



순천향대학교
SOON CHUN HYANG
UNIVERSITY

자료구조 실습 과제

자료구조2 실습

담당교수	홍민 교수님
학과	컴퓨터소프트웨어공학과
학번	20204005
이름	김필중
제출일	2021년 11월 30일

| 목 차 |

1. 쉘 정렬 프로그램

- 1.1 문제분석
- 1.2 소스 코드
- 1.3 소스코드 분석
- 1.4 실행 결과
- 1.5 느낀점

2. 합병 정렬 프로그램

- 2.1 문제분석
- 2.2 소스 코드
- 2.3 소스코드 분석
- 2.4 실행 결과
- 2.5 느낀점

3. 퀵 정렬 프로그램

- 3.1 문제분석
- 3.2 소스 코드
- 3.3 소스코드 분석
- 3.4 실행 결과
- 3.5 느낀점

1.1 문제 분석

■ 셸 정렬 프로그램

- data.txt에 학생의 정보가 이름, 학번, 전화번호로 저장되어 있다. 이를 읽어와 셸정렬을 이용하여 학번 순으로 내림차순 정렬하여 출력하시오.

```
C:\WINDOWS\system32\cmd.exe
<정렬 전>
이성진 20124045 010-2085-1357
감임박 20134047 010-3578-0000
서하늘 20112045 010-4788-0000
이대호 20112043 011-0111-0000

<정렬 후>
감임박 20134047 010-3578-0000
이성진 20124045 010-2085-1357
서하늘 20112045 010-4788-0000
이대호 20112043 011-0111-0000
계속하려면 아무 키나 누르십시오 . . .
```

```
data4.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
이성진 20124045 010-2085-1357
감임박 20134047 010-3578-0000
서하늘 20112045 010-4788-0000
이대호 20112043 011-0111-0000
```

이번 문제는 셸 정렬을 이용해서 데이터를 학번의 내림차순으로 정렬하는 문제이다. 우선 먼저 고려해야 할 사항은 동적할당을 이용해 데이터의 개수만큼 동적할당을 하는 것이 첫번째이다. 셸 정렬을 위해서 셸 삽입 정렬 함수, 셸 정렬 함수 2개를 추가해서 셸 정렬을 구현해야 한다

우선 구조체를 생성해서 구조체 안에 이름 저장 변수, 학번 저장 변수, 전화번호 저장 변수 3개를 선언한다

셸 삽입 정렬 함수를 만들어서 사이 간격 크기만큼 요소들을 삽입 정렬을 반복해 준다. 임시 변수에 데이터들을 저장을 하고 내림 차순으로 정렬을 한 후 변경된 값을 위치를 변경해 준다.

셸 정렬 함수는 크기를 2로 나눈 값을 초기값으로 하고 크기가 0이 아닐 때까지 반복하면서 차이 값만큼 셸 삽입 정렬 함수를 호출 해 주는 함수이다.

메인 함수에서는 파일을 읽고 데이터의 개수를 카운트해준 후 크기만큼 동적할당을 해주고 데이터를 동적할당한 변수에 넣어 주고 셀 정렬을 호출한다.

1.2 소스 코드

```
1 //=====
2 // 제작 기간: 21년 11월 24일 ~ 11월 29일
3 // 제작자 : 20204005 김필중
4 // 프로그램명: 셸 정렬 프로그램
5 //=====
6
7 // 필요한 헤더파일 선언
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11
12 // 오류 방지 구문 제작
13 #pragma warning (disable : 4996)
14
15 // 리스트 구조체 생성
16 typedef struct list
17 {
18     char name[20]; // 이름 저장 변수
19     int num; // 학번 저장 변수
20     char phone[20]; // 전화번호 저장 변수
21 }List;
22
23 // 셸 삽입 정렬
24 void inc_insertion_sort(List list[], int first, int last, int gap)
25 {
26     // 필요한 변수 선언
27     int i, j, key_num;
28     char key_name[20];
29     char key_phone[20];
30     // gap만큼 떨어진 요소들을 삽입 정렬을 한다. 반복 범위는 first + gap에서 last까지
31     for (i = first + gap; i <= last; i = i + gap)
32     {
33         // 현재 삽입될 i번째 데이터들을 key 변수들에 저장을 한다
34         key_num = list[i].num;
35         strcpy(key_phone, list[i].phone);
36         strcpy(key_name, list[i].name);
37         for (j = i - gap; j >= first && key_num > list[j].num; j = j - gap) // 내림차순으로 정렬을 한다
38             list[j + gap] = list[j]; // 데이터를 gap만큼 오른쪽으로 이동시키고 저장한다
39         // j + gap 번째 배열에 키값을 저장
40         list[j + gap].num = key_num;
41         strcpy(list[j + gap].phone, key_phone);
42         strcpy(list[j + gap].name, key_name);
43     }
44 }
45
46 // 셸 정렬 함수
47 void shell_sort(List list[], int n) // n은 크기
48 {
49     // 필요한 변수를 생성
50     int i, gap;
51     // 크기를 2로 나눈 값을 차이로 잡고 0 이하가 될때까지 반복을 한다.
52     for (gap = n / 2; gap > 0; gap = gap / 2)
53     {
54         // 만약 나머지가 0이면 차이값을 1 올려준다
55         if ((gap % 2) == 0)
56             gap++;
57         // 차이값 만큼 셸 삽입 정렬 함수를 호출해준다.
58         for (i = 0; i < gap; i++)
59             inc_insertion_sort(list, i, n - 1, gap);
60     }
61 }
62
```

```

64 // 메인 함수
65 int main(void)
66 {
67     FILE *fp; // 파일 포인터 선언
68     List *user; // user 포인터 구조체 생성
69
70     // 필요한 변수들 생성
71     int temp_number;
72     char temp_name[10];
73     char temp_phone[20];
74     int cnt = 0;
75     int n = 0;
76
77     // 파일을 열고 실패 시 종료
78     fp = fopen("data01.txt", "r");
79
80     if (fp == NULL)
81     {
82         printf("파일 오픈 실패\n");
83         return;
84     }
85
86     // 파일 끝까지 반복
87     while (!feof(fp))
88     {
89         // 파일 데이터를 읽고 저장한다
90         fscanf(fp, "%s %d %s", temp_name, &temp_number, temp_phone);
91         // 개수를 증가시킨다
92         cnt++;
93     }
94
95     // 파일 포인터를 앞으로 돌린다
96     rewind(fp);
97
98     // 개수만큼 동적할당을 한다
99     user = (List *)malloc(sizeof(List) * cnt);
100
101     // 파일 끝까지 반복
102     while (!feof(fp))
103     {
104         // 파일을 읽고 동적할당한 구조체에 저장한다
105         fscanf(fp, "%s %d %s", temp_name, &temp_number, temp_phone);
106         user[n].num = temp_number;
107         strcpy(user[n].name, temp_name);
108         strcpy(user[n].phone, temp_phone);
109         n++;
110     }
111

```

```
112 // 값들을 출력한다
113 printf("< 정렬 전 >\n");
114 for (int i = 0; i < cnt; i++)
115     printf("%s %d %s\n", user[i].name, user[i].num, user[i].phone);
116
117 // 셸 정렬을 호출한다
118 shell_sort(user, cnt);
119
120 // 값들을 출력한다
121 printf("< 정렬 후 >\n");
122 for(int i = 0; i < cnt; i++)
123     printf("%s %d %s\n", user[i].name, user[i].num, user[i].phone);
124
125 // 동적할당을 해제하고 파일을 닫는다
126 free(user);
127 fclose(fp);
128 return 0;
129 }
130
```

1.3 소스 코드 분석

```
7 // 필요한 헤더파일 선언
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11
12 // 오류 방지 구문 제작
13 #pragma warning (disable : 4996)
14
```

1. 필요한 헤더파일을 선언한다
2. 오류 방지 구문을 제작한다

```
// 리스트 구조체 생성
typedef struct list
{
    char name[20]; // 이름 저장 변수
    int num; // 학번 저장 변수
    char phone[20]; // 전화번호 저장 변수
}List;
```

3. 리스트 구조체를 생성한다
4. 구조체 안에는 필요한 데이터 저장 변수들을 선언을 한다

```
// 셀 삽입 정렬
void inc_insertion_sort(List list[], int first, int last, int gap)
{
    // 필요한 변수 선언
    int i, j, key_num;
    char key_name[20];
    char key_phone[20];
    // gap만큼 떨어진 요소들을 삽입 정렬을 한다. 반복 범위는 first + gap에서 last까지
    for (i = first + gap; i <= last; i = i + gap)
    {
        // 현재 삽입될 i번째 데이터들을 key 변수들에 저장을 한다
        key_num = list[i].num;
        strcpy(key_phone, list[i].phone);
        strcpy(key_name, list[i].name);
        for (j = i - gap; j >= first && key_num > list[j].num; j = j - gap) // 내림차순으로 정렬을 한다
            list[j + gap] = list[j]; // 데이터를 gap만큼 오른쪽으로 이동시키고 저장한다
        // j + gap 번째 배열에 키값을 저장
        list[j + gap].num = key_num;
        strcpy(list[j + gap].phone, key_phone);
        strcpy(list[j + gap].name, key_name);
    }
}
```


5. 셸 삽입 정렬 함수를 생성한다
6. 필요한 변수들을 선언한다. 임시로 값을 저장하는 변수 등
7. gap만큼 떨어진 요소들을 삽입 정렬을 하는 반복문을 만든다. 반복 범위는 첫 번째 값 + gap 부터 마지막까지이다.
8. 현재 삽입될 i번째 데이터들을 위에서 만든 key 임시 저장소에 저장을 한다.
9. 내림차순으로 정렬하기 위해 크기를 비교하고 학번이 현재 저장 되어있는 학번의 값이 j번째 학번의 값보다 클경우 바꾼다.
10. 그 후 j+gap 번째 배열에 키값들을 다시 넣는다

```
// 셸 정렬 함수
void shell_sort(List list[], int n) // n은 크기
{
    // 필요한 변수를 생성
    int i, gap;
    // 크기를 2로 나눈 값을 차이로 잡고 0 이하가 될때까지 반복을 한다.
    for (gap = n / 2; gap > 0; gap = gap / 2)
    {
        // 만약 나머지가 0이면 차이값을 1 늘려준다
        if ((gap % 2) == 0)
            gap++;
        // 차이값 만큼 셸 삽입 정렬 함수를 호출해준다.
        for (i = 0; i < gap; i++)
            inc_insertion_sort(list, i, n - 1, gap);
    }
}
```

11. 셸 정렬 함수를 선언한다
12. 필요한 변수들을 생성한다
13. gap은 받은 크기를 반으로 나눈 값으로 설정하고 gap이 0 이하일때 까지 반복을 한다.
14. 만약 나머지가 0이면 차이값을 1 늘려준다
15. 차이값 만큼 셸 삽입 정렬 함수를 호출한다.

```

FILE *fp; // 파일 포인터 선언
List *user; // user 포인터 구조체 생성

// 필요한 변수들 생성
int temp_number;
char temp_name[10];
char temp_phone[20];
int cnt = 0;
int n = 0;

// 파일을 열고 실패 시 종료
fp = fopen("data01.txt", "r");

if (fp == NULL)
{
    printf("파일 오픈 실패\n");
    return;
}

```

16. 메인 함수에서 먼저 파일 포인터를 선언한다
17. user 포인터 구조체를 생성한다
18. 필요한 변수들을 생성한다
19. 파일을 열고 실패 시 종료를 한다

```

// 파일 끝까지 반복
while (!feof(fp))
{
    // 파일 데이터를 읽고 저장한다
    fscanf(fp, "%s %d %s", temp_name, &temp_number, temp_phone);
    // 개수를 증가시킨다
    cnt++;
}

// 파일 포인터를 앞으로 돌린다
rewind(fp);

```

20. 파일 끝까지 반복하면서 데이터를 읽고 저장한 후 카운트의 개수를 증가시킨다
21. 파일 포인터를 맨 앞으로 돌린다

```
// 개수만큼 동적할당을 한다
user = (List *)malloc(sizeof(List) * cnt);
```

22. 위에서 카운트 한 개수만큼 동적할당을 한다

```
// 파일 끝까지 반복
while (!feof(fp))
{
    // 파일을 읽고 동적할당한 구조체에 저장한다
    fscanf(fp, "%s %d %s", temp_name, &temp_number, temp_phone);
    user[n].num = temp_number;
    strcpy(user[n].name, temp_name);
    strcpy(user[n].phone, temp_phone);
    n++;
}
```

23. 파일 끝까지 반복하면서 위에서 동적할당한 구조체에 값을 저장한다.

```
112 // 값들을 출력한다
113 printf("< 정렬 전 >\n");
114 for (int i = 0; i < cnt; i++)
115     printf("%s %d %s\n", user[i].name, user[i].num, user[i].phone);
116
117 // 셸 정렬을 호출한다
118 shell_sort(user, cnt);
119
120 // 값들을 출력한다
121 printf("< 정렬 후 >\n");
122 for(int i = 0; i < cnt; i++)
123     printf("%s %d %s\n", user[i].name, user[i].num, user[i].phone);
124
125 // 동적할당을 해제하고 파일을 닫는다
126 free(user);
127 fclose(fp);
128 return 0;
129 }
130
```

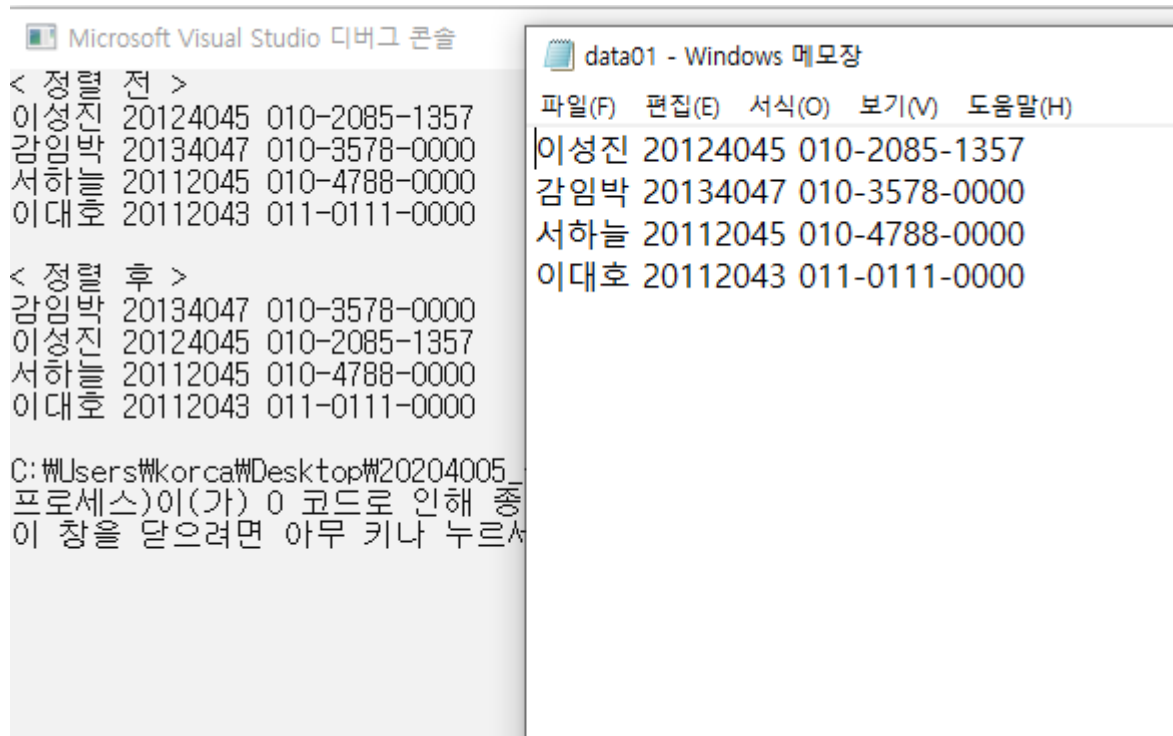
24. 그 후 정렬 전 값들을 출력한다

25. 셸 정렬을 호출하고 정렬 후 값들을 호출한다

26. 동적할당을 해제하고 파일을 닫는다

27. 프로그램을 종료한다

1.4 실행 결과



The screenshot shows two windows. The left window is the 'Microsoft Visual Studio 디버그 콘솔' (Debug Console) showing the output of a sorting process. It lists names and IDs before and after sorting. The right window is a 'data01 - Windows 메모장' (Notepad) showing the same data after sorting.

```
Microsoft Visual Studio 디버그 콘솔
< 정렬 전 >
이성진 20124045 010-2085-1357
김임박 20134047 010-3578-0000
서하늘 20112045 010-4788-0000
이대호 20112043 011-0111-0000

< 정렬 후 >
김임박 20134047 010-3578-0000
이성진 20124045 010-2085-1357
서하늘 20112045 010-4788-0000
이대호 20112043 011-0111-0000

C:\Users\korca\Desktop\20204005_
프로세스)이(가) 0 코드로 인해 종
이 창을 닫으려면 아무 키나 누르세
```

```
data01 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
이성진 20124045 010-2085-1357
김임박 20134047 010-3578-0000
서하늘 20112045 010-4788-0000
이대호 20112043 011-0111-0000
```

1.5 느낀점

이번 과제를 하면서 쉘 정렬이 무엇인지 정확히 알게 되었다. 갭을 이용해 값들을 비교하고 계속 변경해 주면서 내림차순으로 만들 수 있다는 방식을 공부하면서 확실히 앞에서 한 정렬들과는 다르다는 것을 알게 되었다. 진행할수록 갭의 크기가 줄어들고 이를 통해 정렬을 하는 방식이었다. 이것은 삽입 정렬보다 더 빠르다는 사실을 알 수 있었다. 또한 알고리즘이 어렵지 않아 많은 부분에서 사용이 편리하다고 생각했다.

2.1 문제 분석

■ 합병 정렬 프로그램

- data.txt에 저장되어있는 데이터를 불러와 합병정렬을 이용하여 오름차순으로 정렬하여 출력하시오.
 - 배열을 이용하여 작성하시오.

```
C:\WINDOWS\system32\cmd.exe
<< 정렬되지 않은 리스트 >>
<9> <28> <14> <22> <1> <42> <15> <6>

<< 정렬 과정 >>
-----
Left list : 9
Right list : 28
Sorted list : 9 28

Left list : 14
Right list : 22
Sorted list : 14 22

Left list : 9 28
Right list : 14 22
Sorted list : 9 14 22 28

Left list : 1
Right list : 42
Sorted list : 1 42

Left list : 15
Right list : 6
Sorted list : 6 15

Left list : 1 42
Right list : 6 15
Sorted list : 1 6 15 42

Left list : 9 14 22 28
Right list : 1 6 15 42
Sorted list : 1 6 9 14 15 22 28 42
-----
<< 정렬된 리스트 >>
<1> <6> <9> <14> <15> <22> <28> <42>
계속하려면 아무 키나 누르십시오 . . .

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
9 28 14 22 1 42 15 6
```

이번 문제는 합병 정렬을 이용해 데이터를 정렬하는 것이 목표인 과제이다. 앞 문제와 마찬가지로 우선 동적 배열을 통해 데이터를 저장하는 것이 첫번째이다.

합병 정렬을 이용해 데이터들을 정렬하는 것이 이번 과제에 포인트이다. 그래서 정렬되지 않은 데이터를 분할한 후 부분 배열을 정렬하는 정복 방법을 이용해 정렬이 완료되면 결합하는 방식이다.

우선 추가 배열이 필요하기 때문에 배열을 하나 생성한다

합병 정렬 과정 함수를 만들어서 분할된 리스트를 받아 분할 정렬된 리스트를 합병 시킨다. 그 후 남아 있는 값들을 복사하고 위에서 만든 배열의 값들을 원래 리스트에 붙여 넣는다. 그리고 문제 형식에 맞게 출력을 해준다.

합병 정렬 함수에서는 왼쪽이 오른쪽 보다 작을 경우 조건을 넣은 후 mid 값에 리스트의 균등 분할된 값을 넣는다. 그리고 위에 만든 합병 정렬 과정 함수를 리

스트와 왼쪽, mid 값을 넣어서 호출한다. 이렇게 분할된 2개의 리스트를 정렬 시키고 다시 합병 정렬 함수를 재귀 호출해준다.

메인 함수에서는 int형 포인터를 선언해서 데이터의 수 만큼 동적할당을 해주고 해제하는 것이 중요하다.

2.2 소스 코드

```
1  //=====
2  // 제작 기간: 21년 11월 24일 ~ 11월 29일
3  // 제작자 : 20204005 김필중
4  // 프로그램명: 합병 정렬 프로그램
5  //=====
6
7  // 필요한 헤더파일 선언
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11
12 // 필요한 정의 선언
13 #define MAX_SIZE 100
14
15 // 오류 방지 구문 제작
16 #pragma warning(disable : 4996)
17
18 // 추가 공간이 필요하기 때문에 정렬 배열 생성
19 int sorted[MAX_SIZE];
20
21
22 // 합병 정렬 과정
23 void merge(int list[], int left, int mid, int right)
24 {
25     // i는 정렬된 왼쪽 리스트에 대한 인덱스
26     // j는 정렬된 오른쪽 리스트에 대한 인덱스
27     // k는 정렬될 리스트에 대한 인덱스
28     int i, j, k, l;
29     i = left; j = mid + 1; k = left;
30
31     // 형식에 맞게 값들을 출력한다
32     printf("left: ");
33     for (int m = left; m <= mid; m++)
34         printf("%d ", list[m]);
35     printf("\n");
36
37     printf("right: ");
38     for (int m = mid+1; m <= right; m++)
39         printf("%d ", list[m]);
40     printf("\n");
41 }
```

```

42
43 // 분할 정렬된 list의 합병
44 // i 값이 중앙 값 이하 그리고 j 값이 오른쪽 값 이하일 경우
45 while (i <= mid && j <= right)
46 {
47     // 만약 i번째 값이 j 번째 값 이하일 경우
48     if (list[i] <= list[j])
49         // i번째 값 삽입
50         sorted[k++] = list[i++];
51     // 아닐 경우
52     else
53         // j 번째 값 삽입
54         sorted[k++] = list[j++];
55 }
56
57 // 남아 있는 레코드의 일괄 복사
58 // i보다 미드가 작을 경우
59 if (i > mid)
60     // 오른쪽 값만큼 반복문 실행하면서 정렬 배열에 i번째 리스트 배열의 값을 저장한다
61     for (l = j; l <= right; l++)
62         sorted[k++] = list[l];
63 // 아닐 경우는
64 else
65     // 미드 값까지 반복문 실행하면서 정렬 배열에 i번째 리스트 배열의 값을 저장한다
66     for (l = i; l <= mid; l++)
67         sorted[k++] = list[l];
68
69 // 배열 sorted[]의 리스트를 배열 list[]로 재복사
70 for (l = left; l <= right; l++)
71     list[l] = sorted[l];
72
73 // 형식에 맞게 값들을 출력한다
74 printf("Sorted List: ");
75 for(int m = left; m <= right; m++)
76     printf("%d ", list[m]);
77 printf("\n\n");
78 }
79
80 // 프로그램의 시작

```



```

10 // 합병 정렬 함수
80 void merge_sort(int list[], int left, int right)
81 {
82     int mid;
83     if (left < right)
84     {
85         mid = (left + right) / 2; // 리스트의 균등 분할
86         merge_sort(list, left, mid); // 부분 리스트 정렬
87         merge_sort(list, mid + 1, right); // 부분 리스트 정렬
88         merge(list, left, mid, right); // 합병 호출
89     }
90 }
91
92 // 메인 함수
93 int main(void)
94 {
95     FILE *fp; // 파일 포인터 선언
96     int *number; // int 형 포인터 선언
97
98     // 필요한 변수 선언
99     int temp;
100     int cnt = 0;
101
102     // 파일을 열고 실패 시 종료
103     fp = fopen("data02.txt", "r");
104
105     if (fp == NULL)
106     {
107         printf("파일 오픈 실패\n");
108         return;
109     }
110
111     // 파일 끝까지 반복
112     while (!feof(fp))
113     {
114         // 값들을 읽으면서 개수를 카운트 해준다
115         fscanf(fp, "%d", &temp);
116         cnt++;
117     }
118
119     // 파일 포인터 맨 앞으로 돌린다
120     rewind(fp);
121
122     // 개수만큼 동적할당을 해준다
123     number = (int *)malloc(sizeof(int) * cnt);
124
125     // cnt 값 초기화
126     cnt = 0;
127
128     ...
129
130

```

```

130
131 // 파일 끝까지 반복한다
132 while (!feof(fp))
133 {
134     // 값들을 읽고 동적할당한 number 에 저장을 한다
135     fscanf(fp, "%d", &temp);
136     number[cnt] = temp;
137     cnt++;
138 }
139
140 // 값을 프린트
141 printf("< 정렬되지 않은 리스트 > \n");
142 for (int i = 0; i < cnt; i++)
143     printf("<%d> ", number[i]);
144 printf("\n\n");
145
146 printf("<< 정렬 과정 >> \n");
147 printf("===== \n");
148 merge_sort(number, 0, cnt-1); // 호출
149
150
151 printf("===== \n");
152 printf("<< 정렬된 리스트 >> \n");
153 // 정렬 값들 프린트
154 for (int i = 0; i < cnt; i++)
155 {
156     printf("<%d> ", number[i]);
157 }
158 printf("\n\n");
159
160 // 동적할당을 해제하고 파일을 닫고 종료한다
161 free(number);
162 fclose(fp);
163 return 0;
164
165 }

```

2.3 소스 코드 분석

```
// 필요한 헤더파일 선언
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 필요한 정의 선언
#define MAX_SIZE 100

// 오류 방지 구문 제작
#pragma warning (disable : 4996)

// 추가 공간이 필요하기 때문에 정렬 배열 생성
int sorted[MAX_SIZE];
```

1. 필요한 헤더파일을 선언한다
2. 오류 방지 구문을 선언한다
3. 추가 공간이 필요해 int형 배열을 하나 선언한다

```

// 합병 정렬 과정
void merge(int list[], int left, int mid, int right)
{
    // i는 정렬된 왼쪽 리스트에 대한 인덱스
    // j는 정렬된 오른쪽 리스트에 대한 인덱스
    // k는 정렬될 리스트에 대한 인덱스
    int i, j, k, l;
    i = left; j = mid + 1; k = left;

    // 형식에 맞게 값들을 출력한다
    printf("left: ");
    for (int m = left; m <= mid; m++)
        printf("%d ", list[m]);
    printf("\n");

    printf("right: ");
    for (int m = mid+1; m <= right; m++)
        printf("%d ", list[m]);
    printf("\n");

    // 분할 정렬된 list의 합병
    // i 값이 중앙 값 이하 그리고 j 값이 오른쪽 값 이하일 경우
    while (i <= mid && j <= right)
    {
        // 만약 i번째 값이 j 번째 값 이하일 경우
        if (list[i] <= list[j])
            // i번째 값 삽입
            sorted[k++] = list[i++];
        // 아닐 경우
        else
            // j 번째 값 삽입
            sorted[k++] = list[j++];
    }
}

```

4. i는 왼쪽 리스트의 인덱스 j는 오른쪽 리스트의 인덱스 k는 정렬될 리스트의 인덱스로 잡는다
5. 형식에 맞게 값들을 출력한다
6. 분할 정렬된 list의 합병을 위해 우선 i값이 중앙 값 이하 그리고 j 값이 오른쪽 값 이하일 경우 계속 반복해준다.
7. 만약 i번째 배열의 값이 j번째 배열의 값 이하일 경우 i번째 값을 정렬에 삽입하고 인덱스를 증가시킨다
8. 아닐경우는 j번째 값을 정렬에 삽입하고 인덱스를 증가시킨다.

```

// 남아 있는 레코드의 일괄 복사
// i보다 미드가 작을 경우
if (i > mid)
    // 오른쪽 값만큼 반복문 실행하면서 정렬 배열에 i번째 리스트 배열의 값을 저장한다
    for (l = j; l <= right; l++)
        sorted[k++] = list[l];
// 아닐 경우는
else
    // 미드 값까지 반복문 실행하면서 정렬 배열에 i번째 리스트 배열의 값을 저장한다
    for (l = i; l <= mid; l++)
        sorted[k++] = list[l];

// 배열 sorted[]의 리스트를 배열 list[]로 재복사
for (l = left; l <= right; l++)
    list[l] = sorted[l];

// 형식에 맞게 값들을 출력한다
printf("Sorted List: ");
for(int m = left; m <= right; m++)
    printf("%d ", list[m]);
printf("\n\n");

```

9. 만약 i보다 mid가 작을 경우는 오른쪽 값 만큼 반복문을 실행해서 정렬 배열에 i번째 리스트 배열의 값을 저장한다

10. 아닐 경우는 미드 값까지 반복하면서 i번째 리스트의 배열의 값을 저장한다

11 sorted의 리스트를 배열 list로 다시 복사를 한다

12. 형식에 맞게 값들을 출력한다

```

// 합병 정렬 함수
void merge_sort(int list[], int left, int right)
{
    int mid;
    if (left < right)
    {
        mid = (left + right) / 2; // 리스트의 균등 분할
        merge_sort(list, left, mid); // 부분 리스트 정렬
        merge_sort(list, mid + 1, right); // 부분 리스트 정렬
        merge(list, left, mid, right); // 합병 호출
    }
}

// 메인 함수

```

13. 합병 정렬 함수를 선언한다

14. 미드값을 선언하고 미드값을 좌 + 우 / 2로 중간을 알아낸다

15. 왼쪽 리스트를 정렬, 오른쪽 리스트를 정렬한다

16. 그 후 다시 합병 정렬 함수를 호출한다

```
FILE *fp; // 파일 포인터 선언
int *number; // int 형 포인터 선언

// 필요한 변수 선언
int temp;
int cnt = 0;

// 파일을 열고 실패 시 종료
fp = fopen("data02.txt", "r");

if (fp == NULL)
{
    printf("파일 오픈 실패\n");
    return;
}
```

17. 파일 포인터를 선언한다

18. int형 포인터를 선언한다

19. 필요한 변수들을 선언한다

20. 파일을 열고 실패 시 종료한다

```
// 파일 끝까지 반복
while (!feof(fp))
{
    // 값들을 읽으면서 개수를 카운트 해준다
    fscanf(fp, "%d", &temp);
    cnt++;
}

// 파일 포인터 맨 앞으로 돌린다
rewind(fp);
```

21. 파일을 끝까지 반복하고 값들을 읽으면서 하나씩 카운트 한다

22. 파일 포인터 맨 앞으로 보낸다

```
// 개수만큼 동적할당을 해준다
number = (int *)malloc(sizeof(int) * cnt);
```

23. 개수 만큼 동적할당을 해준다

```
// 파일 끝까지 반복한다
while (!feof(fp))
{
    // 값들을 읽고 동적할당한 number 에 저장한다
    fscanf(fp, "%d", &temp);
    number[cnt] = temp;
    cnt++;
}

// 값들 프린트
printf("< 정렬되지 않은 리스트 > \n");
for (int i = 0; i < cnt; i++)
    printf("<%d> ", number[i]);
printf("\n\n");

printf("<< 정렬 과정 >> \n");
printf("===== \n");
merge_sort(number, 0, cnt-1); // 호출

printf("===== \n");
printf("<< 정렬된 리스트 >> \n");
// 정렬 값들 프린트
for (int i = 0; i < cnt; i++)
{
    printf("<%d> ", number[i]);
}
printf("\n\n");
```

24. 파일 끝까지 반복하면서 위에서 동적할당한 곳에 값들을 저장한다.

25. 값들을 형식에 맞게 프린트 한다.

26. 중간에 합병 정렬을 호출하고 정렬된 값들을 출력한다

```
// 동적할당을 해제하고 파일을 닫고 종료한다
free(number);
fclose(fp);
return 0;
```

27. 동적할당을 해제하고 파일을 닫고 종료한다

2.4 실행결과

```
Microsoft Visual Studio 디버그 콘솔
< 정렬되지 않은 리스트 >
<9> <28> <14> <22> <1> <42> <15> <6>

<< 정렬 과정 >>

=====
left: 9
right: 28
Sorted List: 9 28

left: 14
right: 22
Sorted List: 14 22

left: 9 28
right: 14 22
Sorted List: 9 14 22 28

left: 1
right: 42
Sorted List: 1 42

left: 15
right: 6
Sorted List: 6 15

left: 1 42
right: 6 15
Sorted List: 1 6 15 42

left: 9 14 22 28
right: 1 6 15 42
Sorted List: 1 6 9 14 15 22 28 42

=====
<< 정렬된 리스트 >>
<1> <6> <9> <14> <15> <22> <28> <42>
```

```
data02 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
9 28 14 22 1 42 15 6
```

2.5 느낀점

합병 정렬 알고리즘에서 임시 배열의 필요성을 느꼈다. 제자리에서 정렬하는게 아닌 임시로 저장한 후 다시 합치는 것이기 때문이라고 느꼈다. 또한 데이터가 많으면 느릴 수도 있겠다 라고 생각했다. 하지만 앞에서 진행한 버블정렬, 순차정렬보다는 훨씬 빠르고 사용할 곳이 있을 것이라 생각해서 나중에 관련된 프로그램을 만들 경우 한번 써봐야겠다고 느꼈다

3.1 문제 분석

■ 퀵 정렬 프로그램

- data.txt에 저장되어있는 데이터를 불러와 퀵정렬을 이용하여 오름차순으로 정렬하여 출력하시오.

```
C:\WINDOWS\system32\cmd.exe
< 정렬되지 않은 리스트 >
15 > 4 > 8 > 22 > 7 > 54 > 3 > 80 > 48 > 42 > 2 > 98 > 5 >

< 정렬 과정 >
-Pivot : 15-
low:22 high:5 [15 > 4 > 8 > 22 > 7 > 54 > 3 > 80 > 48 > 42 > 2 > 98 > 5 > ]
low:54 high:2 [15 > 4 > 8 > 5 > 7 > 54 > 3 > 80 > 48 > 42 > 2 > 98 > 22 > ]
low:80 high:3 [15 > 4 > 8 > 5 > 7 > 2 > 3 > 80 > 48 > 42 > 54 > 98 > 22 > ]

-Pivot : 3-
low:4 high:2 [3 > 4 > 8 > 5 > 7 > 2 > ]
low:8 high:2 [3 > 2 > 8 > 5 > 7 > 4 > ]

-Pivot : 8-
low: over high:4 [8 > 5 > 7 > 4 > ]

-Pivot : 4-
low:5 high:4 [4 > 5 > 7 > ]

-Pivot : 5-
low:7 high:5 [5 > 7 > ]

-Pivot : 80-
low:98 high:22 [80 > 48 > 42 > 54 > 98 > 22 > ]

-Pivot : 22-
low:48 high:22 [22 > 48 > 42 > 54 > ]

-Pivot : 48-
low:54 high:42 [48 > 42 > 54 > ]

< 정렬된 리스트 >
2 > 3 > 4 > 5 > 7 > 8 > 15 > 22 > 42 > 48 > 54 > 80 > 98
계속하려면 아무 키나 누르십시오 . . .
```

data3 - 메모장

파일(F)	편집(E)	서식(O)	보기(V)	도움말(H)
15 4 8 22 7 54 3 80 48 42 2 98 5				

우선 마찬가지로 이번 과제도 동적할당을 통해 데이터를 저장해야 한다. 또한 데이터의 과정을 보여줘야 하기 때문에 퀵 정렬 중간에 프린트 부분을 구현해야 한다.

퀵 정렬을 통해 데이터를 정렬하는데 퀵 정렬은 다른 정렬보다 빠른 수행속도를 보여주고 피벗이라는 한 개의 기준으로 정렬을 진행한다.

우선 파티션 함수를 만들어 준다. 피벗, high, low 변수를 선언해서 low는 왼쪽 high는 오른쪽+1 값으로 정의해준다. 피벗값은 받은 리스트의 left값으로 정해준다. Do_while 문을 통해 low값을 일단 하나 올려서 피벗값과 비교하고 클 경우 종료 마찬가지로 high값을 하나 내려서 피벗 값과 비교해서 값이 작으면 빠져나오게 한다. 그 이후 만약 low와 high를 비교했을 때 high가 크다면 둘을 스왑 매크로를 이용해 바꿔준다. 이 과정을 low < high 까지 반복을 하는 알고리즘이다.

그 이후 값을 스왑해주고 high를 반환해 주는 함수이다. 중간중간 문제 형식에 맞게 값들을 출력하는 부분을 잘 고려해서 넣어야 한다. 전체 출력이 아니기 때문에 위치와 변수를 계속해서 고려를 해줘야 한다.

퀵 정렬 함수는 만약 왼쪽이 오른쪽 보다 작다면 실행, 변수에 파티션에서 나오는 값을 받은 후 재귀 함수를 호출해준다. 이렇게 호출하면서 퀵 정렬을 진행한다

메인 함수에서는 파일 입출력과 int형 배열을 데이터의 개수만큼 동적할당 해주는 것이 중요한 포인트이다.

3.2 소스 코드

```
1 //=====
2 // 제작 기간: 21년 11월 24일 ~ 11월 29일
3 // 제작자 : 20204005 김필중
4 // 프로그램명: 퀵 정렬 프로그램
5 //=====
6
7 // 필요한 헤더파일 선언
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <time.h>
11
12 // 오류 방지 구문 제작
13 #pragma warning(disable : 4996)
14
15 // 스왑을 정의한다
16 #define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) )
17
18 // 필요한 배열, 변수를 선언한다
19 int tmp[100];
20 int n = 0;
21
22 // 파티션 함수 선언
23 int partition(int list[], int left, int right)
24 {
25     // 피벗, low, high 변수 생성
26     int pivot, temp;
27     int low, high;
28     int check = 0;
29
30     // 왼쪽은 low, 오른쪽 + 1 값은 high로 값을 정한다
31     low = left;
32     high = right + 1;
33     // 피벗값은 list의 left번째 값으로 정해준다
34     pivot = list[left];
35     // tmp 배열에 pivot 값을 저장한다
36     tmp[n] = pivot;
37     n++;
38     // 값을 출력
39     printf("- Pivot : %d -%n", pivot);
40     // 일단 한번 실행
41     do {
42         // 일단 한번 실행
43         do
44         {
45             low++; // 우선 low 값을 1개 올려준다
46         }
47         while (list[low] < pivot); // list의 low 번째 값이 pivot 값보다 크면 빠져나온다
48
49         do // 일단 한번 실행
50         {
51             // 일단 한번 실행
52             high--; // 우선 high 값을 1개 내려준다
53         }
54         while (list[high] > pivot); // list의 high 번째 값이 pivot 값보다 작으면 빠져나온다
55     }
```

```

55
56
57 // 만약 tmp 배열에 pivot 값이 있을 경우 체크 표시를 한다
58 for (int i = 0; i < 101; i++)
59 {
60     if (tmp[i] == list[low])
61         check = 1;
62 }
63
64 // 체크 표시가 있을 경우 over 출력하고 값 출력
65 if(check == 1)
66     printf("low: over high: %d [ ", list[high]);
67 // 아닐 경우는 그냥 출력
68 else
69     printf("low: %d high: %d [ ", list[low], list[high]);
70
71 // left부터 right+1 까지 반복하면서 리스트 값 출력
72 for (int i = left; i < right+1; i++)
73 {
74     printf("%d > ", list[i]);
75 }
76
77 printf("] \n\n");
78
79 // 만약 high 가 low 보다 크다면 값 교체
80 if (low < high)
81     SWAP(list[low], list[high], temp);
82
83
84 } while (low < high); // low < high가 아닐때 까지 반복
85
86 // 스왑 해준다
87 SWAP(list[left], list[high], temp);
88
89 // high값을 반환한다
90 return high;
91
92 }
93

```

```

100 ~
101 // 퀵 정렬 함수
102 void quick_sort(int list[], int left, int right)
103 {
104     // 만약 left가 right 보다 작다면
105     if (left < right)
106     {
107         // q 값을 파티션을 통해 받는다
108         int q = partition(list, left, right);
109         // 재귀 함수를 통해 다시 호출한다
110         quick_sort(list, left, q - 1);
111         quick_sort(list, q + 1, right);
112     }
113 }
114
115 // 메인 함수
116 int main(void)
117 {
118     FILE *fp; // 파일 포인터 선언
119     int *list; // int형 list 포인터 변수 선언
120
121     // 필요한 변수 선언
122     int temp;
123     int cnt = 0;
124
125     // 파일 오픈해서 실패시 종료
126     fp = fopen("data03.txt", "r");
127
128     if (fp == NULL)
129     {
130         printf("파일 오픈 실패\n");
131         return;
132     }
133
134     // 파일 끝까지 반복한다
135     while (!feof(fp))
136     {
137         // 값을 읽고 cnt 값을 증가시킨다
138         fscanf(fp, "%d", &temp);
139         cnt++;
140     }
141
142     // 파일 포인터를 맨 앞으로 보낸다
143     rewind(fp);
144
145     // list를 cnt 개수 만큼 동적할당을 한다
146     list = (int *)malloc(sizeof(int) * cnt);
147
148     // cnt 값 초기화
149     cnt = 0;

```

```

144 // 파일 끝까지 반복한다
145 while (!feof(fp))
146 {
147     // 파일 값을 읽고 list 동적할당한 곳에 저장을 한다
148     fscanf(fp, "%d", &temp);
149     list[cnt] = temp;
150     cnt++;
151 }
152
153 // 필요한 값들을 출력한다
154 printf("< 정렬되지 않은 리스트 > \n");
155 for (int i = 0; i < cnt; i++)
156     printf("%d > ", list[i]);
157 printf("\n\n");
158
159 printf("< 정렬 과정 > \n");
160 // 퀵 정렬 호출
161 quick_sort(list, 0, cnt-1);
162
163 printf("< 정렬된 리스트 > \n");
164 for (int i = 0; i < cnt; i++)
165     printf("%d > ", list[i]);
166 printf("\n\n");
167
168 // 동적할당을 해제하고 파일을 닫고 종료한다
169 free(list);
170 fclose(fp);
171 return 0;
172 }

```

3.3 소스 코드 분석

```
// 필요한 헤더파일 선언
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// 오류 방지 구문 제작
#pragma warning(disable : 4996)

// 스왑을 정의한다
#define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) )

// 필요한 배열, 변수를 선언한다
int tmp[100];
int n = 0;
```

1. 필요한 헤더파일을 선언한다
2. 오류 방지 구문 선언을 한다
3. 스왑 매크로를 정의한다
4. 필요한 변수와 배열을 선언한다. 형식에 맞게 출력하기 위함이다

```
// 파티션 함수 선언
int partition(int list[], int left, int right)
{
    // 피벗, low, high 변수 생성
    int pivot, temp;
    int low, high;
    int check = 0;

    // 왼쪽은 low, 오른쪽 + 1 값은 high로 값을 정한다
    low = left;
    high = right + 1;
    // 피벗값은 list의 left번째 값으로 정해준다
    pivot = list[left];
    // tmp 배열에 pivot 값을 저장한다
    tmp[n] = pivot;
    n++;
    // 값을 출력
    printf("- Pivot : %d -\n", pivot);
```

5. 피벗, low, high 등 변수를 선언한다
6. low는 left값 high는 right+1 값을 정의한다.
7. 피벗은 리스트 배열의 left번째 값을 정해준다

8. n번째 tmp 배열에 pivot 값을 대입하고 n을 하나 늘려준다

```
// 일단 한번 실행
do {
    // 일단 한번 실행
    do
    {
        low++; // 우선 low 값을 1개 올려준다
    }
    while (list[low] < pivot); // list의 low 번째 값이 pivot 값보다 크면 빠져나온다

    do // 일단 한번 실행
    {
        // 일단 한번 실행
        high--; // 우선 high 값을 1개 내려준다
    }
    while (list[high] > pivot); // list의 high 번째 값이 pivot 값보다 작으면 빠져나온다
}
```

9. do_while 구문을 실행한다

10. do_while 구문을 통해 우선 low 값을 하나 올려주면서 list의 low번째 값이 pivot의 값보다 크면 종료한다.

11. 마찬가지로 do_while 구문을 통해 우선 high 값을 하나 내려주면서 list의 high 번째 값이 pivot의 값보다 작으면 종료한다.


```

// 만약 tmp 배열에 pivot 값이 있을 경우 체크 표시를 한다
for (int i = 0; i < 101; i++)
{
    if (tmp[i] == list[low])
        check = 1;
}

// 체크 표시가 있을 경우 over 출력하고 값 출력
if(check == 1)
    printf("low: over high: %d [ ", list[high]);
// 아닐 경우는 그냥 출력
else
    printf("low: %d high: %d [ ", list[low], list[high]);

// left부터 right+1 까지 반복하면서 리스트 값 출력
for (int i = left; i < right+1; i++)
{
    printf("%d > ", list[i]);
}

printf("\n\n");

// 만약 high 가 low 보다 크다면 값 교체
if (low < high)
    SWAP(list[low], list[high], temp);

} while (low < high); // low < high가 아닐때 까지 반복

```

12. 만약 tmp배열에 pivot 값이 있을 경우는 체크를 해준다
13. 체크 표시가 있으면 over 출력 아닐 경우는 그냥 출력을 해준다
14. left부터 right+1까지 반복하면서 리스트의 값들을 출력한다.
15. 만약 high가 low 보다 크다면 스왑해준다.
16. 맨 위 do_while 구문 조건은 low < high 가 아닐 때 까지 반복을 해준다

```

// 스왑 해준다
SWAP(list[left], list[high], temp);

// high값을 반환한다
return high;

```

17. 반복이 끝나면 스왑 매크로를 호출한다
18. high값을 반환한다

```

// 퀵 정렬 함수
void quick_sort(int list[], int left, int right)
{
    // 만약 left가 right 보다 작다면
    if (left < right)
    {
        // q 값을 파티션을 통해 받는다
        int q = partition(list, left, right);
        // 재귀 함수를 통해 다시 호출한다
        quick_sort(list, left, q - 1);
        quick_sort(list, q + 1, right);
    }
}

```

19. 퀵 정렬 함수를 선언한다
20. 만약 left보다 right 보다 작다면 실행한다
21. Q 값을 파티션에서 반환 된 값을 저장한다
22. 재귀 함수를 통해 Q를 보내면서 다시 호출한다

```

FILE *fp; // 파일 포인터 선언
int *list; // int형 list 포인터 변수 선언

// 필요한 변수 선언
int temp;
int cnt = 0;

// 파일 오픈해서 실패시 종료
fp = fopen("data03.txt", "r");

if (fp == NULL)
{
    printf("파일 오픈 실패\n");
    return;
}

```

23. 파일 포인터를 선언한다
24. int형 포인터 변수를 선언한다
25. 필요한 변수를 선언한다
26. 파일을 오픈해서 실패시 종료한다

```

// 파일 끝까지 반복한다
while (!feof(fp))
{
    // 값을 읽고 cnt 값을 증가시킨다
    fscanf(fp, "%d", &temp);
    cnt++;
}

// 파일 포인터를 맨 앞으로 보낸다
rewind(fp);

// list를 cnt 개수 만큼 동적할당을 한다
list = (int *)malloc(sizeof(int) * cnt);

// cnt 값 초기화
cnt = 0;

```

27. 파일 끝까지 반복을 한다

28. 값을 읽으면서 카운트를 한다

29. 파일 포인터를 맨 앞으로 보낸다

30. 포인터 선언한 list를 카운트한 값 만큼 동적할당을 한다

31. 카운트를 초기화한다.

```

// 파일 끝까지 반복한다
while (!feof(fp))
{
    // 파일 값을 읽고 list 동적할당한 곳에 저장한다
    fscanf(fp, "%d", &temp);
    list[cnt] = temp;
    cnt++;
}

// 필요한 값들을 출력한다
printf("< 정렬되지 않은 리스트 > \n");
for (int i = 0; i < cnt; i++)
    printf("%d > ", list[i]);
printf("\n\n");

printf("< 정렬 과정 > \n");
// 퀵 정렬 호출
quick_sort(list, 0, cnt-1);

printf("< 정렬된 리스트 > \n");
for (int i = 0; i < cnt; i++)
    printf("%d > ", list[i]);
printf("\n\n");

```

32. 파일 끝까지 반복을 한다.

33. 파일 값을 읽고 list 동적할당 한 곳에 저장을 한다

34. 필요한 값들을 출력하고 중간에 쉼 정렬을 호출한다

35. 정렬된 리스트를 프린트한다

```
// 동적할당을 해제하고 파일을 닫고 종료한다
free(list);
fclose(fp);
return 0;
```

36. 동적할당을 해제하고 파일을 닫고 종료한다

3.4 실행 결과

```
Microsoft Visual Studio 디버그 콘솔
< 정렬되지 않은 리스트 >
15 > 4 > 8 > 22 > 7 > 54 > 3 > 80 > 48 > 42 > 2 > 98 > 5 >

< 정렬 과정 >
- Pivot : 15 -
low: 22 high: 5 [ 15 > 4 > 8 > 22 > 7 > 54 > 3 > 80 > 48 > 42 > 2 > 98 > 5 > ]
low: 54 high: 2 [ 15 > 4 > 8 > 5 > 7 > 54 > 3 > 80 > 48 > 42 > 2 > 98 > 22 > ]
low: 80 high: 3 [ 15 > 4 > 8 > 5 > 7 > 2 > 3 > 80 > 48 > 42 > 54 > 98 > 22 > ]

- Pivot : 3 -
low: 4 high: 2 [ 3 > 4 > 8 > 5 > 7 > 2 > ]
low: 8 high: 2 [ 3 > 2 > 8 > 5 > 7 > 4 > ]

- Pivot : 8 -
low: over high: 4 [ 8 > 5 > 7 > 4 > ]

- Pivot : 4 -
low: 5 high: 4 [ 4 > 5 > 7 > ]

- Pivot : 5 -
low: 7 high: 5 [ 5 > 7 > ]

- Pivot : 80 -
low: 98 high: 22 [ 80 > 48 > 42 > 54 > 98 > 22 > ]
low: 98 high: 22 [ 80 > 48 > 42 > 54 > 22 > 98 > ]

- Pivot : 22 -
low: 48 high: 22 [ 22 > 48 > 42 > 54 > ]

- Pivot : 48 -
low: 54 high: 42 [ 48 > 42 > 54 > ]

< 정렬된 리스트 >
2 > 3 > 4 > 5 > 7 > 8 > 15 > 22 > 42 > 48 > 54 > 80 > 98 >

C:\Users\korca\Desktop\20204005_김필중_자료구조13주차 소스코드\Debug\20204005_김필중_2
프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.
```

3.5 느낀점

퀵 정렬은 정렬 중 이름답게 가장 빠른 정렬이다. 그만큼 코드도 다른 코드에 비해 조금 복잡한 편이었다. 하지만 속도에서는 가장 빠르기 때문에 잘 사용할 수 있을 것 같다. 또한 퀵정렬은 알고리즘으로 구현이 가능하지만 함수도 있기 때문에 많은 곳에서 사용이 가능하다. 단 정렬되어있는 리스트라면 그만큼 수행시간이 오래 걸린다. 많은 프로그램들을 정렬을 한다고 생각했을 때 처음 정렬은 퀵으로 빠르게 정렬을 하고 그 이후는 다른 방법으로 정렬을 하면 되겠다고 생각했다. 또한 시간 복잡도가 작기 때문에 그 이점을 계속해서 활용하면 더 빠르고 가벼운 프로그램을 제작할 수 있겠다고 생각했다.