**자료구조 실습 보고서**

**실습 8. 정 렬**

**2016년 5월 13일**

**학번: 201404051**

**이름: 정 용 석**

**1. 실습 문제 소개**

강의 자료를 바탕으로 5가지 정렬인 선택, 버블, 퀵, 삽입, 그리고 쉘 정렬 함수를 구현으로 시작한다. 실습 자료에 있는 소스코드 템플릿에 알맞은 형식으로 작성하여야 하며, 작성이 완료되면 메인 문을 실행하여 작동 여부를 확인한다.

모든 정렬 함수 작성이 완료되면, 각 정렬 함수의 성능을 측정, 분석할 잣대가 되는 데이터 비교 횟수 및 데이터 교환 횟수를 확인할 수 있도록 한다. 각 정렬 함수가 호출될 때 마다 비교, int nCompare, 그리고 교환, int nMove 변수의 횟수를 증가시키도록 한다. 성능 측정을 위해 데이터의 증가에 따른 총 비교 횟수와 교환 횟수를 비교할 수 있는 표와 그래프를 만들도록 한다. 데이터는 100부터 1000까지 100씩 증가하도록 한다.

**2. 소스 코드**

void swap(int \*a, int \*b)

{

int t = \*a;

\*a = \*b;

\*b = t;

nMove += 3; // 데이터 이동 카운터 증가

}

void selectionSort(int \*a, int n) {

int i, j;

int min;

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

min = i;

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

nCompare++; //데이터 비교 카운터 증가

if (a[j] < a[min])

min = j;

}

swap(&a[i], &a[min]);

}

}

void bubbleSort(int \*a, int n) {

int i, j;

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

{

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

nCompare++; // 데이터 비교 카운터 증가

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

swap(&a[j], &a[j + 1]);

}

}

}

int partition(int \*a, int begin, int end)

{

int p = begin;

int L = begin;

int R = end;

while (L < R)

{

while (a[L] <= a[p] && L < end){

nCompare++; // 데이터 비교 카운터 증가

L++;}

while (a[R] > a[p]) {

nCompare++; // 데이터 비교 카운터 증가

R--;}

if (L < R) {

nCompare++; // 데이터 비교 카운터 증가

swap(&a[L], &a[R]);

}

}

swap(&a[p], &a[R]);

return R;

}

void qSort(int \*a, int begin, int end)

{

int pivot;

if (begin < end)

{

nCompare++; // 데이터 비교 카운터 증가

pivot = partition(a, begin, end);

qSort(a, begin, pivot - 1);

qSort(a, pivot + 1, end);

}

}

void quickSort(int \*a, int n) {

qSort(a, 0, n - 1);

}

void insertionSort(int \*a, int n) {

int i;

int val;

int pos;

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

{

val = a[i];

for (pos = i; pos > 0; pos--) {

nCompare++; // 데이터 비교 카운터 증가

if (val < a[pos - 1]) {

a[pos] = a[pos - 1];

nMove++;

}

else

break;

}

a[pos] = val;

nMove++; // 데이터 이동 카운터 증가

}

}

void shellSort(int \*a, int n) {

int interval;

int i, pos, val;

for (interval = n / 2; interval > 0; interval = interval / 2)

{

for (i = interval; i < n; i++)

{

val = a[i];

for (pos = i; pos >= interval; pos -= interval)

{

nCompare++; // 데이터 비교 카운터 증가

if (val < a[pos - interval]) {

a[pos] = a[pos - interval];

nMove++; // 데이터 이동 카운터 증가

}

else

break;

}

a[pos] = val;

nMove++; // 데이터 이동 카운터 증가

}

}

}

void printSort(int n) //n: 1~5까지 1.선택, 2.버블, 3.퀵, 4.삽입, 5.쉘 정렬

{

switch (n) {

case 1:

printf("\t\tSelction Sort\t\t\n");

break;

case 2:

printf("\t\tBubble Sort\t\t\n");

break;

case 3:

printf("\t\tQuick Sort\t\t\n");

break;

case 4:

printf("\t\tInserton Sort\t\t\n");

break;

case 5:

printf("\t\tShell Sort\t\t\n");

break;

}

}

void Sort(int n, int\* work, int size) { //n: 1~5까지 1.선택, 2.버블, 3.퀵, 4.삽입, 5.쉘 정렬

switch (n) {

case 1:

selectionSort(work, size);

break;

case 2:

bubbleSort(work, size);

break;

case 3:

quickSort(work, size);

break;

case 4:

insertionSort(work, size);

break;

case 5:

shellSort(work, size);

break;

}

}

void checkFunction()

{

int \*data, \*work;

int size = 100;

int i;

for (i = 1; i <= 5; i++) //i: 1~5까지 1.선택, 2.버블, 3.퀵, 4.삽입, 5.쉘 정렬

{

printf(" -------------------------------------------\n");

printSort(i);

printf(" size \t compare move \tcomp+move\n");

printf(" ------\t---------- ----------\t----------\n");

while (size <= 1000) {

//size 100 ~ 1000, nCompare, nMove, nCompare+nMove 출력

data = (int\*)malloc(size\*sizeof(int));

work = (int\*)malloc(size\*sizeof(int));

makeRandomData(data, size);

memcpy(work, data, size\*sizeof(data));

Sort(i, work, size);

printf(" %5d\t%10d %10d \t%10d\n", size, nCompare, nMove, nCompare + nMove);

//증진 및 초기화

size += 100;

nCompare = 0;

nMove = 0;

free(work);

free(data);

}

printf(" -------------------------------------------\n");

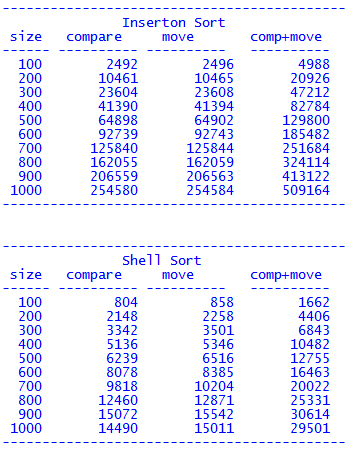
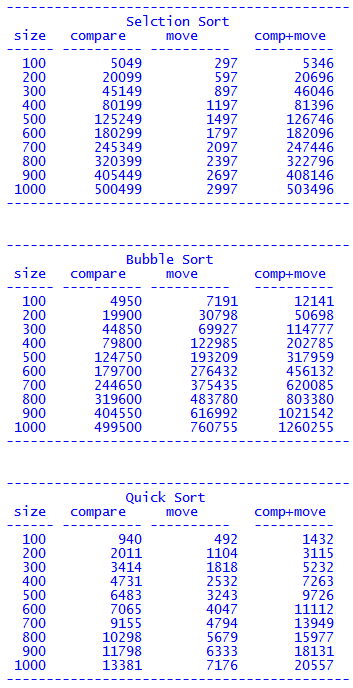
printf("\n\n");

size = 100;

}

}

**3. 테스트 결과**



**4. 작성자 코멘트**

실습 8-1 같은 경우는 솔직히 내가 한 게 없는 것 같다. 이미 수업 자료와 실습 자료 모두 각 정렬에 대한 알고리즘이 나와 있기 때문에 그냥 보면서 순조롭게 적은 것 밖에 한 것이 없다. 하나 복병이었다고 한다면 아마 퀵 정렬 부분에서 소스코드 템플릿과 알고리즘의 전달 인자가 조금 달라서 qSort라는 함수를 따로 만들어서 이에 맞추었다. 이 또한 TA와 함께 했기에 딱히 복병이라고 할 수도 없을 것 같다.

실습 8-2 또한 간단했던 것 같다. 배열의 교환 부분과 대입 부분을 각각의 변수를 선언하여 횟수를 세기만 하면 되는데, 대부분의 정렬 함수가 swap 함수를 가지고 있고, 이는 이미 실습 내용에 나와있기 때문에 추가적으로 각각의 횟수를 증가시켜야 하는 부분만 찾으면 됐다. 그리고 데이터 사이즈를 100부터 시작해서 1000까지 100씩 증가시키며 각각 비교와 교환 횟수 그리고 둘을 더한 횟수를 나타내는 표와 그래프를 만들기만 하면 되었다. 표를 만들기 위하여 함수를 통하여 선택부터 쉘 정렬까지 한번에 출력을 시켰고, 그래프 같은 경우는 엑셀의 힘을 빌렸다.

이미 우리는 선택, 버블, 퀵, 삽입, 그리고 쉘 정렬이 각각 O(n^2), O(n^2), O(nlog(n)), O(n^2), O((nlog(n))^2)의 평균 시간 복잡도를 가지는 것을 알 수 있다(쉘 정렬의 경우 인터넷을 이용했다). 그리고 표를 분석해 보면, 데이터 크기가 증가함에 따라서 각각의 정렬이 가지는 총 비교, 교환 횟수가 각각의 시간 복잡도의 모습을 보이는 것을 알 수 있다. 이를 더 편히 보려면 그래프를 보면 된다. 언급했듯이 선택, 버블, 그리고 삽입 정렬은 모두 n^2의 그래프 성향을 보인다. 조금 놀란 것이 물론 계수를 떼고 그래프 성향만 본 것이지만 버블 정렬과 선택, 삽입 정렬의 차이가 생각했던 것보다 훨씬 커 보였다. 그리고 반대로 nlogn의 성향을 가지는 퀵과 쉘 정렬 같은 경우는 차이가 거의 미미해 보였다. 물론 데이터 크기가 커지면서 비교와 교환 횟수가 상대적으로 늘어나고, 이를 다른 그래프에 다른 정렬 알고리즘과 비교해 보았다 하더라도 확실히 차이점을 볼 수 있었다.