**Computer Algorithm**

**Chapter 05**

**QuickSort**

**[실습] Insert Sort와 Quick Sort 구현**

**실습 목표**

* **InsertSort와 QuickSort의 동작방식을 이해할 수 있다.**
* **정렬 알고리즘의 종류와 시간복잡도에 대해 학습한다.**
* **QuickSort의 최악복잡도가 어떠한 경우인지를 동작방식으로부터 알 수 있다.**

**요구사항**

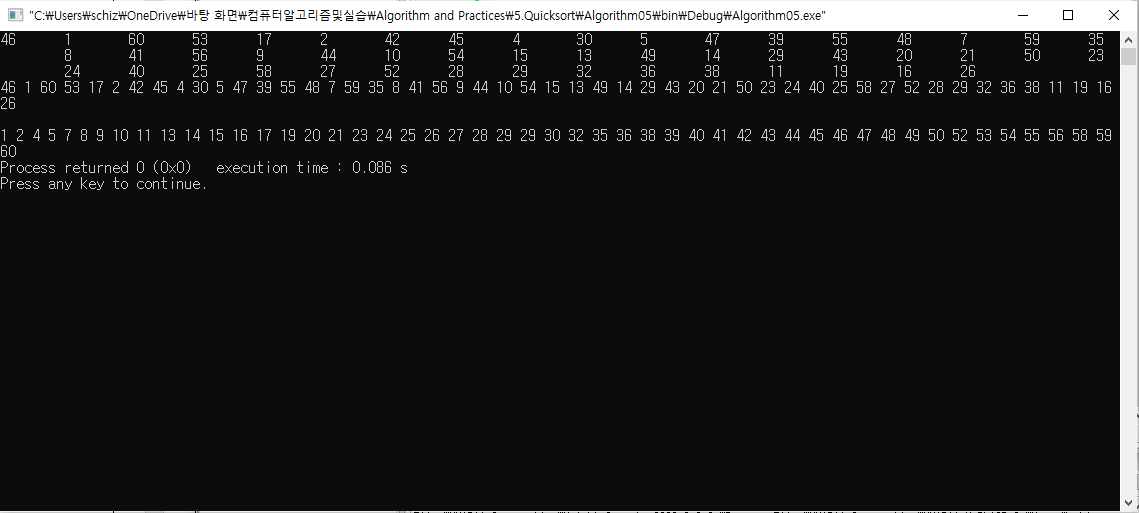
* **실습과제 1) (20점)**

Insert Sort(삽입정렬)를 구현한다.

입력은 파일에서 받는 것으로 하며 파일이름은 input.txt 로 한다. 또한 파일의 내용은 아래와 같으며 숫자들은 탭으로 구분된다.

46 1 60 53 17 2 42 45 4 30 5 47 39 55 48 7 59 35 8 41 56 9 44 10 54 15 13 49 14 29 43 20 21 50 23 24 40 25 58 27 52 28 29 32 36 38 11 19 16 26

출력은 화면에 오름차순으로 정렬된 숫자들을 보이도록 한다.



* **실습과제 2) (35점)**

Partition 알고리즘을 사용하여 정수를 정렬하는 QuickSort 알고리즘을 구현한다.

\* 실습과제 1과 동일한 입력을 이용한다.

* **실습과제 3) (35점)**

Randomized QuickSort를 구현한다.

(Randomized Quick sort는 pivot을 정할 때 할당된 배열 중에 랜 덤하게 pivot을 정해 가장 왼쪽에 위치한 수와 swap을 한 후 partition을 진행하는 것을 말한다. 실습과제 1의 quick sort는 가 장 좌측의 있는 수를 pivot으로 설정함)

* **실습과제 4) (10점)**

일반 quick sort 알고리즘과 randomized quick sort, insertion sort 알고리즘의 성능을 비교하는 코드를 작성한다.

InsertSort, QuickSort, Randomized QuickSort을 모두 실행하여 각 알고리즘이 비교연산을 몇 번씩 수행하였나를 출력하도록 한다. 이 경우 화면에는 정렬된 숫자를 보이지 않아도 된다. 비교연산 의 횟수는 비교가 일어나는 곳 (두 값을 비교한 명령어 a < b)의 다음에 counter 변수를 두고 하나씩 증가시킨다.

**배경지식**

정렬은 주어진 숫자들을 오름차순 또는 내림차순으로 위치시키는 것을 말한다. 대표적인 정렬 방법으로는 삽입정렬 (insert sort), 버블정렬, 퀵정렬, 머지정렬 등이 있으며 각각은 서로 다른 방법으로 숫자들을 배치한다.

본 실습에서는 삽입정렬, 퀵정렬을 구현하도록 하며 각 알고리즘의 성능을 비교하도록 한다.

1)

삽입정렬의 알고리즘은 다음과 같다.

insertion\_sort(A[], n) 🡨 A[1 ... n]을 정렬한다.

{

int i,j,tmp;

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

{

tmp = A[i];

// A[i]가 삽입될 자리를 찾음. 삽입될 자리는 j.

for(j = i; A[j-1] > tmp && j > 0; j--)

{

... // A[j]에 값을 할당해 줌. 어떤 값을 할당해 주는 가 하는 부분은 아래 그림을 참고해서 작성할 것.

}

A[j] = tmp; // tmp의 값을 A[j]에 할당

}

}

위 알고리즘의 동작 예는 아래와 같다.



2~4)

Quick sort는 divide-and-conquer를 사용한 정렬방법으로서 재귀호출을 중심으로 재귀 호출의 각 단계별로 pivot이라는 수를 중심으로 partition을 반복적으로 수행함에 의하여 정렬을 수행하는 알고리즘이다.

Quick sort의 알고리즘은 다음과 같이 기술된다.

void quickSort(A[], p, r) 🡨 A[p ... r]을 정렬한다

{

**if** (p < r) **then** {

q = partition(A, p, r);  // partition() 수행

quickSort(A, p, q-1);

// 좌측 partition에 대해 quick sort 재귀호출

quickSort(A, q+1, r);

// 우측 partition에 대해 quick sort 재귀 호출

        }

}

Quick sort을 구현할 때 가장 중요한 것은 partition() 함수를 구현하는 것이다. partition알고리즘을 옳게 구현했다면 나머지는 위의 pseudo code를 C언어 코드로 바로 구현할 수 있다.

partition()은 주어진 배열에서 임의의 pivot (일반적으로 가장 오른쪽 수나 또는 가장 왼쪽에 위치한 수)을 중심으로 pivot보다 작은 값은 pivot보다 왼쪽에 pivot보다 큰 값은 pivot보다 오른 쪽에 위치시키는 작업을 한다. 즉 partition()은 아래 그림과 같이 pivot을 중심으로 숫자들을 재 배치하는 역할을 한다.

<원 배열>

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 4 | 7 | 6 | 1 | 6 | 3 | 8 |

⇓ partition() : pivot이 5일 때

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 1 | 5 | 7 | 6 | 6 | 8 |

*ai*< 5 *ai*>5

partition()은 여러 형태로 구현될 수 있으며 그 중 한 방법의 pseudo code는 아래와 같다.

int partition(A[], p, r) 🡨 A[p ... r]을 파티션한다.

{

pivot = A[p] // 파티션의 처음 값인 A[p]를 pivot으로 한다.

h = p

for(k = p + 1; k <= r; k++)

if (A[k] < pivot) {

h = h + 1

swap(A[h], A[k])

}

swap(A[p], A[h])

return h   // h의 위치를 리턴한다.

}

입력 배열 [5 4 7 6 1 6 3 8]에 대하여 위의 pseudo code에 의한 동작 예는 아래 그림과 같다.

키보드이(가) 표시된 사진

자동 생성된 설명

화면, 방이(가) 표시된 사진

자동 생성된 설명

Randomized Quick Sort는 pivot값의 위치를 랜덤한 값으로 정하는 것으로, 이전 실습에서의 pivot의 위치는 입력 파티션의 가장 앞의 위치로 설정하였던 데 반해서, Randomized Quick Sort는 랜덤한 값을 구해서 그 값을 pivot의 위치로 설정하는 방식이다. 랜덤한 값을 피봇의 위치로 잡기 때문에 데이터 값의 분포 특성에 영향을 받지 않고 랜덤하게 파티션이 이루어 질 수 있어 확률적으로 안정적인 정렬 성능을 보여준다.

Randomized Quick Sort에서는 전체적인 내용은 일반 Quick Sort와 비슷하게 구성되나 다만 pivot의 값이 주어진 파티션의 처음 원소(A[p])로 잡는 일반 Quick Sort 알고리즘과는 달리 파티션의 pivot의 값은 A[p]와 A[r] 사이의 임의의 값 (r과 p사이에 랜덤으로 선택된 위치에 저장된 값)으로 하며 이 임의의 값은 A[p]의 값과 바뀌지며 알고리즘이 시작된다.

**제출방법**

* 보고서 작성방법: 실습문제 번호별로 결과가 나온 화면의 내용을

캡쳐하여 보고서에 붙여 놓는다.

* 소스코드의 파일이름에 연습문제 번호를 붙이는 것을 잊지 않는다. 예) ex-1.c, ex-2.c
* 결과 보고서에 이름과 작성 날짜를 기입하는 것을 잊지 않는다. 예) 김웅섭\_2020\_09\_01.doc
* 실행결과를 보고서에 작성하여 소스코드와 함께 제출한다.
* 제출 마감 : e-class 제출 마감시간까지