

# 고급응용C프로그래밍 8차 과제 보고서

20191245 노유정

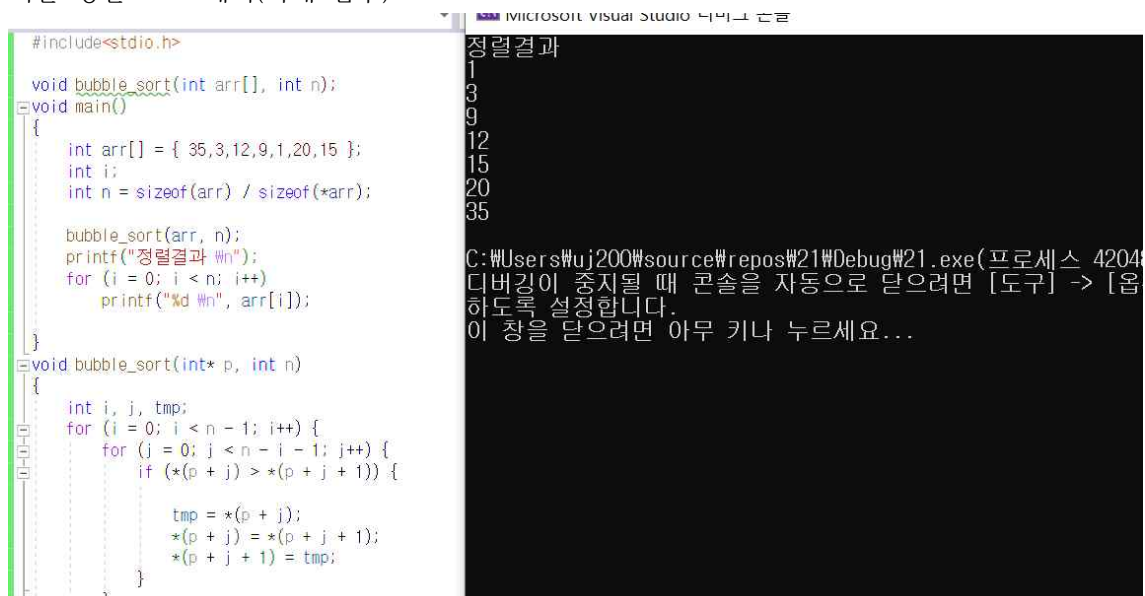
## \* 정렬 알고리즘이란?

우선 정렬이란, 데이터를 원하는 순서로 나열하는 것을 뜻한다. 그렇다면 정렬 알고리즘이 필요한 이유는 무엇일까? 바로 이진탐색을 하기 위함이다. 이진탐색은 선형탐색보다 훨씬 시간이 절약된다는 장점이 있는데, 이때 선형탐색은 처음부터 마지막까지 순서대로 쭉 탐색하는 방법이고, 이진탐색은 '정렬된 배열'에 대해 검색 단계별로 검색의 범위를 반으로 줄여나가면서 탐색한다. 따라서 이진탐색이 훨씬 효율적이다. 다만 이진탐색은 정렬된 배열을 사용하는 것이 필수이기 때문에 정렬을 하는 알고리즘부터 필요한 것이다. 정렬 알고리즘에는 버블정렬, 퀵정렬, 선택정렬, 병합정렬 등이 있다.

## \* 버블 정렬

우선 버블 정렬이란 오름차순 정렬 알고리즘으로 배열의 첫 번째부터 마지막까지 '인접한 두 값을 비교하여 오른쪽에 큰 수, 왼쪽에 작은 수를 세팅하는 형태로 진행된다. 이 한 번의 과정을 스캔이라고 하며 스캔을 '모든 숫자가 정렬될 때'까지 수행하는 것을 버블 알고리즘이라고 한다. 버블 정렬은 인접 값만 비교하여 구현이 쉽다는 장점이 있다.

버블 정렬 코드 예시(아래 첨부)



```
#include<stdio.h>

void bubble_sort(int arr[], int n);
void main()
{
    int arr[] = { 35,3,12,9,1,20,15 };
    int i;
    int n = sizeof(arr) / sizeof(*arr);

    bubble_sort(arr, n);
    printf("정렬결과\n");
    for (i = 0; i < n; i++)
        printf("%d\n", arr[i]);
}

void bubble_sort(int* p, int n)
{
    int i, j, tmp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (*(p + j) > *(p + j + 1)) {

                tmp = *(p + j);
                *(p + j) = *(p + j + 1);
                *(p + j + 1) = tmp;
            }
        }
    }
}
```

정렬결과  
1  
3  
9  
12  
15  
20  
35

C:\Users\Wuj200\source\repos\21\Debug\21.exe (프로세스 4204)  
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [오퍼레이팅 시스템] -> [콘솔] -> [콘솔을 자동으로 닫으려면]을 선택하십시오.  
이 창을 닫으려면 아무 키나 누르세요...

위 코드의 bubble\_sort 함수는 데이터의 길이 n과 배열을 포인터로 받은 매개변수를 이용한다. 이때 sizeof(\*arr) 는 맨 처음 원소의 길이를 말한다. for문에서 i는 반복 횟수를 지정하기 위하여 사용하고, j가 정렬을 위해 사용된다. 배열의 j번째 인덱스를 가진 원소가 그 다음 인덱스의 원소보다 크다면( = if(\*(p+j)>\*(p+j+1))문에 해당하면) 포인터를 이용한 swap을 진행하여 큰값이 오른쪽에 오도록 위치시켜 버블 알고리즘을 구현한다.

#### \* 퀵 정렬

퀵 정렬은 임의의 원소를 피벗(기준)으로 삼는다. 그리고 첫 번째 원소에(피벗이 첫 번째라면 2번째 원소에) 왼쪽 포인터, 오른쪽 포인터를 위치시킨다. 그 다음 왼쪽 포인터는 기준보다 큰 값 오른쪽은 기준보다 작은 값이 올 때까지 인덱스를 증가시킨다. 그리고 왼쪽, 오른쪽을 서로 바꾼다(Swap). 그 다음 두 포인터 모두 오른쪽으로 한 칸씩 이동한다. 이를 오른쪽 포인터가 피벗에 도달할 때까지 반복한다. 왼쪽 포인터와 피벗을 변경한다(즉 피벗의 배열 내 자리가 바뀐다). 피벗 기준 왼, 오른 나누어 퀵 정렬을 반복해서 수행하고, 이것이 완료되면 배열을 합친다. 퀵정렬은 이름대로 빠르며 효율이 좋다. 하지만 만약 이미 데이터가 내림차순 혹은 오름차순 정렬이 되어있다면 매우 비효율적인 알고리즘이 된다는 특징이 있다.

#### \* 선택 정렬

선택 정렬은 제일 왼쪽에 있는 원소를 기준으로 잡는다. 그리고 차례로 오른쪽에 있는 모든 원소와 비교하여 자신보다 작은 값을 모두 기억해둔 후 해당 값들 중에 가장 작은 값과 자신의 자리를 바꾼다. 그러면 첫 번째 원소, 제일 작은 원소가 확정된다. 이 과정을 다음 인덱스를 기준으로 하여 다시 진행한다. 단순한 알고리즘이라 쉽게 구현 가능하나, 불안정하고 비효율적이라는 단점이 있다.

소스의 예시는 다음과 같다.(코드는 아래에 다시 첨부)

The screenshot shows the Microsoft Visual Studio IDE. On the left, the C++ source code for Selection Sort is displayed. The code includes `<stdio.h>` and defines a `SelectionSort` function that takes an integer array `p` and sorts it in ascending order. The `main` function initializes an array `intNum` with values {3, 2, 1, 5, 4} and calls `SelectionSort` on it. On the right, the 'Debug Console' window shows the output of the program, which is the sorted array: 1, 2, 3, 4, 5. Below the output, there is a message from the debugger: 'C:\Users\Wuj200\source\repos\21\Debug\21.exe(프로세스 42540개)이(가) 종료되었습니다. 디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] 하도록 설정합니다. 이 창을 닫으려면 아무 키나 누르세요...'.

```
#include <stdio.h>

void SelectionSort(int* p)
{
    int i, j, temp;
    for (i = 0; i < 5 - 1; i++) {
        for (j = i + 1; j < 5; j++) {
            if (*(p + i) > *(p + j)) {
                temp = *(p + i);
                *(p + i) = *(p + j);
                *(p + j) = temp;
            }
        }
    }
}

void main(void)
{
    int intNum[5] = { 3, 2, 1, 5, 4 };
    int i;
    SelectionSort(intNum);

    for (i = 0; i < 5; i++) {
        printf("%d\n", intNum[i]);
    }
}
```

```
1
2
3
4
5
C:\Users\Wuj200\source\repos\21\Debug\21.exe(프로세스 42540개)이(가) 종료
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅]
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

위 코드의 Selectionsort 함수는 주어진 intNum 배열을 포인터로 받아서 \*(p+i), 즉 i번째 인덱스의 원소에 대하여 그 원소의 오른쪽에 있는(j = i + 1부터) 모든 원소와 비교한다. if문에 걸리면, 즉 j번째 원소가 i번째 원소보다 작다면 포인터를 이용한 swap 과정을 첫 번째 원소부터 마지막에서 2번째 원소까지(i < 5 - 1) 진행함으로써 선택 정렬 알고리즘을 구현한다.

#### \*병합정렬

병합정렬은 정렬할 데이터를 나누어서 정렬하고 다시 합치는 정렬 알고리즘이다. 보통 데이터를 2개씩 쪼개며, 2개는 한 번에 비교 가능하므로 2개짜리 정렬된 데이터 set이 여러 개 나오게 된다. 이것을 병합하는 과정이 중요하기에 병합 정렬이라고 한다. 병합은 두 2개짜리 원소를 합칠 때 왼쪽에 있는 값 2개, 오른쪽에 있는 값 2개를 비교한 후 왼쪽 중 더 큰값, 오른쪽 중 더 작은 값 2개를 비교하여 위치시킨다. 총 연산의 횟수는  $N \cdot \log_2 N$  이고, N은 데이터의 원소 개수를 뜻한다.

#### \*선택 정렬 코드

```
#include <stdio.h>
```

```
void SelectionSort(int* p)
{
    int i, j, temp;
    for (i = 0; i < 5 - 1; i++) {
        for (j = i + 1; j < 5; j++) {
            if (*(p + i) > *(p + j)) {
                temp = *(p + i);
                *(p + i) = *(p + j);
                *(p + j) = temp;
            }
        }
    }
}
```

```
void main(void)
{
    int intNum[5] = { 3, 2, 1, 5, 4 };
    int i;

    for (i = 0; i < 5; i++) {
        printf("%d\n", intNum[i]);
    }
}
```

#### 버블 정렬 코드

```
#include<stdio.h>
```

```

void bubble_sort(int arr[], int n);
void main()
{
    int arr[] = { 35,3,12,9,1,20,15 };
    int i;
    int n = sizeof(arr) / sizeof(*arr);

    bubble_sort(arr, n);
    printf("정렬결과 \n");
    for (i = 0; i < n; i++)
        printf("%d \n", arr[i]);

}
void bubble_sort(int* p, int n)
{
    int i, j, tmp;
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (*(p + j) > *(p + j + 1)) {

                tmp = *(p + j);
                *(p + j) = *(p + j + 1);
                *(p + j + 1) = tmp;

            }
        }
    }
}

```