Report 2

학 번	32202546
성 명	안지성
담당교수	황두성교수님
제 출 일	2021.10.10

1. 문제 기술

기능과 사양(Function and specification)

• 주어진 입력 데이터의 크기에 무관한 정렬 함수의 구현

• 정렬 알고리즘 -> <<구현>>

1. Bubble sort algorithm
2. Insertion sort algorithm
3. Selection sort algorithm

• 각 정렬 알고리즘을 구현하기 위한 자료구조

- C 언어의 기본 자료구조인 배열
- 기본 자료구조의 변형

평가(Evaluation)

• 알고리즘은 독립된 함수로 설계되어야 함.

• 정렬할 데이터 크기는 10000에서 1000000의 범위에서 선택

• 입력데이터는 random 함수를 사용하며 10개의 데이터 집합으로 구성

• 정렬 알고리즘은 데이터 크기 별 약 5개 이상의 입력 데이터에 대한 실행시간을 측정

• 출력

- 구현된 정렬 알고리즘의 2개 이내 테스트 결과
- 정렬 알고리즘의 배열의 크기에 대한 평균 실행 시간의 그래프(예: MSExcel, Matplotlib, R, Matlab 등 이용)의 결과를 데이터 크기와 실행시간을 비교하고 설명하시오.

2. 분석과 설계 방법

우선 main.c에 main함수를 만들어서 전반적인 코드를 실행시키고 공통적인 특성을 가진 함수들을 묶어서 각각 새로운 소스코드인 .c파일로 만들고 이를 main함수에서 사용할 수 있게 각 소스코드 파일에 해당하는 헤더파일인 .h파일들을 만들어 각 함수들을 정의한다. main함수에서 10000 ~ 1000000 사이로 배열의 크기를 입력하고 매번 정렬을 전에 배열의 각 인덱스에 해당하는 메모리에 입력받은 데이터 크기만큼의 수를 무작위로 하나씩 넣는다. 구현한 각각의 정렬 함수마다 5번씩 배열을 정렬시키고 5개의 데이터 값에 해당하는 평균을 구한다. 이 때, 정렬 함수들을 사용하기 이전에는 start로, 사용한 이후에서는 end로 실행 시간을 측정하고 result에 end-start로 실행하는데 걸린 시간을 측정하여 기록한다. 그리고 이 모든 결과들을 출력으로 보여서 확인하며 체크한다. 마지막으로 각기 다른 데이터의 크기로 총 10번의 테스트를 통해 각 테스트마다 걸린 평균 수행 시간들을 그래프로 그려 비교해본다.

3. 프로그램 리스트(Make file 포함)

<<사용하는 파일 리스트>>

- makefile
- main.c
- common.h, common.c
- sort.h, sort.c

- makefile

```
#
# makefile
#
CC=gcc -g
LIBS= -lstdc++

#
# all: hw
#
hw: main.o common.o sort.o
	gcc -o hw main.o common.o sort.o

#
main.o: main.c
	gcc -c main.c -o main.o

#
common.o: common.c common.h
	gcc -c common.c -o common.o

#
sort.o: sort.c sort.h
	gcc -c sort.c -o sort.o

#
clean:
	rm hw.exe *.o
```

- main.c

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

```
#include "common.h"
#include "sort.h"

int main(void){
    int *a, n;
    int i, j, c;
    int flag = 0;
    long start, end, result = 0;
    srand(time(NULL)); // rand()함수값이 실행될 때마다 다 다른 랜덤한 값을 불러오기 위해 srand()를 선언하고 계속 바뀌는 time을 seed값으로 주어서 계속 다른 값들을 채워넣는다.

    printf("정렬할 데이터의 크기(10000 ~ 1000000): ");
    scanf("%d", &n);
    printf("\n");

    a = (int*)malloc(sizeof(int)*n);

    printf("모든 출력은 20개로 제한하였다.\n\n");

    for(c=0; c<15; c++){
        for(i=0; i<n; i++){
            a[i] = (rand()%n)+1; // 1~n까지의 수를 배열에 넣는다.
            for(j=0; j<i; j++){
                if(a[i] == a[j])
                    i--;
            }
        }

        printf("<정렬 전>\n");
        print_array(a, n);

        if(0<=flag && flag<5){
            printf("<삽입 정렬 -> ");
            start = get_runtime();
            insertion(a, n);
            end = get_runtime();

        }
        else if(5<=flag && flag<10){
```

```
            printf("<버블 정렬 -> ");
            start = get_runtime();
            bubble(a, n);
            end = get_runtime();
        }

        else{
            printf("<선택 정렬 -> ");
            start = get_runtime();
            selection(a, n);
            end = get_runtime();
        }

        printf("정렬 후>\n");
        result += (end-start);
        print_array(a, n);
        printf("걸린시간>> %ld\n\n", end-start);

        if(flag == 4 || flag == 9 || flag == 14){
            printf("평균값>> %ld\n\n\n", result/5);
            result = 0;
        }
        flag++;
    }
    free(a);

    return 0;
}
```

<<Common>>

- common.h

```
#ifndef _COMMON_

#define _COMMON_

void print_array(int *, int);
long get_runtime(void);

#endif
```

- common.c

```
#include <stdio.h>
#include <time.h>
#include "common.h"

// 출력은 10~20개 정도로 보여줌
void print_array(int *a, int n){
    for(int i=0; i<20; i++){
        printf("%d ",a[i]);
    }
    printf("\n");
}

long get_runtime(void)
{
   clock_t start;
   start = clock();
   return ((long)((double)start * 100.0 / (double)CLOCKS_PER_SEC));
}
```

<<Sort>>

- sort.h

```
#ifndef _SORT_

#define _SORT_

void insertion(int *, int);
void bubble(int *, int);
void selection(int *, int);

#endif
```

- sort.c

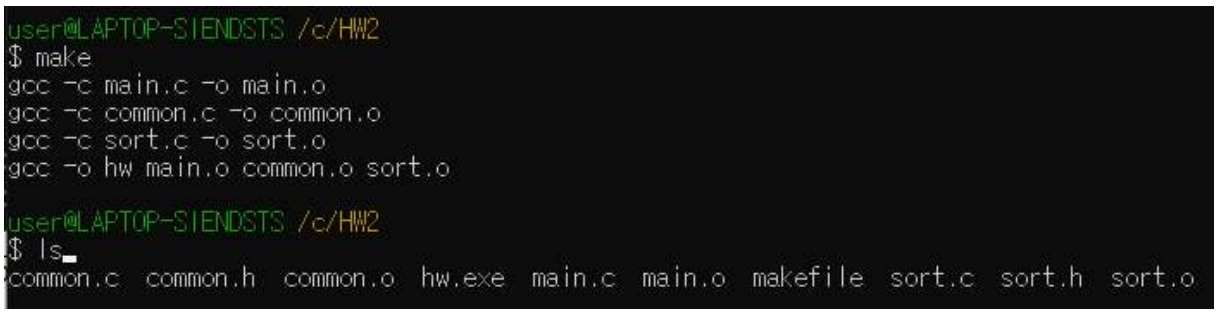
```
#include <stdio.h>
#include <stdbool.h>
#include "sort.h"

// 오름차순 삽입정렬
void insertion(int *a, int n){
    int i, j, move;
```

```
    int val;

    for(i=1; i<n; i++){
        val = a[i];
        j = i;
        if(a[j-1] > val)
            move = true;
        else
            move = false;

        while(move){
            a[j] = a[j-1];
            j = j-1;
            if(j>0 && a[j-1]>val)
                move = true;
            else
                move = false;
        }
        a[j] = val;
    }
}

// 오름차순 버블정렬
void bubble(int *a, int n){
    int i, j;
    int val;

    for(i=n-1; i>0; i--){
        for(j=0; j<i; j++){
            if(a[j]>a[j+1]){
                val = a[j];
                a[j] = a[j+1];
                a[j+1] = val;
            }
        }
    }
}

// 오름차순 선택정렬
void selection(int *a, int n){
```

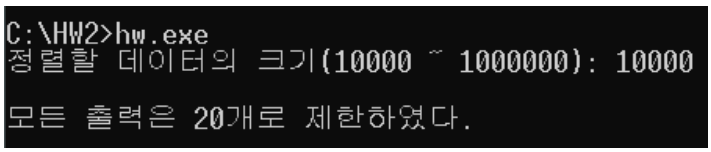
```
    int i, j, least;
    int val;

    for(i=0; i<n-1; i++){
        least = i;
        for(j=i+1; j<n; j++){
            if(a[j] < a[least])
                least = j;
        }
        val = a[least];
        a[least] = a[i];
        a[i] = val;
    }
}
```

4. 결과(출력 결과의 예로 한 개만 작성)

makefile 실행 -> make command 입력



hw.exe 실행 (ex -> 크기: 10000)



<<삽입정렬>> => 5번 테스트

```
<정렬 전>
96 2677 2966 6023 1707 6780 7217 918 9493 5262 7825 8765 2796 2347 4734 5700 5523 6802 3137 9531
<삽입 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 5

<정렬 전>
4063 5375 3410 2063 8824 577 4395 382 9074 5258 3477 8253 2959 2744 2266 5695 8242 8020 9343 8281
<삽입 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 5

<정렬 전>
9479 1381 8728 1977 3246 8741 9664 6817 6613 1232 2875 8037 587 2620 6496 4878 3952 6 1667 4038
<삽입 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 5

<정렬 전>
1239 2450 7127 7224 4841 842 5563 695 309 9222 7700 3719 3162 7164 48 6200 6658 480 9717 814
<삽입 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 5

<정렬 전>
1637 8852 6484 5644 9341 315 9088 8711 2563 5721 7698 1016 3994 1704 8081 9738 8705 52 8533 296
<삽입 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 5

평균값>> 5
```

<<버블정렬>> => 5번 테스트

```
<정렬 전>
161 7685 1202 7260 6988 6668 8173 2089 6330 8187 2538 2167 1508 9762 745 1076 4019 5672 8338 7703
<버블 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 20

<정렬 전>
6619 5959 7702 1425 1048 1102 59 9641 2833 7622 3067 4314 9301 751 3003 2362 1798 5930 9367 355
<버블 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 20

<정렬 전>
5489 4428 7779 8966 1319 4629 1800 9678 1566 6149 4201 8033 481 3353 2631 4190 4034 9661 9673 4851
<버블 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 21

<정렬 전>
5427 9215 5201 6700 9989 7884 96 758 6596 7328 567 7005 2856 1007 1604 4567 1928 5102 9088 1772
<버블 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 20

<정렬 전>
9060 3854 2457 8704 374 9264 5442 2188 8072 9145 2104 2327 3563 2472 7571 5663 4849 7927 6706 7422
<버블 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 20

평균값>> 20
```

<<선택정렬>> => 5번 테스트

```
<정렬 전>
7294 8589 9484 9384 2439 8789 574 4203 5960 7120 4414 7478 7411 8556 1572 4036 805 8735 1247 8795
<선택 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 9

<정렬 전>
9448 1350 4606 3757 8689 9555 6656 1500 5951 4049 8109 2533 2535 2145 6542 2123 7789 8700 7676 4610
<선택 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 8

<정렬 전>
6646 8202 9876 133 831 1545 2325 4200 2318 7182 8727 2385 210 9703 8923 3027 2754 1857 2256 3870
<선택 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 9

<정렬 전>
3087 2580 1239 2135 7137 7140 7461 7338 7344 3408 9447 9897 4368 4071 5175 5064 8516 9799 4442 121
<선택 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 10

<정렬 전>
3934 2138 5360 9210 1354 725 5855 689 5437 1942 5983 3667 7833 6931 5955 8542 7452 6759 6021 5933
<선택 정렬 -> 정렬 후>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
걸린시간>> 9

평균값>> 9
```

<<시간복잡도>>

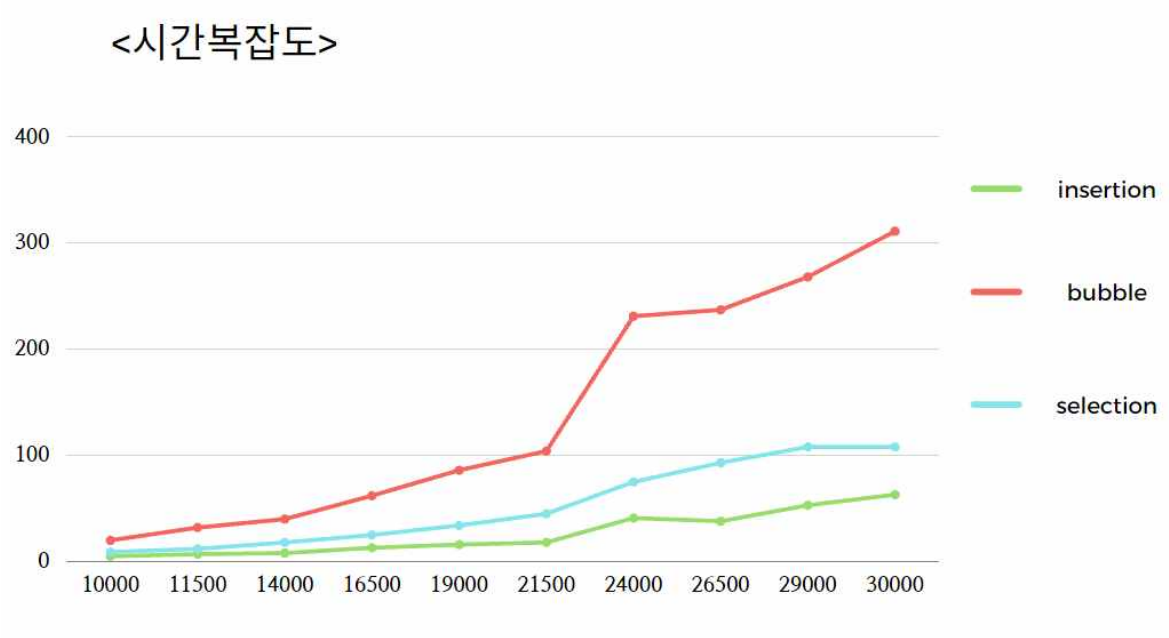
<sort>

best case: O(n) (이미 정열)

삽입정렬 -> insertion() -> worst case: $O(n^2)$ (worst case만 O(n^2)라서 아마 제일 적은 시간이 걸릴 것이다.)

버블정렬 -> bubble() -> **O(n^2)** (내부 for문에서 swap을 계속 진행하므로 시간이 제일 오래 걸릴 것이다.)

선택정렬 -> selection() -> **O(n^2)** (내부 for문에서 swap할 index만 선택하고 외부 for문에서 그 index의 값만 swap을 진행하므로 bubble보다는 적게 시간이 걸릴 것이다.)



<<결론>>

insertion sort, bubble sort, selection sort 3개의 시간복잡도를 구해보고 10개의 데이터 크기마다 5번씩 테스트를 해서 평균을 구해 그래프를 그려 본 결과, 예상했던 결과와 유사하게 나왔다. 즉, 시간복잡도를 구하면 이렇게 직접 테스트를 해보지 않더라도 어느 정도는 합리적으로 실행시간을 유추할 수 있다.

5. 토의와 개선점

• 테스트 결과에 대한 비교(ex: 계산의 제한성, 올바른 계산 결과의 여부 등)

-> 테스트 시, 배열에 값들을 랜덤하게 주었기 때문에 best case와 worst case사이에서 측정시간 변동이 발생한다. 그렇기에 insertion sort, bubble sort, selection sort가 같은 데이터 크기일 때, 걸리는 시간이 짧은 순으로 insertion sort > selection sort > bubble sort 처럼 나올 거라고 시간복잡도를 구해서 예상하였지만 값이 같아지거나 순서가 바뀔 수 있는 가능성이 있다. 대신 값이 커질수록 그 오차는 줄어들어 예상과 흡사한 결과가 더 잘 나올 것 같다.

• 데이터 크기에 따른 측정된 실행시간 비교

-> 데이터 크기가 커질수록 같은 함수를 쓰더라도 더 실행시간이 늘어남을 알 수 있다. 결론은 데이터 크기가 늘어날수록 실행시간이 더 커지며, bubble sort > selection sort > insertion sort 순으로 실행시간이 크다.

스스로 과제 평가

모듈라 기법 사용	Make 사용	테스트	결과	분석, 토의&개선점
O	O	O	O	O