강의명: 자료구조

숙제 번호: 7

숙제 제목: Sort I (정렬 |)

학생 이름: 김성현

학번: 201910783

1. Selection sort(선택 정렬)

1.1

\*selection sort 알고리즘 기본 원리 쉽게 자세한 설명:

-selection sort, 즉 선택 정렬의 알고리즘은 정렬 알고리즘 중 가장 쉬운 편이다. 선택 정렬은 리스트, 즉 배열을 상대적인 왼쪽, 오른쪽으로 구분하여 볼 수 있다.

\*왼쪽 리스트(배열) 부분: 정렬되어진 data들

\*오른쪽 리스트(배열) 부분: 정렬 안되어져 해야하는 것으로 남아 있는 data들

-선택 정렬은 간단히 말해 매 step마다 오른쪽 리스트(부분)에서 순서 상관없이 가장 작은, 최소값을 “선택”하여 오른쪽 리스트의 첫째값과 바꾸는 것(SWAP)이다. 더 쉽고 간단하게 설명하자면, 매 step마다 오른쪽에서 가장 작은 값들을 “선택”하여 그걸 왼쪽에 차례대로 정렬시키는 것이다. 각 step마다 최소값을 “선택”하여 왼쪽에 차례대로 정렬한다면 하나의 step의 그 다음 step에서는 그 다음 최소값이 오게 되니 자동적으로 왼쪽에 오름차순으로 정렬되게 되는 것이다. 이렇게 step을 거듭해 나갈 때 마다 당연히 왼쪽에 정렬시키니 왼쪽 리스트(배열)1씩 계속 증가하고, 오른쪽 리스트(배열)은 선택해서 나가버리니 1씩 감소하여 최종 step이 끝나면 오른쪽 리스트(배열)이 빈 상태가 되는 것이다. 여기서 각 step마다 최소값을 항상 “선택”해서 SWAP시키고, 오른쪽 리스트(배열)이 빈 상태가 끝나는 상태이니 n개의 item, element가 있는 리스트(배열)에서는 n-1번의 step이 끝나면 선택 정렬이 끝나는 것이다.

쉽게 정리하자면 선택 정렬은 오른쪽 리스트에서 가장 작은 숫자를 택하여 왼쪽 리스트로 차차 쌓아가는 작업을 오른쪽 리스트가 공백이 될 때까지 반복하는 정렬 기법이다.

1.2

// ======================================================================

#include <stdio.h>

#include <stdlib.h>

#include "selection.h"

// ======================================================================

void selection\_sort(int list[], int n)

{

int i, j, least, temp;

for (i = 0; i < n - 1; i++) {

least = i;

for (j = i + 1; j < n; j++) {

if (list[j] < list[least])

least = j;

}

SWAP(list[i], list[least], temp);

print\_array(list, n);

}

}

// ======================================================================

\*이렇게 프로그램한 이유(with알고리즘 원리 + 각 index 확실한 설명):

-선택 정렬 프로그램에서 가장 핵심은 바로 두 개의 for문을 돌리는 두 개의 index들입니다. 1.1에서 설명한 알고리즘대로 설명하자면, 먼저 리스트의 item들을 탐색, 비교, 선택을 하기 위한 index에 쓸 변수 i와 j, 그리고 비교할 대상과 옮길 대상을 지정하기 위한 변수 least, 그리고 교환 SWAP을 위한 변수 temp선언하였습니다. 그 이후 본격적인 프로그램이 시작되는데, 먼저 각 step의 진행을 위한 큰 for문을 선언합니다. 여기서 i는 index로써 매 step에서의 오른쪽 리스트의 첫번째 index, 첫번째 순서인 data를 가리키는 index입니다. 이를 변수 least에 넣어 least도 오른쪽 리스트의 첫째로 초기화하여 그 뒤에 있는 item들이랑 비교할 준비를 하였습니다. 그 다음 오른쪽 첫째 리스트와 그 이후 리스트들의 비교를 위하여 for문을 하나 중첩되도록 열어 j를 I + 1로 주고 n-1까지 1씩 증가하도록 반복시키는데, 여기서 j를 I + 1로 주는 것은 오른쪽 첫째 index가 i이니 그 다음 index인 I + 1부터 비교시키기 위해서고, n이전까지, n-1까지 돌리는 것인 n-1이 가장 마지막 index이기 때문입니다. 그렇게 for문 중첩을 열고 오른쪽 첫째 리스트의 index를 넣은 least를 j를 증가시켜가면서 비교하던 도중 만약 첫째 리스트의 index인 least가 가리키는 값이 j의 index를 가지는 값보다 큰 경우는 오름차순으로써 더 작은 값이 앞으로 정렬되어야 하기 때문에 오른쪽 첫째가 저장되어 있던 least를 j의 index로 변경시켜 정렬이 안되어 있는 값을 지칭하도록 합니다.그 다음 왼쪽 리스트에 정렬되게 하기 위해 오른쪽 첫번째 리스트이자 왼쪽 리스트의 끝 리스트를 가리키고 있는 i와 least를 temp변수와 SWAP함수를 이용해 바꾸어 주면서 왼쪽 리스트에 정렬시켜주었습니다. 이렇게 해줌으로써 결과적으로 if문 조건에 부합하는, 즉 정렬 순서가 맞지 않을 때에는 오른쪽 리스트의 최소값이 오른쪽의 첫번째이자 왼쪽의 끝으로, 순서가 맞아서 if문 만족 안하고 그냥 넘어간다면 least=i=j니까 변화가 없으니 정렬이 끝나게 되는 겁니다.

1.3

s1910783@oak:hw07$ gcc test-selection.c selection.c -o test-selection

s1910783@oak:hw07$ ./test-selection

step 0: 5 3 8 1 2 7

step 1: 1 3 8 5 2 7

step 2: 1 2 8 5 3 7

step 3: 1 2 3 5 8 7

step 4: 1 2 3 5 8 7

step 5: 1 2 3 5 7 8

\*각 step 설명(슬라이드 12-6참고):

-강의 자료 슬라이드 12-6을 직접 가지고 와서 비교하면서 설명드리겠습니다.

키보드이(가) 표시된 사진

자동 생성된 설명

Step0: 초기 아직 정렬되지 않은 상태임으로 넣은 그대로 상태라는 것을 알 수 있습니다.

Step1: step0의 상태에서 왼쪽 리스트가 공백이고 오른쪽 리스트에 6개의 item이 다 있으니 위 배열의 첫 index인 5와 나머지 오른쪽 리스트들을 비교해서 최소값인 1이 SWAP되어 왼쪽 리스트에 온 결과값을 step1에서 확인할 수 있고 슬라이드 그림과 실제 출력값 또한 동일하단 것을 확인할 수 있습니다.

Step2: step1의 상태에서 왼쪽리스트(배열)의 끝 다음 index이자 오른쪽 리스트의 첫번째인 3을 나머지 오른쪽 리스트들의 item들 중 최소값인 2와 비교하여 정렬이 맞지 않음을 확인하고 SWAP하고 왼쪽 리스트 1증가시켜 넣어 1, 2가 정렬된 수행을 하였음을 알 수 있고 슬라이드와 동일한 결과인 것을 알 수 있습니다.

Step3: step2의 상태에서 왼쪽리스트(배열)의 끝 다음 index이자 오른쪽 리스트의 첫번째인 8과 나머지 오른쪽 리스트의 item들 중 최소값인 3과 비교하여 정렬이 맞지 않음을 확인하고 SWAP하고 왼쪽 리스트 1증가시켜 넣어 1,2,3이 정렬된 수행을 하였음을 알 수 있고 슬라이드와 동일한 결과인 것을 알 수 있습니다.

Step4: step3의 상태에서 왼쪽리스트(배열)의 끝 다음 index이자 오른쪽 리스트의 첫번째인 5와 나머지 오른쪽 리스트의 최소값인 7을 비교하여 정렬이 옳바른 상태임을 확인하여 그대로 왼쪽 리스트 1증가시켜 넣어 1,2,3,5의 정렬 상태로 되었음을 확인할 수 있습니다. 또한 이 결과가 슬라이드와 동일함을 비교하면 알 수 있습니다.

Step5: step4의 상태에서 왼쪽리스트(배열)의 끝 다음 index이자 오른쪽 리스트의 첫번째인 8과 나머지 오른쪽 리스트 item인 7을 비교시켜 정렬이 맞지 않음을 확인하여 이 두 개를 SWAP하고 왼쪽 리스트 1증가시켜 넣어 12,3,5,7의 정렬 상태가 되었고, 나머지 오른쪽 유일하게 남은 item인 8은 정렬상태를 앞과 비교하여 맞음으로 그대로 왼쪽 리스트에 넣어 오른쪽 리스트를 공백상태로 하여 정렬을 완성시켰음을 볼 수 있습니다. 이는 슬라이드와 비교하여도 동일한 상태임을 확인할 수 있습니다.

2. Insertion sort(삽입 정렬)

2.1

\*insertion sort 알고리즘 기본 원리 쉽고 자세한 설명:

-insertion sort, 삽입 정렬은 쉽게 말해 “삽입”해서 정렬하는 정렬 기법입니다. 삽입 정렬은 선택 정렬과 비슷하게 정렬된 왼쪽 리스트(배열) 부분과 정렬 안 된 오른쪽 리스트(배열)부분으로 나누어 집니다. 삽입 정렬은 각 step마다 숫자 크기 상관없이 무조건 오른쪽 리스트에서 첫번째 순서로 오는 항목을 왼쪽 리스트에 정렬 규칙에 맞추어 바른 위치에 “삽입”합니다. 이때 삽입할 위치 오른쪽에 있는 “모든”항목들, 어차피 오른쪽 리스트의 가장 첫 항목을 사용하니 삽입할 위치에 있는 모든 “왼쪽 리스트(배열)항목”들만 옮겨주면 됩니다. 이렇게 옮기면 당연히 왼쪽 리스트에는 정렬된 게 하나 추가되니 1증가하고, 오른쪽에선 당연히 하나가 빠져나가서 정렬이 되니 1감소합니다. 이러한 작업 step들을 계속 수행하면서 오른쪽 리스트(배열)비게 되면 정렬할 것이 더 안남아있게 되니 정렬이 완료되는 것입니다. 첫 step부터 왼쪽 리스트에 무엇인가 존재해야하니 우리는 총 n-1개의 숫자를 비교, 삽입함으로 총 n-1번의 step을 수행한다고 할 수 있습니다.

2.2

// ======================================================================

#include <stdio.h>

#include <stdlib.h>

#include "insertion.h"

// ======================================================================

void insertion\_sort(int list[], int n)

{

int i, j, key;

for (i = 1; i < n; i++) {

key = list[i];

for (j = i - 1; (j >= 0) && (list[j] > key); j--)

list[j+1] = list[j];

list[j+1] = key;

print\_array(list, n);

}

}

// ======================================================================

\*이렇게 프로그램한 이유(with알고리즘 원리 + 각 index 설명):

-먼저 배열의 index로써 사용하기 위해 I,j를 선언하고 비교를 위해 오른쪽 리스트의 첫째 항목을 저장할 변수 key를 선언하고 그 이후 본격적으로 프로그램을 시작합니다. 먼저 for문을 하나 열어서 i를 각 step마다의 오른쪽 리스트 첫번째 index를 지정하기 위해 for문 조건에 i를 1부터 index의 끝인 n-1까지 반복하도록 하였습니다. 그 다음 이 index i인 오른쪽 첫째 항목을 비교에 사용하고 후에 있을 배열 항목들의 이동에 영향을 주지 않도록 하기 위해 key변수에 따로 보관합니다. 이 다음 for문 중첩을 하나 열어서 조건에 j를 사용하는데, 이 j는 왼쪽 항목의 가장 마지막 항목의 index로써 사용합니다. 그리고 여기서의 for문은 선택 정렬과는 다르게 하나의 조건으로써 사용합니다. J가 index이니 0보다 당연히 같거나 커야하고 그리고 동시에 index i에 해당하는 list의 항목이 오른쪽 리스트의 첫번째 항목인 key보다 클 경우는 즉 왼쪽 리스트에 있는 숫자가 오른쪽 리스트에 있는 숫자보다 커서 정렬이 안맞음으로, j+1 index의 위치에 j index 위치의 항목을 넣음으로써 한 칸 뒤로 밀리게 하였습니다. 이걸 굳이 if문이 아니라 for문으로 반복시킨 것은 정확한 위치에 넣어야 되고 그러기 위해선 왼쪽의 마지막 index인 j를 내려오면서 한번에 한번씩 비교해서 오른쪽으로 옮겨 key값이 들어갈 올바른 위치를 만들어야 되기 때문입니다. 이렇게 옮겨서 오른쪽 첫째 항목을 침범한다 하더라도 미리 key값에 저장시켜 놓았으므로 우리는 계속 기억할 수 있습니다. 그리고 이렇게 저장시켜놓은 key값을 j+1 index에 해당하는 위치에 넣는데, 이는 이전 for문 중첩을 통해 바른 위치까지, j+1까지 내려오고난 다음 for문에서 j를 한번 더 감소시킨다음 조건을 따져서 for문 중첩을 종료시키기 때문에 이렇게 j+1 index에 해당하는 항목 자리에 key을 넣어 하나의 step을 완료시키고 미리 test파일에 만들어져있는 print\_array함수를 호출하여 step 수행의 결과를 출력하도록 하였습니다.

2.3

s1910783@oak:hw07$ gcc test-insertion.c insertion.c -o test-insertion

s1910783@oak:hw07$ ./test-insertion

step 0: 5 3 8 1 2 7

step 1: 3 5 8 1 2 7

step 2: 3 5 8 1 2 7

step 3: 1 3 5 8 2 7

step 4: 1 2 3 5 8 7

step 5: 1 2 3 5 7 8

\*각 step 설명(슬라이드 12-9참고):

-위 출력 결과에서의 각 step 수행 작업을 아래 슬라이드를 보고 비교하면서 같이 설명 드리겠습니다.



Step0: step0는 아직 아무런 정렬 수행도 일어나지 않은 초기 상태입니다.

Step1: step0에서 왼쪽 리스트 부분 1개에 오른쪽 리스트 부분 5개 중 첫번째 항목인 3을 선택하여 왼쪽 리스트에 5와 비교하여 올바른 위치인 5 앞에 넣어 3,5의 순서로 정렬이 되었고, 슬라이드와 동일하다는 것을 알 수 있습니다.

Step2: step1에서 정렬된 왼쪽 리스트 이후 시작되는 오른쪽 리스트의 가장 첫번째 항목인 8을 왼쪽 리스트와 비교하여 정렬이 되었는지 체크한 결과, 이미 오름차순으로 정렬이 되어 있으므로 그 자리가 그대로 유지된 상태로 왼쪽 리스트 부분으로 추가되고 그걸 슬라이드와 비교하면 동일하다는 것을 알 수 있습니다.

Step3: step2에서 정렬된 왼쪽 리스트 이후 시작되는 오른쪽 리스트의 가장 첫번째 항목인 1을 왼쪽 리스트와 비교하여 정렬이 올바른 위치인 가장 앞으로 넣기 위해 넣을 자리의 오른쪽 항목들을 1칸씩 밀고 그 자리에 1을 넣어 1,3,5,8의 정렬로 되었다는 것을 볼 수 있습니다. 또한 슬라이드와 비교해보면 동일한 결과로 나왔다는 것을 알 수 있습니다.

Step4: step3에서 정렬된 왼쪽 리스트 이후 시작되는 오른쪽 리스트의 가장 첫번째 항목인 2를 왼쪽 리스트와 비교하여 정렬이 올바른 위치인 1다음으로 넣기 위해 그 자리의 모든 오른쪽 항목을 한칸씩 밀고 그 자리에 넣어 1,2,3,5,8의 정렬이 되었다는 것을 볼 수 있습니다. 이 결과는 슬라이드에서도 확인할 수 있어서 동일하다는 것을 알 수 있습니다.

Step5: step4에서 정렬된 왼쪽 리스트 이후 시작되는 오른쪽 리스트의 가장 첫번째 항목인 7을 왼쪽 리스트와 비교하여 정렬이 올바른 위치인 5와 8사이에 넣기 위해 8을 한칸 뒤로 밀고 그 자리에 7을 넣어 1,2,3,5,7,8의 정렬을 완성시킵니다. 오른쪽 리스트가 공백이 되었기 때문에 정렬이 끝나 step이 여기서 끝납니다. 이러한 결과는 슬라이드에서도 동일하게 확인할 수 있습니다.

3. Bubble sort(버블 정렬)

3.1

\*bubble sort 알고리즘 기본 원리 쉽고 자세한 설명:

-bubble sort, 버블 정렬은 정말 간단하게 말하자면 첫 항목부터 인접한 것들끼리 비교해서 순서를 바꿔주면서 끝 리스트(배열)항목까지 가는게 한 세트, 즉 1개 step으로 구성되어 최종적으로 정렬이 완료될 때까지 진행되는 정렬 기법입니다. 자세히 설명하자면, 1개의 step에서 인접한 2개의 항목을 비교하여 순서대로 되어 있지 않으면 서로 교환합니다. 여기서 이 “비교 및 교환(bubble)”과정을 리스트(배열)의 왼쪽 시작부터 오른쪽 끝까지 가면서 한 step동안 반복합니다. 한 step이 끝날 때마다 그 step에서의 최대값이 오른쪽 리스트(배열)의 가장 끝에 저장됩니다. 예를 들어 step1에서는 최대값이 배열의 가장 끝에 저장되고, 이 저장된 항목을 제외한 n-2까지의 범위를 버블 비교하여 거기에서의 최대값을 오른쪽 끝에 이어 저장합니다. 이렇게 step을 반복하여 비교하는 item들의 범위를 1개씩 줄여가면서 거기에서의 최대값을 각 범위의 가장 오른쪽 끝에 저장하게 되면서 정렬이 차차 되는 것입니다. 지금까지 봐왔던 선택이나 삽입 정렬과는 정렬되는 위치가 반대라고 할 수 있습니다.

3.2

// ======================================================================

#include <stdio.h>

#include <stdlib.h>

#include "bubble.h"

// ======================================================================

void bubble\_sort(int list[], int n)

{

int i, j, temp;

for (i = n - 1; i > 0; i--) {

for (j = 0; j < i; j++)

if(list[j] > list[j+1])

SWAP(list[j], list[j+1], temp);

print\_array(list, n);

}

}

// ======================================================================

\*이렇게 프로그램한 이유(with알고리즘 원리 + 각 index 설명):

-먼저 배열의 index접근을 위한 변수 I, j를 선언하고 bubble교환을 위한 임시 변수 temp를 선언합니다. 그 다음 본격적으로 프로그램을 시작하는데, for문 시작하여 i를 조건으로 주는데, 여기서 이 i는 bubble교환을 통해 산출된 각 step에서의 최대값을 이 step의 배열의 오른쪽 끝 index인 n-1을 가리키는 변수 index이고, step을 거듭할수록 오른쪽에 최대값을 저장하여 내려와야하니 index i또한 감소하는 형태로 반복하여 각 step을 수행합니다. 그 다음 for문 중첩을 열어서 j을 조건으로 주는데, 여기서 이 j는 가장 처음 index부터 시작하여 그 step에서의 오른쪽 끝까지를 인접한 것들끼리 비교하도록 j, j+1을 사용하게 되는 비교를 위한 index변수라고 할 수 있습니다. 이 for문 중첩을 열어서 if문 조건을 주는데, 방금 설명했던 대로 bubble비교교환, 즉 인접한 두 개를 비교, 교환하기 위해 j index의 항목과 j+1 index의 항목을 비교, 앞의 것인 j항목이 더 크면 정렬이 맞지 않는 것이므로 j항목과 j+1항목을 SWAP하여 주도록 하였습니다. 그 다음 for문 중첩도 빠져 나오면 총 1개의 step을 끝낸 것이므로 print\_array함수를 통해 각 step별로 정렬 수행 결과를 출력할 수 있게 하였습니다.

3.3

s1910783@oak:hw07$ gcc test-bubble.c bubble.c -o test-bubble

s1910783@oak:hw07$ ./test-bubble

step 0: 5 3 8 1 2 7

step 1: 3 5 1 2 7 8

step 2: 3 1 2 5 7 8

step 3: 1 2 3 5 7 8

step 4: 1 2 3 5 7 8

step 5: 1 2 3 5 7 8

\*각 step 설명(슬라이드 12-12참고):

-슬라이드에는 step1만 나와있다. 일단 이것을 출력결과와 보며 step의 수행을 설명하겠습니다.



Step0: step0는 아직 정렬을 수행하지 않은 초기 상태임으로 초기 그대로가 출력되었다는 것을 알 수 있습니다.

Step1: step1은 슬라이드를 비교해보며 알 수 있는데, 앞에서부터 인접한 2개끼리를 비교하면서 진행, 순서가 맞지 않으면 두 개의 자리를 SWAP하여 작은 것이 앞으로, 큰 것이 뒤로 오도록 하여 마치 계단 형식처럼 이 step에서의 가장 큰 값이 오른쪽 가장 끝에 마치 정렬된 듯한 모습을 슬라이드와 출력 결과를 보면 알 수 있습니다.

Step2: step2는 step1과 마찬가지인데 이미 이전 step에서의 최대값인 8이 오른쪽 끝에 위치해 있으므로, 이걸 제외한 나머지 것들만 가지고 비교를 진행, 이 step에서의 최대값인 7이 차피 이전 step에서 swap되면서 제대로 정렬되어 있다는 것을 알 수 있습니다. 다만 앞에서부터 비교, 교환되니 앞 부분, 5,3이 3,5의 순서로 바뀌는 등의 수행이 있긴 하였습니다.

Step3: step3에서도 step2와 마찬가지로 이전 step에서의 최대값을 제외한 나머지 왼쪽에 있는 것들끼리 비교, 교환하여 결국은 최대값은 5가 오른쪽에 정렬되었음을 확인할 수 있습니다.

Step4: step4에서도 이전 step들과 마찬가지로 이전 최대값들을 제외한 나머지 왼쪽 것들을 가지고 비교, 교환을 하는데, 여기서 이미 바꿀 것이 없고 정렬은 완성되었으므로 변화가 없습니다.

Step5: 이것도 마찬가지로 이미 정렬은 완성된 상태이고, 다만 바깥 for문에서 각 step을 담당하는 I index가 남아있어서 여기까지 step이 진행되었다는 것을 우리는 알 수 있습니다.

끝.