2020년도 2학기 컴퓨터공학설계및실험I
X주차 정렬 알고리즘 예비보고서

20201234 홍길동

1. 실습 목적

(해당 주차의 실습 목적이 무엇인지 명료하게 작성)

정렬 알고리즘 중 거품 정렬, 삽입 정렬, 퀵 정렬을 구현하고 다양한 입력 데이터로 실행해보며 소요 시간을 비교한다.

2. 관련 이론

(해당 주차의 실습에 대한 관련 이론 지식을 서술)

정렬 알고리즘은 탐색 알고리즘과 함께 알고리즘의 가장 기본적인 주제로써, 문제는 간단하지만 다양한 해결 방법이 존재하기 때문에 효율적으로 이 문제를 해결하기 위한 연구가 많이 진행되어 왔다. 아래의 내용은 리스트를 오름차순으로 정렬한다고 가정하고 작성하였다.

정렬 알고리즘에서 가장 먼저 고안된 방법은 거품 정렬(Bubble Sort)이다.[1] 거품 정렬은 주어진 리스트를 처음부터 끝까지 탐색하면서, 인접한 두 원소를 비교하며 정렬하는 방법이다. 거품 정렬의 C-like 의사 코드(Pseudo Code)는 아래 **[그림 1]**과 같다. swap(a, b, temp)의 의미는 a와 b의 값을 바꾼다는 뜻이며, temp는 swap 과정에서 데이터를 임시로 저장하는 변수이다.

```
void BubbleSort (int list[], int n) {
    for (int i=n-1; i>0; i--) {
        for (int j=0; j<i; j++)
            if(list[j+1] < list[j])
                swap(list[j+1], list[j], temp);
    }
}
```

그림 1. 거품 정렬의 의사 코드

거품 정렬은 n개의 데이터에 대해 항상 모든 원소를 비교하기 때문에 최악의 경우 swap 연산의 횟수는 n(n-1)/2가 된다. 따라서 거품 정렬의 시간 복잡도는 $O(n^2)$이다.

삽입 정렬(Insertion Sort)은 전체 리스트를 정렬된 부분과 정렬되지 않는 부분으로 나누고, 정렬되지 않은 부분에서 원소를 하나씩 가져와 정렬된 부분의 적절한 위치에 삽입하여 정렬하는 방식이다.[2] 삽입 정렬의 C-like 의사 코드는 아래 **[그림 2]**와 같다.

```
void InsertionSort (int list[], int n) {
    for (int i=1; i<n; i++) {
        next = list[i];
        for (int j=i-1; j>=0 && list[j]>next; j--)
            list[j+1] = list[j];
        list[j+1] = next;
    }
}
```

그림 2. 삽입 정렬의 의사 코드

삽입 정렬은 주어진 리스트가 역순으로 정렬되어 있을 때가 최악의 경우가 되고, 이 때의 연산 횟수는 n(n-1)/2 + 2(n-1)으로써 시간 복잡도는 거품 정렬과 마찬가지로 $O(n^2)$이다. 삽입 정렬은 선택 정렬보다 성능이 좋다고 알려져 있으나, 선택 정렬이 최적으로 구현될 경우에는 삽입 정렬보다 뛰어난 성능을 발휘하기도 한다.[3]

퀵 정렬(Quick Sort)은 찰스 앤터니 리처드 호어가 개발한 정렬 알고리즘으로써[4], 다른 원소와의 비교만으로 정렬을 수행하는 특이한 성질을 가지고 있다. 분할 정복 방법(Divide and Conquer)에 기반하였으며 리스트를 피벗(Pivot)을 기준으로 절반으로 쪼개고, 이 과정을 쪼갠 리스트의 크기가 0이나 1이 될 때까지 반복한다. 이렇게 끝까지 쪼갠 후에는 쪼갠 부분들끼리 정렬을 시킨 다음, 마지막으로 정렬된 부분을 합침으로써 정렬을 완성하는 방식이다. 퀵 정렬의 C-like 의사 코드는 아래 **[그림 3]**과 같다.

```
void QuickSort (int list[], int left, int right) {
    if(left<right) {
        pivot = partition(list, left, right);
        QuickSort(list, left, pivot-1);
        QuickSort(list, pivot+1, right);
    }
}
```

그림 3. 퀵 정렬의 의사 코드

퀵 정렬 의사 코드에서 partition 함수의 의사 코드는 아래 **[그림 4]**와 같다.

```
int partition (int list[], int left, int right) {
    low = left;
    high = right + 1;
    pivot = list[left];

    do {
        do {
            low++;
        } while (low<=right && list[low]<pivot);
        do {
            high--;
        } while (high>=left && list[high]>pivot);
        if (low < high)
            swap(list[low], list[high], temp);
    } while (low < high);

    swap(list[left], list[high], temp);
    return high;
}
```

그림 4. partition 함수의 의사 코드

퀵 정렬은 일반적인 상황에서 리스트를 절반씩 나누고 다시 합치는 과정을 거치기 때문에 시간 복잡도는 $O(n\log n)$이지만, 만약 피벗이 항상 최솟값이나 최댓값으로 잡게 되는 경우에는 다른 정렬 알고리즘과 같이 $O(n^2)$의 시간 복잡도를 갖게 된다.[5] 피벗을 어떻게 잡는지에 따라 퀵 정렬의 성능 차이가 크기 때문에, 신중하게 선택하는 것이 필요하다. 본 실습에서는 리스트 왼쪽 첫 번째 원소를 피벗으로 선택할 계획이다.

3. 실습 방법

(실습 방법과 과정을 자세하게 서술)

실습 방법은 거품 정렬, 삽입 정렬, 퀵 정렬을 모두 C언어로 구현한 뒤, 여러 데이터 리스트로 정렬을 수행하여 각각의 소요 시간을 비교하도록 한다. 사용할 데이터 리스트는 완전 무작위 정수가 나열된 리스트와 대부분 정렬이 된 리스트, 그리고 역순으로 정렬된 리스트를 사용하도록 한다. 또한 데이터 집합의 크기에 따른 알고리즘의 효율성도 비교하기 위해, 같은 형태의 데이터 리스트를 여러 크기로 나누어서 수행하고, 그 결과를 정리하도록 한다.

4. 기타

(실습에 관련되어 본인이 조사한 내용을 자유롭게 서술)

주어진 데이터에서 동일한 값이 존재하는 경우, 정렬하기 전 그 값들의 순서가 정렬이 된 후에도 변하지 않는다면 그 정렬 알고리즘을 안정적인 정렬(Stable Sort)이라 한다.[6] 예를 들어, 주어진 데이터가 3 1 2 1 4 라고 가정해보자. 첫 번째로 나타난 1을 1(a)로 하고, 두 번째로 나타난 1을 1(b)로 하면, 주어진 데이터를 3 1(a) 2 1(b) 4로 표현할 수 있다. 만약 어떤 정렬 알고리즘을 수행했을 때, 1(a) 1(b) 2 3 4 와 같이 정렬 전 1의 순서가 정렬 후에도 변하지 않는 것이 보장되면 안정적인 정렬이라고 할 수 있다.

이번 주차 실습의 거품 정렬과 삽입 정렬은 안정적인 정렬이지만, 퀵 정렬은 안정적인 정렬이 아니다.

5. 참고 문헌

(보고서를 작성하면서 참고했던 책, 논문 등을 순서대로 작성)

참고문헌 작성 요령

저널 논문 : 저자명, 제목, 논문지명(이탤릭체), 권(Volumn), 호(Number), 쪽(Page), 출판년도

학술대회 논문 : 저자명, 제목, 학술대회명(이탤릭체), 쪽(Page), 개최장소, 국가, 년월

도서 : 저자명, 도서명, 출판사명(이탤릭체), 출판년도

[1] E. H. Friend, “Sorting on Electronic Computer Systems”, *Journal of the ACM*, Vol. 3, no. 3, pp. 134-168, 1956.

[2] D. E. Knuth, “The Art of Computer Programming (Volume 3) : Sorting and Searching”, *Addison-Wesley Professional*, 1998.

[3] S. Jadoon, S. F. Solehria, and M. Qayum, “Optimized Selection Sort Algorithm is faster than Insertion Sort Algorithm : a Comparative Study”, *International Journal of Electrical & Computer Sciences IJECS-IJENS*, Vol. 11, no. 2, pp. 19-24, 2011.

[4] C. A. Hoare, “Quicksort”, *The Computer Journal*, Vol. 5, no. 1, pp. 10-16, 1962.

[5] W. Xiang, “Analysis of the Time Complexity of Quick Sort Algorithm”, *In 2011 International Conference on Information Management, Innovation Management and Industrial Engineering*, pp. 408-410, Shenzhen, China, November 2011.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, “Introduction to Algorithms”, *MIT Press*, 2009.