알고리즘 숙제 #4 - 201901551 컴퓨터공학부 김정목

1. 문제

버블 정렬 알고리즘에서 각 패스에 대해 정렬을 수행할 때, 마지막으로 자리바꿈이 수행된 곳을 기억하면, 그 다음 패스부터는 마지막으로 자리를 바꾼 원소 이후로는 이미 원소들이 정렬되어 있으므로 더 이상 비교하지 않아도 된다. 이를 반영한 버블 정렬 알고리즘을 1) c언어로 구현하고, 2) 시간복잡도와 3) 작동 방법을 자세히 설명하시오.

2. 시간 복잡도

함수 bubbleSort(int arr[]) 안에서 두 개의 중첩된 반복을 사용하여 배열을 정렬한다. 첫 번째 반복문은 배열의 크기에 비례하여 반복하고, 두 번째 반복문은 현재 패스에서의 비교와 교환 작업을 수행한다. 만약 이미 정렬된 arr가 들어온다면 교환이 전혀 이루어지지 않고 한 번의 패스만으로 해결되므로 O(n)이라는 최선의 시간복잡도를 갖는다. 최악의 경우 배열이 역순으로 정렬되어 있다면 매 패스에서 모든 인접한 원소들을 비교하고 교환해야하므로 O(n^2)이라는 최악의 시간복잡도를 갖는다.

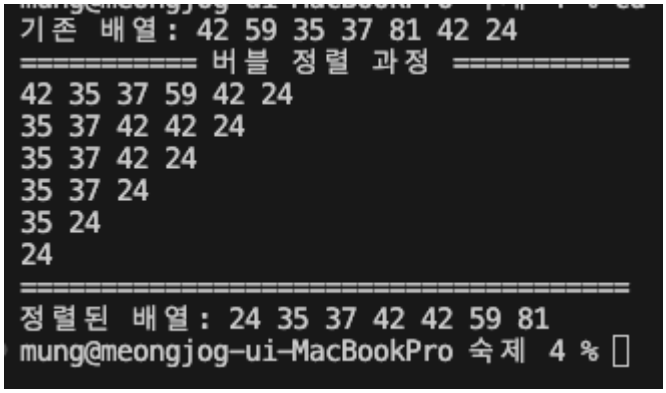
3. 작동 방법

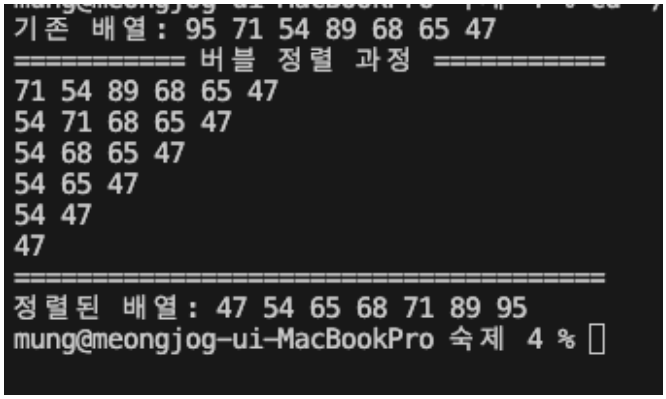
주요 함수	설명
void bubbleSort(int arr[])	배열 arr를 버블 정렬하여 정렬을 해 줄 함수

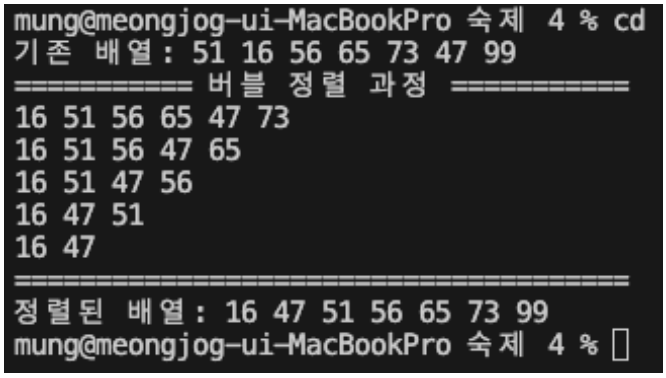
주요 변수	설명
int swapped	교환이 이루어졌는지 확인해주는 변수
int size	배열의 크기를 나타내는 변수

1. 랜덤 난수를 생성하여 배열 arr의 값을 무작위로 넣는다.
2. 배열 arr를 bubbleSort(int arr[]) 함수에 넣어 버블 정렬을 통해 정렬을 한다.
   - 배열의 첫 번재 원소부터 마지막 원소까지 반복하며 인접한 두 원소를 비교한다.
   - 현재 원소가 다음 원소보다 크면 두 원소의 자리를 바꾼다.
   - 한 번의 패스가 완료되면 가장 큰 원소가 배열의 마지막으로 이동하게 된다.
   - 다음 패스를 수행할 땐 전의 패스에서 이미 정렬된 부분은 수행하지 않아도 됨으로 수행하지 않는다.
   - 위 과정을 반복하면 최종적으로 배열이 정렬된다.
   - 매 패스마다 swapped 변수를 통해 교환이 이루어졌는지 확인하고, 만약 교환이 이루어지지 않았다면 이미 정렬된 상태이므로 반복문을 종료한다.
3. 기존 배열, 버블 정렬 과정, 정렬된 배열을 출력한다.

4. 출력 화면


기존 배열: 42 59 35 37 81 42 24
========== 버블 정렬 과정 ==========
42 35 37 59 42 24
35 37 42 42 24
35 37 42 24
35 37 24
35 24
24
==========
정렬된 배열: 24 35 37 42 42 59 81
mung@meongjog-ui-MacBookPro 숙제 4 %


기존 배열: 95 71 54 89 68 65 47
========== 버블 정렬 과정 ==========
71 54 89 68 65 47
54 71 68 65 47
54 68 65 47
54 65 47
54 47
47
==========
정렬된 배열: 47 54 65 68 71 89 95
mung@meongjog-ui-MacBookPro 숙제 4 %


mung@meongjog-ui-MacBookPro 숙제 4 % cd
기존 배열: 51 16 56 65 73 47 99
========== 버블 정렬 과정 ==========
16 51 56 65 47 73
16 51 56 47 65
16 51 47 56
16 47 51
16 47
==========
정렬된 배열: 16 47 51 56 65 73 99
mung@meongjog-ui-MacBookPro 숙제 4 %

1. 문제

선택 정렬은 입력 배열 전체에서 최솟값을 '선택'하여 배열의 0번 원소와 자리를 바꾸고, 다음에는 0번 원소를 제외한 나머지 원소에서 최솟값을 선택하여 배열의 1번 원소와 자리를 바꾼다. 이와 다르게 최댓값을 선택하여, 마지막 원소(배열의 (n-1)번 원소)와 자리를 바꾸고, 나머지 원소 중에서 최댓값을 선택하여 배열의 (n-2)번 원소와 자리를 바꾸는 방식으로 정렬을 할 수도 있다. 이러한 최댓값 선택 방식의 선택 정렬 알고리즘을 1) c언어로 구현하고, 2) 시간복잡도 와 3) 작동 방법을 자세히 설명하시오.

2. 시간 복잡도

함수 selectionSort(int arr[]) 안에서 두 개의 중첩된 반복을 사용하여 배열을 정렬한다. 첫 번째 반복문은 배열의 크기에 비례하여 반복하고, 두 번째 반복문은 현재 패스에서의 정렬 안 한 숫자 중 가장 큰 수를 찾아 교환 작업을 한다. 정렬이 된 배열이든 역순의 배열이든 최댓값을 찾는 과정에서 매번 모든 원소를 비교해야 하므로, 최선, 최악 모두 O(n^2) 시간 복잡도를 갖는다.

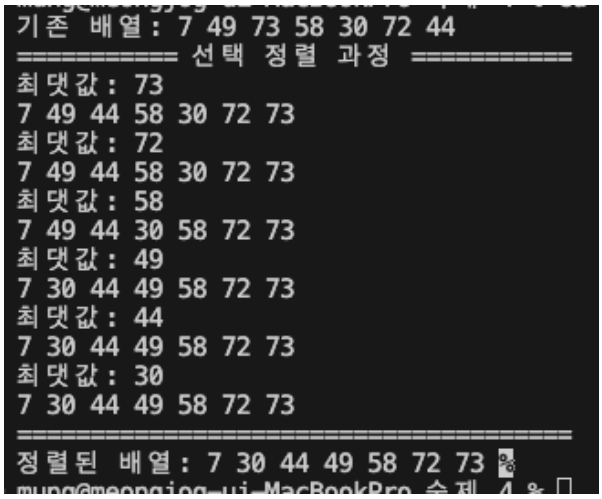
3. 작동 방법

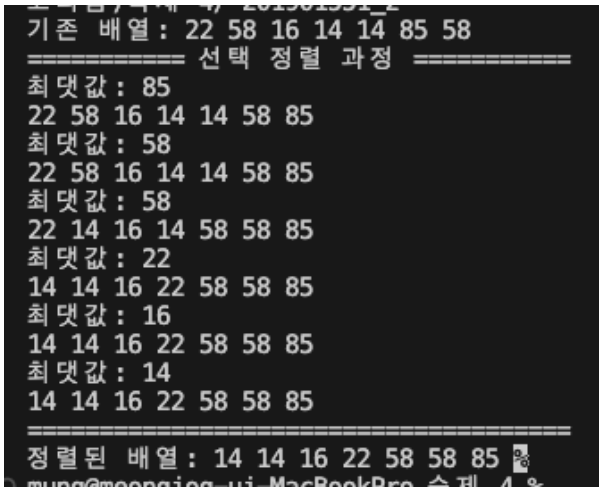
주요 함수	설명
void selectionSort(int arr[])	배열 arr를 선택 정렬하여 정렬을 해 줄 함수

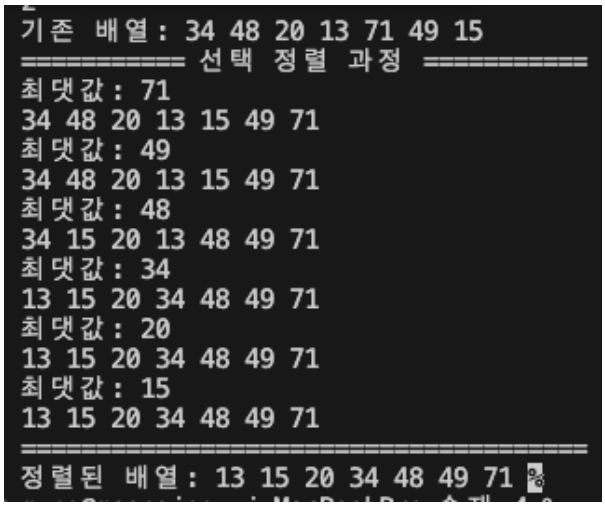
주요 변수	설명
int size	배열의 크기를 나타내는 변수

1. 랜덤 난수를 생성하여 배열 arr의 값을 무작위로 넣는다.
2. 배열 arr를 selectionSort(int arr[]) 함수에 넣어 선택 정렬을 통해 정렬을 한다.
   - 배열의 끝에서부터 시작하여 정렬되지 않은 부분 중에서 최댓값을 찾는다.
   - 최댓값을 현재 패스에서의 마지막 원소와 교환한다.
   - 위 과정을 반복하여 정렬되지 않은 부분에서 최댓값을 찾아 마지막 위치에 배치한다.
   - 정렬되지 않은 부분의 크기를 하나씩 감소시키면서 위 과정을 반복한다.
3. 기존 배열, 선택 정렬 과정, 현재 패스의 최댓값, 정렬된 배열을 출력한다.

4. 출력 화면


기존 배열: 7 49 73 58 30 72 44
========== 선택 정렬 과정 ==========
최댓값: 73
7 49 44 58 30 72 73
최댓값: 72
7 49 44 58 30 72 73
최댓값: 58
7 49 44 30 58 72 73
최댓값: 49
7 30 44 49 58 72 73
최댓값: 44
7 30 44 49 58 72 73
최댓값: 30
7 30 44 49 58 72 73
==========
정렬된 배열: 7 30 44 49 58 72 73


기존 배열: 22 58 16 14 14 85 58
========== 선택 정렬 과정 ==========
최댓값: 85
22 58 16 14 14 58 85
최댓값: 58
22 58 16 14 14 58 85
최댓값: 58
22 14 16 14 58 58 85
최댓값: 22
14 14 16 22 58 58 85
최댓값: 16
14 14 16 22 58 58 85
최댓값: 14
14 14 16 22 58 58 85
==========
정렬된 배열: 14 14 16 22 58 58 85


기존 배열: 34 48 20 13 71 49 15
========== 선택 정렬 과정 ==========
최댓값: 71
34 48 20 13 15 49 71
최댓값: 49
34 48 20 13 15 49 71
최댓값: 48
34 15 20 13 48 49 71
최댓값: 34
13 15 20 34 48 49 71
최댓값: 20
13 15 20 34 48 49 71
최댓값: 15
13 15 20 34 48 49 71
==========
정렬된 배열: 13 15 20 34 48 49 71