



# 4주차 [과제]

이름 : 유재영

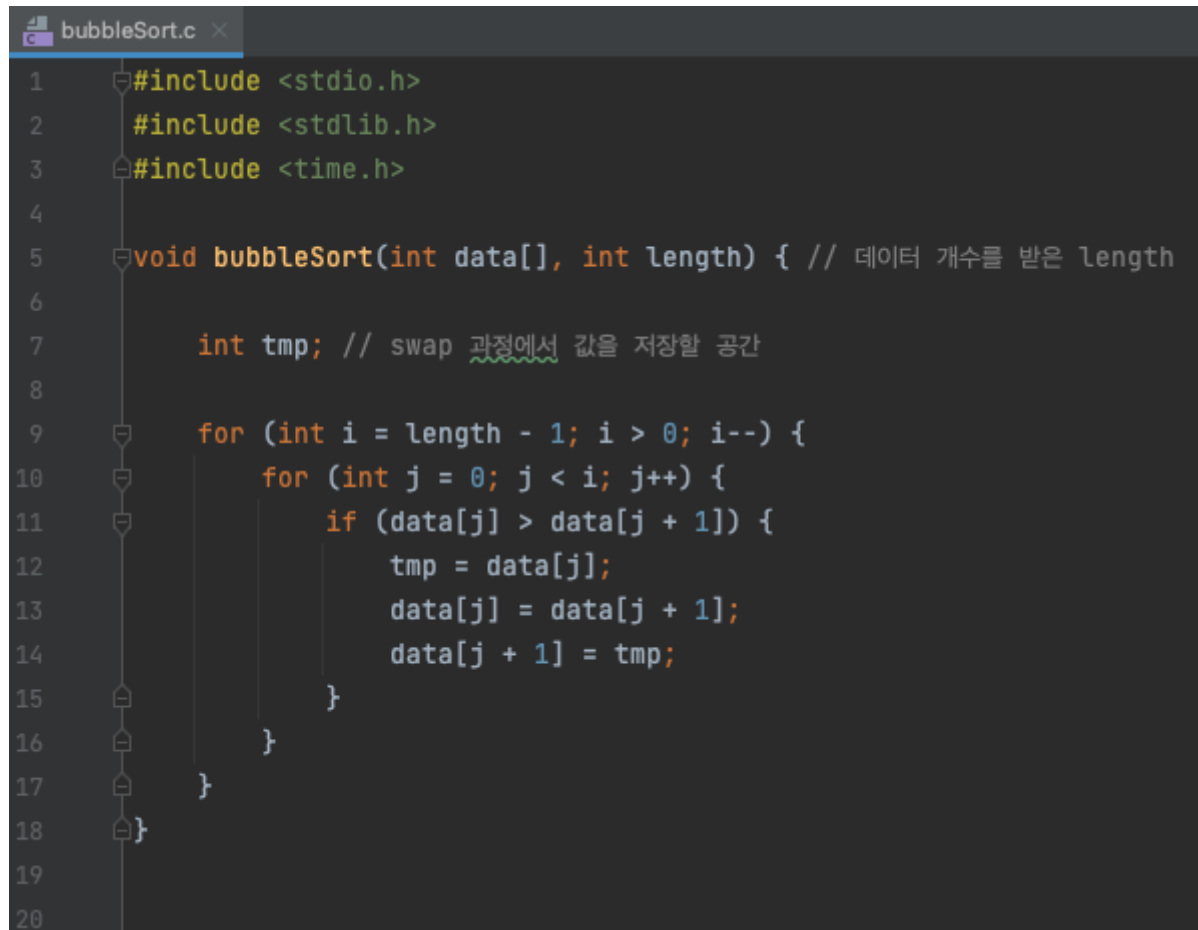
학번 : 20181650

제출일 : 2023.04.02

담당교수 : 최해철 교수님

## 과제 1, 2. 버블 정렬 고도화 및 수행시간 분석

먼저, 실습 간 작성한 버블 정렬 분석을 진행해보려고 한다.



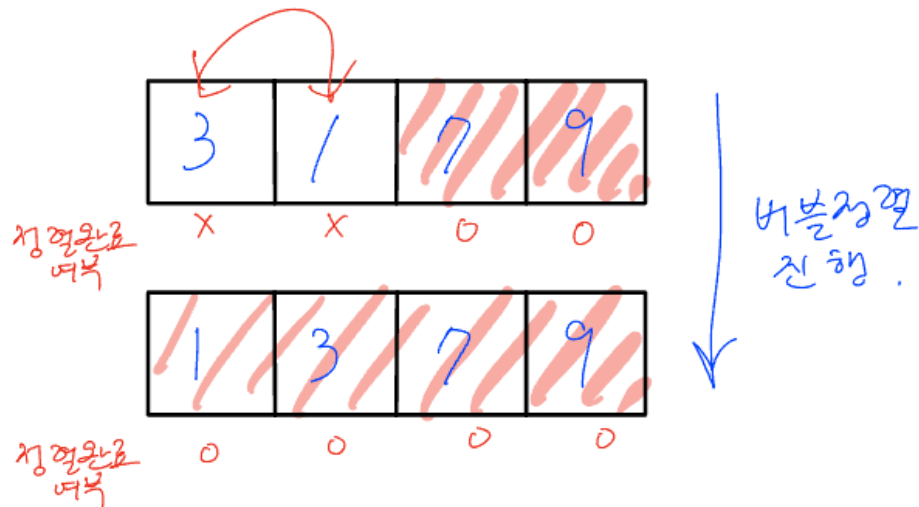
```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  void bubbleSort(int data[], int length) { // 데이터 개수를 받은 length
6
7      int tmp; // swap 과정에서 값을 저장할 공간
8
9      for (int i = length - 1; i > 0; i--) {
10         for (int j = 0; j < i; j++) {
11             if (data[j] > data[j + 1]) {
12                 tmp = data[j];
13                 data[j] = data[j + 1];
14                 data[j + 1] = tmp;
15             }
16         }
17     }
18 }
19
20
```

**9행** 부분에서 첫 for문이 시작된다.

해당 for문은 전체 데이터(배열)에 대해서 버블 정렬을 진행해야되기 때문에,

**5행**에서 매개변수로 받은 `int length` 즉, 데이터의 개수 만큼 for문을 진행하도록 작성되었다.

이때, for문의 초기화식에서 `int i = length - 1;` 로 선언한 이유를 그림으로 살펴보면,



정렬되지 않은 데이터가 두개가 남았을 때, 한 번의 버블 정렬이 진행되면 모두 정렬된 상태가 되기 때문이다.

따라서 버블 정렬 실행 횟수는 `int i = length - 1;` 번이다.

**9행의 for문은, 감소 연산을 사용하여 for문을 돌고 있는데,**

버블 정렬의 진행 과정 설계를 **배열의 원소를 한 바퀴 돌면서 각각 버블 정렬 후 마지막 자리에 큰 값을 고정 시키며,**

고정된 자리를 제외하고 다시 버블 정렬 반복하도록 설계하였기 때문이다.

**만약 앞부터 고정을 진행하도록 설계하는 경우,** 해당 부분의 연산이 **증가 연산**으로 진행될 것이다.

**10행의 for문은, 배열 원소의 값을 비교 진행하는 과정이다.**

해당 알고리즘의 설계는,

**배열의 왼쪽(인덱스 번호 0번)부터 오른쪽 방향으로 오름차순 정렬하도록 작성되었다.**

11행을 시작으로, **비교 연산자 >** 로 피연산자인 좌항과 우항을 대상으로,

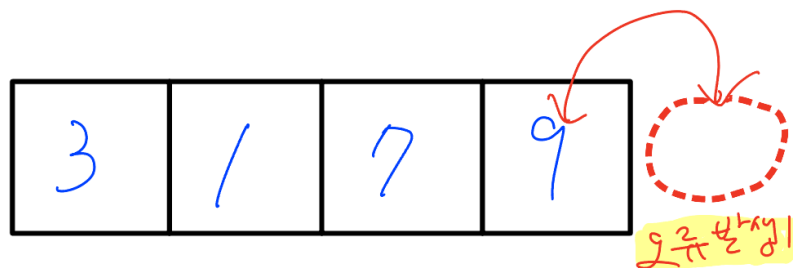
**좌항의 값이 우항의 값보다 큰 경우 서로 자리를 교환하도록 설계되었다. (버블 정렬 방식)**

### 버블 정렬이 진행되는 과정은,

1. **7행**에서 선언된 데이터를 담을 공간 `tmp`에 교환되는 우항의 값을 넣고
2. 좌항의 값을 우항에 넣은 뒤,
3. `tmp`에 저장된 값을 좌항에 다시 넣어주는 방식으로 교환이 진행되고 있다.

여기서, **10행**의 조건식이 `j < i` 인 이유는 버블 정렬이 일어나는 과정에 이유가 있다.

1. **정렬되는 대상이 배열**이므로 시작 **인덱스 번호가 0**이며,
2. 버블 정렬 과정이 배열의 끝을 넘어서 진행하면 메모리를 초과해서 사용하면 오류가 발생하기 때문이다.



따라서, 들어온 데이터 개수 `length - 1` 까지 **10행의 조건식**이 되어야 한다.

### **고도화 진행 → 임의의 loop에서 남은 원소들이 모두 정렬되어 있다면?**

위에서 작성한 버블 정렬 코드는,

남은 원소들이 모두 정렬되어있는 경우에도 버블 정렬 가능 여부 체크하는 과정을 계속 진행하게 된다.

즉, **불필요하게 10행 이후 부분이 진행될 수 있다는 것이다.**

해당 부분이 불필요하게 진행되지 않기 위해서는 구분할 수 있게 하기 위한 추가 조건 설계가 필요해 보인다.

이미 정렬이 완료되었으니 진행하지 말라고 알릴 조건, 방지턱과 같은 역할로 `swapCount`를 추가해보려 한다.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  void bubbleSort(int data[], int length) { // 데이터 개수를 받은 length
6
7      int tmp; // swap 과정에서 값을 저장할 공간
8
9      int swapCount = -1;
10
11     for (int i = length - 1; i > 0; i--) {
12
13         printf("%d\n", swapCount);
14
15         if (!swapCount) {
16             break;
17         }
18         swapCount = 0;
19         for (int j = 0; j < i; j++) {
20             if (data[j] > data[j + 1]) {
21                 tmp = data[j];
22                 data[j] = data[j + 1];
23                 data[j + 1] = tmp;
24                 swapCount++;
25             }
26         }
27     }
28 }

```

먼저 **9행**에 `int swapCount = -1;` 을 선언해 주었다.

**-1**로 초기화한 이유는, 초기화를 진행하지 않으면 쓰레기 값이 들어가기 때문이다.

**13행**의 `printf("%d\n", swapCount);` 는 정상 작동 여부를 체크를 위해 넣어 주었다.

**15행**의 if문에서, 버블 정렬 실행에 따라 for문 탈출을 결정 하도록 `if(!swapCount)` 라는 조건을 걸어두었다.

만약 실행되지 않았다면 16행의 `break;` 를 통해 for문을 탈출하게 되며, bubbleSort 메소드 실행이 종료된다.

18행의 `swapCount = 0;` 은 아래 버블 정렬 과정에서 증가된 swapCount를 초기화 하는 역할이다.

해당 부분의 초기화로 인해, 만약 이후 버블 정렬 과정이 실행되지 않아 swapCount의 값에 변화가 없으면,

15행의 if문으로 하여금 bubbleSort 메소드의 실행이 종료되는 것이다.

20행의 버블 정렬을 진행하는 if문이 실행되는 경우, 24행의 증가 연산 `swapCount++` 를 통해 정렬이 실행되었음을 알리도록 진행하였다.

해당 방법으로써, 남은 원소가 정렬된 경우 무의미한 19행 for문이 실행되는 것을 방지하였다.

## 수행 시간 분석

해당 버블 정렬의 수행시간을 분석해 보면,

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  void bubbleSort(int data[], int length) { // 데이터 개수를 받은 length
6
7      int tmp; // swap 과정에서 값을 저장할 공간
8
9      int swapCount = -1;
10
11     for (int i = length - 1; i > 0; i--) {
12
13         printf("%d\n", swapCount);
14
15         if (!swapCount) {
16             break;
17         }
18         swapCount = 0;
19         for (int j = 0; j < i; j++) {
20             if (data[j] > data[j + 1]) {
21                 tmp = data[j];
22                 data[j] = data[j + 1];
23                 data[j + 1] = tmp;
24                 swapCount++;
25             }
26         }
27     }
28 }
```

Handwritten annotations on the code:

- A circle around the outer `for` loop (line 11) with a handwritten  $n$  next to it.
- A bracket on the right side of the inner `if` statement (lines 15-17) with the handwritten text "2번이든 C".
- A circle around the inner `for` loop (line 19) with a handwritten  $n$  next to it.
- A bracket on the right side of the inner `if` statement (lines 20-24) with the handwritten text "2번이든 C".
- At the bottom, the complexity formula:  $= n^2 + C \rightarrow O(n^2)$ .

고도화가 진행된 부분은 오버헤드에 해당 하므로, 기존 버블 정렬 수행시간에서의 변화는 없다.

$$T(n) = O(n^2)$$

최종적으로, 고도화된 버블 정렬 코드는 아래와 같다.

```
void bubbleSort(int data[], int length) { // 데이터 개수를 받은 length
    int tmp; // swap 과정에서 값을 저장할 공간
    int swapCount = -1;
    for (int i = length - 1; i > 0; i--) {

        if (!swapCount) {
            break;
        }
        swapCount = 0;
        for (int j = 0; j < i; j++) {
            if (data[j] > data[j + 1]) {
                tmp = data[j];
                data[j] = data[j + 1];
                data[j + 1] = tmp;
                swapCount++;
            }
        }
    }
}
```

고도화 진행 부분을 테스트하기 위해 임의로 작성한 일부 정렬된 데이터를 넣어 진행해보면,

```
printf("정렬 전 배열 : ");
for (int i = 0; i < n; i++) {
    printf("%d ", testData[i]);
}

bubbleSort( data: testData, length: 6);

printf("\n정렬 후 배열 : ");
for (int i = 0; i < n; i++) {
    printf("%d ", testData[i]);
}
```

```
int main() {

    int testData[6] = {1, 2, 3, 4, 6, 5};

    printf("정렬 전 배열 : ");
    for (int i = 0; i < n; i++) {
        printf("%d ", testData[i]);
    }

    bubbleSort(testData, 6);

    printf("\n정렬 후 배열 : ");
    for (int i = 0; i < n; i++) {
```

```

/Users/yujaeyeong/Develo
정렬 전 배열 : 1 2 3 4 6 5
정렬 후 배열 : -1
1
0
1 2 3 4 5 6

```

```

        printf("%d ", testData[i]);
    }
}

```

버블 정렬이 한 번 진행되어서 1이라는 값이 출력된 것을 볼수 있고,

이후 값이 0으로 초기화된 후, 버블 정렬이 진행되지 않아 0으로 값이 유지되어

위 과제 1에서 설명한, 15행의 조건에 따라 bubbleSort 메소드를 종료하게 된다.

## 최종 코드 및 실행 결과

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  void bubbleSort(int data[], int length) { // 데이터 개수를 받은 length
6      int tmp; // swap 과정에서 값을 저장할 공간
7      int swapCount = -1;
8      for (int i = length - 1; i > 0; i--) {
9
10         if (!swapCount) {
11             break;
12         }
13         swapCount = 0;
14         for (int j = 0; j < i; j++) {
15             if (data[j] > data[j + 1]) {
16                 tmp = data[j];
17                 data[j] = data[j + 1];
18                 data[j + 1] = tmp;
19                 swapCount++;
20             }
21         }
22     }
23 }

```



```

29 ▶ int main() {
30
31     int n;
32     printf("랜덤 수의 개수를 입력하시오. : ");
33
34     scanf("%d",&n);
35
36     int *data = (int *) malloc( size: sizeof(int) * n);
37
38     srand(time(NULL));
39
40     for (int i = 0; i < n; i++) {
41         data[i] = rand() % 100;
42     }
43
44     printf("정렬 전 배열 : ");
45
46     for (int i = 0; i < n; i++) {
47         printf("%d ", data[i]);
48     }
49
50     printf("\n정렬 후 배열 : ");
51
52     bubbleSort(data, length: n);
53
54     for (int i = 0; i < n; i++) {
55         printf("%d ", data[i]);
56     }
57
58     free(data);
59 }

```

```

/Users/yujaeyeong/Developments/algorithm/
랜덤 수의 개수를 입력하시오. : 10
정렬 전 배열 : 31 16 92 8 66 74 23 99 61 60
정렬 후 배열 : 8 16 23 31 60 61 66 74 92 99

```

## 과제 3. 삽입 정렬

삽입 정렬 코드를 다음과 같이 설계하였다.

```
5 void insertionSort(int data[], int length) {  
6     int i, j, key;  
7     for (i = 1; i < length; i++) {  
8         key = data[i];  
9         for (j = i - 1; j >= 0 && data[j] > key; j--) {  
10            data[j + 1] = data[j];  
11        }  
12        data[j + 1] = key;  
13    }  
14 }
```

각 행의 분석을 진행해보면

**7행** for문의 초기화식은 1부터 시작하도록 설정하였다.

이유는, 삽입정렬의 경우 비교를 위한 데이터를 하나씩 추가해 가면서 비교 후 정렬을 진행하는 것이기 때문이다.

조건식 `i < length;` 은 배열 데이터를 정렬하는 것이며, 배열 인덱스 번호가 0부터 시작하므로 정렬을 진행할 마지막 데이터의 인덱스 번호가 `length-1`에 해당하기 때문에 조건식을 위와 같이 작성한 것이다.

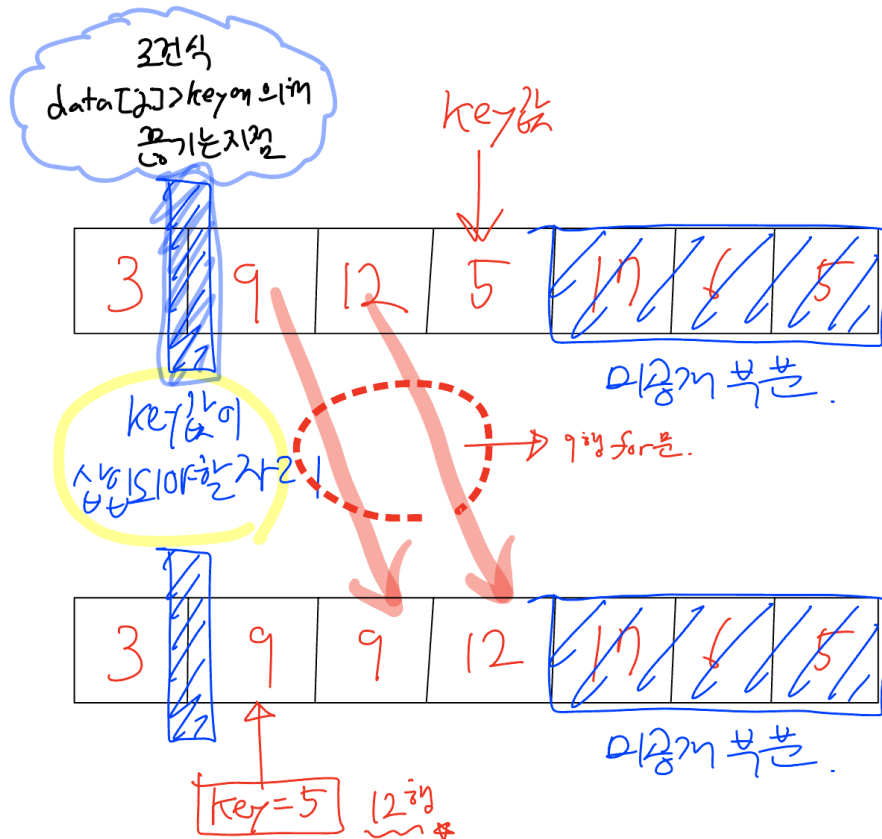
**8행**에서는 정렬 대상 데이터 값을 key에다 저장을 진행하였다.

**9행** for문의 경우 삽입되는 데이터가 알맞은 자리를 찾고, 삽입될 위치(뒤편)를 찾기 위해 데이터들을 한칸씩 확인하면서 오른쪽으로 옮기는 작업을 진행하기 위한 for문이다.

for문의 초기화식의 경우, for문 조건식에서 알맞은 자리를 찾아가는 과정에서 삽입될 자리를 찾기 위해서

`j = i - 1` 로 설정하였다.

조건식 부분 `j >= 0 && data[j] > key` 를 그림을 통해 좀더 자세히 살펴보면,



**9행** for문을 통해 값의 이동이 발생하고,

**12행**의 `data[j+1] = key;` 를 통해 올바른 위치에 삽입되게 된다.

위 방식을 통해 삽입 정렬이 진행되게 된다.

추가적으로 for문의 실행 구조를 좀 더 자세히 파악해보고자, for문 내부에 코드를 추가하였다.

```
void insertionSort(int data[], int length) {
    int i, j, key;
    for (i = 1; i < length; i++) {
        key = data[i];
        for (j = i - 1; j >= 0 && data[j] > key; j--) {
            data[j + 1] = data[j];
            printf("for문 탈출 전 j 값 : %d / ", j);
        }
        printf("for문 탈출 이후 j 값 : %d\n", j);
        data[j + 1] = key;
    }
}
```

위 코드의 실행 결과는 다음과 같다.

```
/Users/yujaeyeong/Developments/algorithm/univ-algorithm/230328/cmake-build-debug/230328
랜덤 수의 개수를 입력하시오. : 5
정렬 전 배열 : 71 96 59 88 68
for문 탈출 이후 j 값 : 0
for문 탈출 전 j 값 : 1 / for문 탈출 전 j 값 : 0 / for문 탈출 이후 j 값 : -1
for문 탈출 전 j 값 : 2 / for문 탈출 이후 j 값 : 1
for문 탈출 전 j 값 : 3 / for문 탈출 전 j 값 : 2 / for문 탈출 전 j 값 : 1 / for문 탈출 이후 j 값 : 0
59 68 71 88 96
```

해당 과정으로 9행의 for문의 증감식으로써 삽입될 자리가 결정되는 것과, 만약, 오른쪽 부터 오름 차순으로 정렬을 진행하는 경우 해당 부분 변경을 통해 진행할 수 있음을 예상할 수 있었다.

`data[j + 1] = key;` 처리로 인해 가장 작은 값의 경우 j + 1, 0번 자리에 들어가는 것을 확인할 수 있었다.

## 최종 코드 및 실행 결과

```
insertionSort.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  void insertionSort(int data[], int length) {
6      int i, j, key;
7      for (i = 1; i < length; i++) {
8          key = data[i];
9          for (j = i - 1; j >= 0 && data[j] > key; j--) {
10             data[j + 1] = data[j];
11         }
12         data[j + 1] = key;
13     }
14 }
```

```

16 ▶ int main() {
17
18     int n;
19     printf("랜덤 수의 개수를 입력하시오. : ");
20
21     scanf("%d",&n);
22
23     int *data = (int *) malloc( size: sizeof(int) * n);
24
25     srand(time(NULL));
26
27     for (int i = 0; i < n; i++) {
28         data[i] = rand() % 100;
29     }
30
31     printf("정렬 전 배열 : ");
32
33     for (int i = 0; i < n; i++) {
34         printf("%d ", data[i]);
35     }
36
37     printf("\n정렬 후 배열 : ");
38
39     insertionSort(data, length: n);
40
41     for (int i = 0; i < n; i++) {
42         printf("%d ", data[i]);
43     }
44
45     free(data);
46 }

```

```

/Users/yujaeyeong/Developments/algorithm/
랜덤 수의 개수를 입력하시오. : 10
정렬 전 배열 : 67 73 29 12 16 52 12 3 31 13
정렬 후 배열 : 3 12 12 13 16 29 31 52 67 73

```