(&end, NULL);
        printf("도서 분류 코드를 기준으로 기수 정렬:\n");
        break;
```

도서 분류 정렬

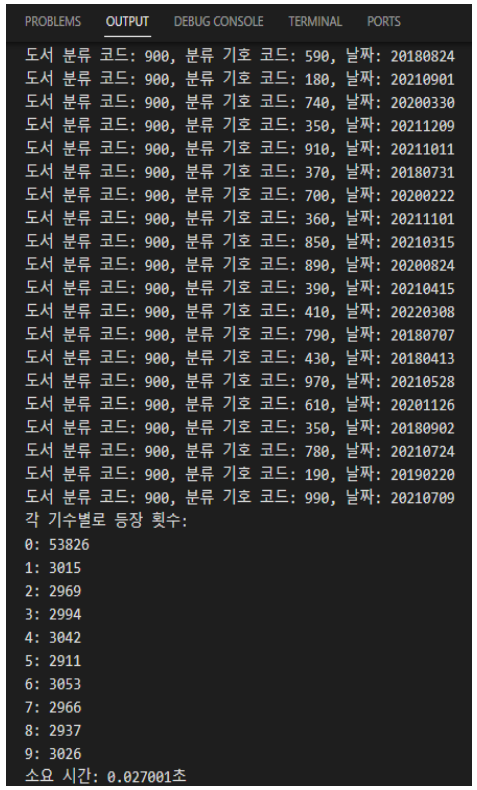
```
case 2:
    gettimeofday(&start, NULL);
    radixSort(books, index_a, 1, count);
    radixSort(books, index_a, 10, count);
    radixSort(books, index_a, 100, count);
    gettimeofday(&end, NULL);
    printf("분류 기호 코드를 기준으로 기수 정렬:\n");
    break;
```

분류 기호 정렬

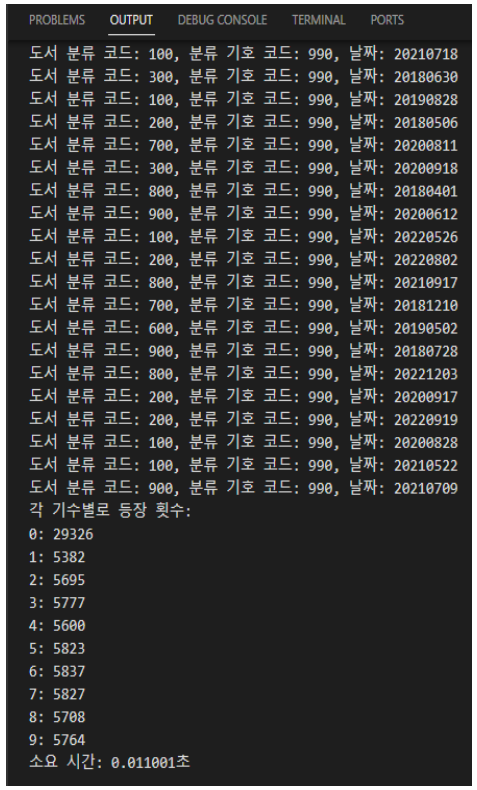
```
case 3:
    gettimeofday(&start, NULL);
    radixSortDate(books, index_a, 1, count);
    radixSortDate(books, index_a, 10, count);
    radixSortDate(books, index_a, 100, count);
    radixSortDate(books, index_a, 1000, count);
    radixSortDate(books, index_a, 10000, count);
    radixSortDate(books, index_a, 100000, count);

    gettimeofday(&end, NULL);
    printf("날짜를 기준으로 기수 정렬:\n");
    break;
```

날짜 정렬

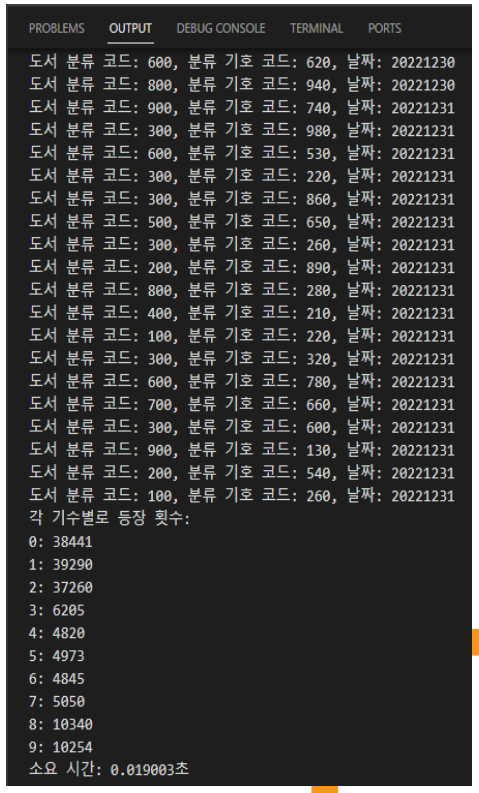
# 정렬 완료한 도서분류, 분류기호, 날짜를 출력




도서 분류 정렬




분류 기호 정렬




날짜 정렬





전체 정렬

* Intel – Pentium® CPU N4200 기준

| 정렬 방법 | 시간 |
|---|---|
| 도서 분류 기준 정렬 | 0.027sec |
| 분류 기호 기준 정렬 | 0.011sec |
| 날짜 기준 정렬 | 0.019sec |
| 전체 정렬 | 4.468sec |

# RUN TIME

소요 시간

# Run Time Graph

| | 전체 | 도서 분류 | 분류 기호 | 연월일 |
|---|---|---|---|---|
| Quick | 0.01 | 0.005 | 0.010 | 0.019 |
| Merge | 0.025 | 0.020 | 0.022 | 0.024 |
| Bubble | 5.414 | 5.471 | 5.725 | 5.728 |
| Radix | 4.468 | 0.027 | 0.011 | 0.019 |

# Thank you!