

Sort Testing

학번 : 201002513

이름 : 최 혁수

☐ 실습 3. Sort testing

○ 개요 & 알고리즘

- Time expression

○ Source Code

- Buuble Sort
- Insertion Sort
- Quick Sort

○ 실행결과 분석 & 구현상의 오류 및 한계

☐ 소감문 & 한계성

☐ All Source Code

실습 3. Sort Testing

○ 개요 & 알고리즘

- Time expression

○ Source Code

- Buuble Sort
- Insertion Sort
- Quick Sort

○ 실행결과 분석 & 구현상의 오류 및 한계

Sort Testing

□ 개요

정렬(Sort) 알고리즘 중 버블정렬(Bubble Sort), 삽입정렬(Insertion Sort), 퀵정렬(Quick Sort)의 연산시간을 초당 시간주기(Clock ticks)로 표현한다.

□ 접근방법

- 각 정의된 개수에 따른 난수를 발생하여, 정렬 알고리즘의 시간계산을 구현
- Time.h 정의된 clock_t, clock()를 이용하여 함수의 프로세서 시간을 구함
- 난수의 발생차에 따른 한계를, 같은 난수를 각 정렬 알고리즘에 대입하여 시간을 계산하는 것으로 대처함
- quick time의 시간이 빨라 0.000000으로 표현될경우가 많아 *10을 하여 표현값 증대로 구현

□ Time expression

○ Header File

#include "time.h" : clock_t, clock(); 정의

○ Function

clock_t start, stop; : clock_t 틱 수를 저장하는 자료형

* clock_t 는 long형으로 정의된다

start = clock(); : 시작

Sort_Function(Sort_Data, totalNum); : 각 정렬함수 정의

stop = clock(); : 끝

즉, 시작, 끝값의 ticks을 구한값의 차(stop-start)를 구하여
Sort_Function()의 시간을 구한다

Sort Time Source Code(1/4)

□ sort_time.h

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

/* 2byte 표현값 -32768~32767으로 MAX, MIN 정의 */
/* 추가적으로, 각 정렬에 필요한 1차원 배열의 크기로 쓰이는 MAX, MIN 이므로 */
/* 컴파일러는 정적할당에 한계치를 0x7FFFFFFF 약 1MB(1,000,000)로 둔다 */
#define MAX_INT 32767
#define MIN_INT -32768

clock_t start, stop; // clock()에 의해 계산된 초당 시간주기(clock ticks)의 수를 나타낸다.
double duration;

/* 각 정렬에 쓰이는 배열 */
int bubbleData[MAX_INT];
int insertionData[MAX_INT];
int quickData[MAX_INT];

/* 정렬함수 정의 */
void bubbleSort(int*, int);
void insertionSort(int *, int);
void quickSort(int*, int, int);

/* 정렬에 대한 clock ticks를 구하는 함수 */
void sortTest(int*, int*, int*, int);

/* 난수구현 */
void createRandomNum(int*, int*, int*, int);
```

□ void createRandomNum(int*, int*, int*, int)

```
void createRandomNum(int* bubbleData, int* insertionData, int* quickData, int totalNum){
    int data;
    int i;
    srand((unsigned)time(NULL)); // 난수발생
    quickData[totalNum] = MAX_INT; // 최대값 대입

    for(i=0; i<totalNum; i++){ // totalNum 갯수만큼 난수 발생후, 각 정렬에 쓰일 변수에 저장
        data = rand() % RAND_MAX;
        bubbleData[i] = data;
        insertionData[i] = data;
        quickData[i] = data;
    }
}
```

Sort Time Source Code(2/4)

□ void bubbleSort(int *, int)

```
/* 버블정렬 */
void bubbleSort(int *bubbleData, int totalNum){
    int count = totalNum -1;
    int temp;
    int i, j;

    for(i=0; i<count; i++)
        for(j=i+1; j<totalNum; j++){ // i+1 ~ totalNum-1 반복
            if(bubbleData[i]>bubbleData[j]){ // 좌, 우 비교후 좌측이 크면 Swap.
                temp=bubbleData[i];
                bubbleData[i]=bubbleData[j];
                bubbleData[j]=temp;
            }
        }
    }
```

□ void insertionSort(int*, int)

```
/* 삽입정렬 */
void insertionSort(int* insertionData, int totalNum)
{
    int i, j;
    int temp;
    for (i = 1; i<totalNum; i++ )
    {
        temp = insertionData[(j=i)]; // insertionData[1]값을 temp 저장, 첫 삽입정렬 비교 데이터
        /* 데이터값과 앞의 수를 비교해가면서 앞의 값이 클때 Swap 기능 */
        while (--j>=0 && temp<insertionData[j]) insertionData[j+1] = insertionData[j];
        /* 정렬이 끝난 후 삽입해야 될 위치에 첫 temp 저장 값을 넣는다. */
        insertionData[j+1] = temp;
    }
}
```

Sort Time Source Code(3/4)

□ void quickSort(int *, int, int)

```
/* 퀵 정렬 */
void quickSort(int* quickData, int left, int right)
{
    /* Left_Num, Right_Num = 맨 왼쪽, 오른쪽 값을 유지하기 위한 변수 */
    int pivot, Left_Num, Right_Num;
    Left_Num = left;
    Right_Num = right;
    /* 첫 중간값은 맨 왼쪽 값으로 지정 */
    pivot = quickData[left];

    /* left, right 각각 오른쪽, 왼쪽으로 탐색해가면서 큰값을 찾아내는데, */
    /* 왼쪽, 오른쪽을 나타내는 값이 서로 뒤바뀔때, 즉 left값이 right값보다 더 오른쪽에 있을때까지 수행 */
    while (left < right)
    {
        /* 피벗보다 작은 오른쪽값이 나타날때까지 검색 */
        while ((quickData[right] >= pivot) && (left < right)) right--;
        /* 왼쪽값에 오른쪽값을 저장 */
        if (left != right)
        {
            quickData[left] = quickData[right];
            left++;
        }
        /* 피벗보다 큰 왼쪽값이 나타날때까지 검색 */
        while ((quickData[left] <= pivot) && (left < right)) left++;
        /* 오른쪽값에 왼쪽값을 저장 */
        if (left != right)
        {
            quickData[right] = quickData[left];
            right--;
        }
    }
    /* 피벗을 재위치에 저장, quickData[left] 이전값들은 피벗보다 작다 */
    quickData[left] = pivot;
    /* 현재 pivot을 기준으로, left<pivot<right 의 데이터 크기를 가진다. */
    pivot = left;
    left = Left_Num;
    right = Right_Num;

    /* pivot을 기준으로 반으로 나뉜 left, right 데이터들을 재귀적으로 다시 정렬 */
    if (left < pivot)
        quickSort(quickData, left, pivot-1);
    if (right > pivot)
        quickSort(quickData, pivot+1, right);
}
```

Sort Time Source Code(4/4)

□ void sortTest(int*, int*, int*, int)

```
/* 각 정렬별 시간주기 탐색 */
void sortTest(int* bubbleData, int* insertionData, int* quickData, int totalNum)
{
    printf("Number = %d\n", totalNum);
    createRandomNum(bubbleData, insertionData, quickData, totalNum);
    start = clock(); // star ~ stop = bubbleSort()의 시간주기 탐색
    bubbleSort(bubbleData, totalNum);
    stop = clock();
    duration = (((double)(stop-start))/CLK_TCK)*10; // 초당 시간주기 계산
    printf("Bubble Time : %f\n", duration);

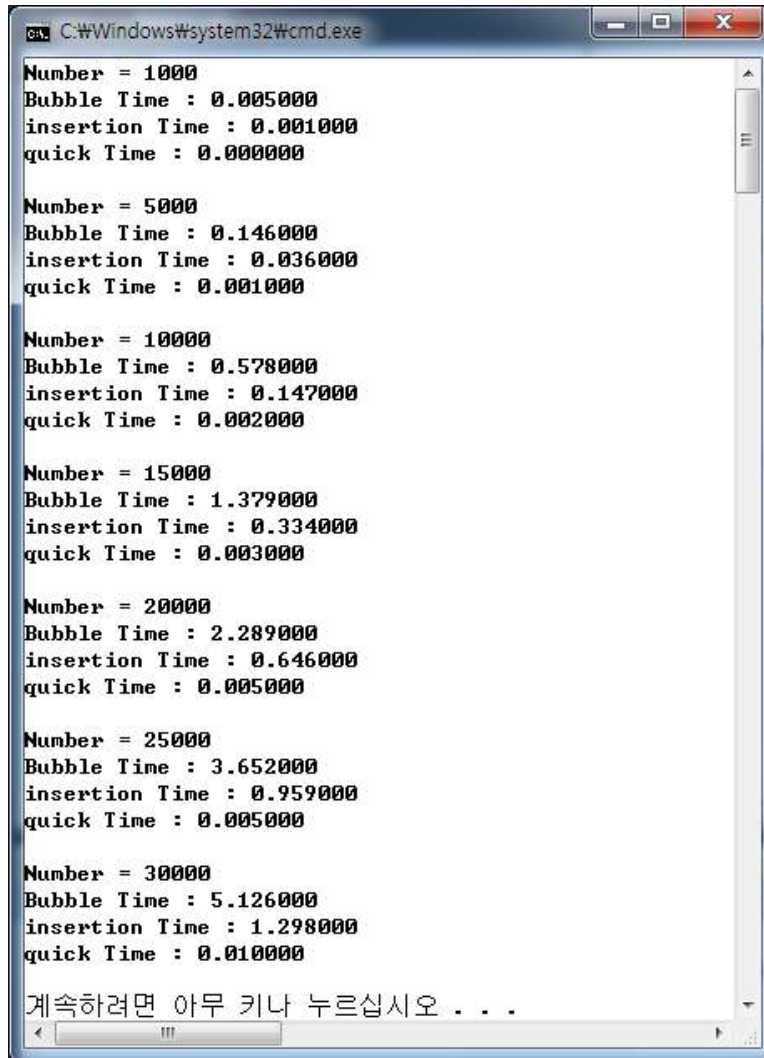
    start = clock();
    insertionSort(insertionData, totalNum);
    stop = clock();
    duration = (((double)(stop-start))/CLK_TCK)*10;
    printf("insertion Time : %f\n", duration);

    start = clock();
    quickSort(quickData, 0, totalNum);
    stop = clock();
    duration = (((double)(stop-start))/CLK_TCK)*10;
    printf("quick Time : %f\n", duration);

    printf("\n");
}
```


실행결과 분석 & 구현상의 오류 및 한계

□ 실행 결과화면



```
C:\Windows\system32\cmd.exe
Number = 1000
Bubble Time : 0.005000
insertion Time : 0.001000
quick Time : 0.000000

Number = 5000
Bubble Time : 0.146000
insertion Time : 0.036000
quick Time : 0.001000

Number = 10000
Bubble Time : 0.578000
insertion Time : 0.147000
quick Time : 0.002000

Number = 15000
Bubble Time : 1.379000
insertion Time : 0.334000
quick Time : 0.003000

Number = 20000
Bubble Time : 2.289000
insertion Time : 0.646000
quick Time : 0.005000

Number = 25000
Bubble Time : 3.652000
insertion Time : 0.959000
quick Time : 0.005000

Number = 30000
Bubble Time : 5.126000
insertion Time : 1.298000
quick Time : 0.010000

계속하려면 아무 키나 누르십시오 . . .
```

Number은 main함수에서 정의된 개수에 따라 난수를 발생시키고 있으며, 각 난수 발생에 따른 정렬 알고리즘들의 시간을 계산하여 출력하고 있다.

□ 구현상의 오류 및 한계

- (stop-start)/CLK_TCK 값을 double형으로 강제변형하여 double형 duration변수에 저장한후 출력해서 시간을 표현하고 있지만 이를 float형 변환후 float형 자료형에 저장하여도 double에 비해 전혀 손실값 없이 표현되고 있다. 이를 위해 float로 사용할 경우, 단일적으로 double의 8byte에 비해 공간절약이 4byte나 절약될수 있다.
- 각 시간이 정렬(Sort)방법의 정확한 시간계산을 의미하진 않는다. 이는 난수의 발생순서에 따라 정렬하는 시간의 차이도 나기 때문에 각 방법들의 효율성에 대해 알려줄뿐 정의할순 없다.

소감문 & 한계성

소감문 & 한계성

□ 소감문

각 정렬방법에 대한 알고리즘 개념은 이전 학습에서 정립하였고, 이번 과제의 키워드는 정렬방법에 따른 시간을 계산하는 것이다.

이를 통해 time.h에 정의된 clock_t, clock()을 살펴보고, 부가적으로 위 코드에서 구현된 ((double)(stop-start))/CLK_TCK를 (double)difftime(stop, start)/CLK_TCK, CLK_TCK는 MSDN에서 CLOCK_PER_SEC로 명칭이 바뀐 것, CLK_TCK는 1초에 100번의 clock ticks가 발생하므로, 이것으로 나뉘어야 초당 시간계산이 나오는 것을 알수있었다.

이번 과제를 통해 앞으로 구현하고 정의할수 있는 코드를 시간적으로 구하여 프로그램의 효율성에 적용할수 있다는 것을 알게됨으로써 조금더 나은 프로그램을 만들고자 할 수 있는 방향을 제시하였다.

□ 한계성

앞서 구현상의 한계에서 제시한 것처럼 시간적 계산이 각 구현된 함수의 정확한 시간을 정의하진 않는다. 단지 각 난수를 정렬하는 방법에 따른 시간차이로 시간상으로 볼 때 어떤 정렬방법이 빠르게 수행될수 있는지 확인할 뿐이므로, Best, Worst 방식으로 나뉜 빅오표현을 통한 이해가 더 좋은방법으로 통할수 있다.

부가적으로 위 코드에서 MAX_INT 32767로 정의하고 있는데 이는 2byte에 대한 한계치로 4byte로 정의되는 각 정렬에 쓰일 1차원배열에 MAX값으로는 작은 값이다. 다만 윈도우 운영체제에서 정적 메모리 할당으로 스택 영역이 약 1MB를 지원하고 있어 이상으로는 메모리 부족으로 프로그램 구현이 어렵기 때문에 그보다 2byte 낮은 ~32768 ~ 32767로 정의하고 있을 뿐이다.

더욱 큰 빅데이터 형식으로 동적 메모리 할당을 하여 사용할수 있는데 이의 한계치는 한 프로세서 내에 약 2GB를 할당할수 있다.

이를 앞으로 쓰일 프로그램 구현에 개념을 두고 사용하면 무분별한 메모리 사용을 줄일수 있을것이라 생각된다.

All Source Code

Sort Timing(1/4)

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

/* 2byte 표현값 -32768~32767으로 MAX, MIN 정의 */
/* 추가적으로, 각 정렬에 필요한 1차원 배열의 크기로 쓰이는 MAX, MIN 이므로 */
/* 컴파일러는 정적할당에 한계치를 0x7FFFFFFF 약 1MB(1,000,000)로 둔다 */
#define MAX_INT 32767
#define MIN_INT -32768

clock_t start, stop; // clock()에 의해 계산된 초당 시간주기(clock ticks)의 수를 나타낸다.
double duration;

/* 각 정렬에 쓰이는 배열 */
int bubbleData[MAX_INT];
int insertionData[MAX_INT];
int quickData[MAX_INT];

/* 정렬함수 정의 */
void bubbleSort(int*, int);
void insertionSort(int *, int);
void quickSort(int*, int, int);

/* 정렬에 대한 clock ticks를 구하는 함수 */
void sortTest(int*, int*, int*, int);

/* 난수구현 */
void createRandomNum(int*, int*, int*, int);

void createRandomNum(int* bubbleData, int* insertionData, int* quickData, int totalNum){
    int data;
    int i;
    srand((unsigned)time(NULL)); // 난수발생
    quickData[totalNum] = MAX_INT; // 최대값 대입

    for(i=0; i<totalNum; i++){ // totalNum 갯수만큼 난수 발생후, 각 정렬에 쓰일 변수에 저장
        data = rand() % RAND_MAX;
        bubbleData[i] = data;
        insertionData[i] = data;
        quickData[i] = data;
    }
}
```

Sort Timing(2/4)

/* 버블정렬 */

```
void bubbleSort(int *bubbleData, int totalNum){
    int count = totalNum -1;
    int temp;
    int i, j;

    for(i=0; i<count; i++)
        for(j=i+1; j<totalNum; j++){ // i+1 ~ totalNum-1 반복
            if(bubbleData[i]>bubbleData[j]){ // 좌, 우 비교후 좌측이 크면 Swap.
                temp=bubbleData[i];
                bubbleData[i]=bubbleData[j];
                bubbleData[j]=temp;
            }
        }
}
```

/* 삽입정렬 */

```
void insertionSort(int* insertionData, int totalNum)
{
    int i, j;
    int temp;
    for (i = 1; i<totalNum; i++ )
    {
        temp = insertionData[(j=i)]; // insertionData[1]값을 temp 저장, 첫 삽입정렬 비교 데이터
        /* 데이터값과 앞의 수를 비교해가면서 앞의 값이 클때 Swap 기능 */
        while (--j>=0 && temp<insertionData[j]) insertionData[j+1] = insertionData[j];
        /* 정렬이 끝난 후 삽입해야 될 위치에 첫 temp 저장 값을 넣는다. */
        insertionData[j+1] = temp;
    }
}
```

Sort Timing(3/4)

```
/* 퀵 정렬 */
void quickSort(int* quickData, int left, int right)
{
    /* Left_Num, Right_Num = 맨 왼쪽, 오른쪽 값을 유지하기 위한 변수 */
    int pivot, Left_Num, Right_Num;
    Left_Num = left;
    Right_Num = right;
    /* 첫 중간값은 맨 왼쪽 값으로 지정 */
    pivot = quickData[left];

    /* left, right 각각 오른쪽, 왼쪽으로 탐색해가면서 큰값을 찾아내는데, */
    /* 왼쪽, 오른쪽을 나타내는 값이 서로 뒤바뀔때, 즉 left값이 right값보다 더 오른쪽에 있을때까지 수행
    */
    while (left < right)
    {
        /* 피벗보다 작은 오른쪽값이 나타날때까지 검색 */
        while ((quickData[right] >= pivot) && (left < right)) right--;
        /* 왼쪽값에 오른쪽값을 저장 */
        if (left != right)
        {
            quickData[left] = quickData[right];
            left++;
        }
        /* 피벗보다 큰 왼쪽값이 나타날때까지 검색 */
        while ((quickData[left] <= pivot) && (left < right)) left++;
        /* 오른쪽값에 왼쪽값을 저장 */
        if (left != right)
        {
            quickData[right] = quickData[left];
            right--;
        }
    }
    /* 피벗을 재위치에 저장, quickData[left] 이전값들은 피벗보다 작다 */
    quickData[left] = pivot;
    /* 현재 pivot을 기준으로, left<pivot<right 의 데이터 크기를 가진다. */
    pivot = left;
    left = Left_Num;
    right = Right_Num;

    /* pivot을 기준으로 반으로 나뉜 left, right 데이터들을 재귀적으로 다시 정렬 */
    if (left < pivot)
        quickSort(quickData, left, pivot-1);
    if (right > pivot)
        quickSort(quickData, pivot+1, right);
}
```

Sort Timing(4/4)

```
/* 각 정렬별 시간주기 탐색 */
void sortTest(int* bubbleData, int* insertionData, int* quickData, int totalNum)
{
    printf("Number = %d\n", totalNum);
    createRandomNum(bubbleData, insertionData, quickData, totalNum);
    start = clock(); // star ~ stop = bubbleSort()의 시간주기 탐색
    bubbleSort(bubbleData, totalNum);
    stop = clock();
    duration = (((double)(stop-start))/CLK_TCK)*10;
    printf("Bubble Time : %f\n", duration);

    start = clock();
    insertionSort(insertionData, totalNum);
    stop = clock();
    duration = (((double)(stop-start))/CLK_TCK)*10;
    printf("insertion Time : %f\n", duration);

    start = clock();
    quickSort(quickData, 0, totalNum);
    stop = clock();
    duration = (((double)(stop-start))/CLK_TCK)*10;
    printf("quick Time : %f\n", duration);

    printf("\n");
}

int main(){

    /* 함수 끝에 해당하는 수의 난수를 발생하여, 각 정렬에 대한 시간주기 계산 */
    sortTest(bubbleData, insertionData, quickData, 1000);
    sortTest(bubbleData, insertionData, quickData, 5000);
    sortTest(bubbleData, insertionData, quickData, 10000);
    sortTest(bubbleData, insertionData, quickData, 15000);
    sortTest(bubbleData, insertionData, quickData, 20000);
    sortTest(bubbleData, insertionData, quickData, 25000);
    sortTest(bubbleData, insertionData, quickData, 30000);
    return 0;
}
```