

자료구조 실습 보고서

숙제명: [제13주]이진탐색트리

제출일: 2018.05.29

학번/이름: 201404377 / 진승언

<프로그램 설명서>

먼저 이번과제에서는 4가지의 정렬방법에 대해 코딩을 하였다. 각 정렬 방법에 대해 알아보면 다음과 같다.

1. 삽입 정렬

=> 삽입정렬은 만약 자료가 모두 정렬되어있는 경우 외부 루프는 $n-1$ 번만 실행되고 각 단계에서 이동 없이 1번의 비교만 이루어지므로 총 비교 횟수는 $n-1$ 번이 되어 알고리즘의 시간 복잡도는 $O(n)$ 이다. 그리고 최악의 경우는 자료가 역순으로 정렬되어있는 경우로서 각 단계에서 앞에 놓인 자료들을 전부 한 칸씩 뒤로 이동하여야 한다. 최악의 경우 외부 루프 안의 각 반복마다 1번의 비교가 수행되고 총 이동 횟수는 외부 루프의 각 단계마다 $I+2$ 번의 이동이 이루어지므로 최악의 시간복잡도는 $O(n^2)$ 이다. 평균시간복잡도는 빅오로 하니 이것이 평균시간복잡도이다. 삽입 정렬의 장점은 자료가 정렬이 되어있는 경우는 빠르다는 점과 구현이 비교적 쉽다는 것이다. 단점은 레코드 수가 많을 경우 효율이 떨어진다는 점이 있다. 느낀점은 레코드의 수가 적을 경우는 간단하게 이 정렬 알고리즘을 쓰는것도 괜찮다고 느꼈다.

2. 퀵 정렬

=> 퀵 정렬은 만약 리스트의 분할이 균형있게(2분의1로) 나누어 질 때이다. 이 경우 $\log n$ 개의 패스가 필요하고 각각의 패스에서는 전체 리스트의 대부분의 레코드를 비교해야 하므로 평균 n 번 정도의 비교가 이루어지므로 퀵 정렬은 비교 연산을 총 $n \log n$ 번 실행하게 되어 $O(n \log n)$ 의 복잡도를 가지는 알고리즘이 된다. 여기서 레코드의 이동 횟수는 비교 횟수보다 적으므로 무시할 수 있다. 최악의 경우의 복잡도는 하나의 리스트와 나머지리스트로 불균형하게 계속 쪼개질 때이다. 이 경우 레코드의 수만큼 총 n 번의 패스가 실행되고, 각 패스에서 n 번의 비교가 이루어지게 되므로 비교 연산을 n^2 번 실행하게 된다. 즉 퀵 정렬은 최악의 경우 $O(n^2)$ 의 시간 복잡도를 가지게된다. 그럼에도 불구하고 퀵 정렬은 평균적인 경우의 시간 복잡도가 $O(n \log n)$ 으로 나타난다.

퀵정렬은 속도가 빠르고 추가 메모리 공간을 필요로 하지 않는 등의 장점이 있는 반면에 정렬된 리스트에 대해서는 오히려 수행시간이 더많이

결린다는 단점이 있다. 느낀점은 레코드수가 많고 좀 좋은 복잡도를 쓰고 싶을때는 퀵 정렬을 쓰면 좋겠다고 느꼈다.

3. 거품 정렬

=> 버블 정렬은 인접한 2개의 레코드를 비교하여 크기가 순서대로 되어 있지 않으면 서로 교환하는 비교-교환 과정을 리스트의 왼쪽 끝에서 시작하여 오른쪽 끝까지 진행하는 정렬 방법이다. 이 정렬의 비교횟수는 최상, 평균, 최악의 어떠한 경우에도 항상 일정하고 $O(n^2)$ 이다. 이동 횟수는 최악의 이동 횟수는 입력 자료가 역순으로 정렬되어 있는 경우에 발생하고 그 횟수는 비교 연산의 횟수에 3을 곱한 값이다. 왜냐하면 하나의 swap 함수가 3개의 이동을 포함하고 있기 때문이다. 최상의 경우는 입력 자료가 이미 정렬이 되어 있는 ruddnelk. 이런 경우는 자료 이동이 한 번도 발생하지 않는다. 평균적인 경우에는 자료 이동이 0번에서 1번까지 같은 확률로 일어날 것이다. 따라서 이를 기반으로 계산하여 보면 $O(n^2)$ 의 알고리즘임을 알 수 있다.(평균, 최악 모두 $O(n^2)$ 이다)

4. 힙 정렬

=> 힙은 우선 순위 큐를 완전 이진 트리로 구현하는 방법으로 힙은 최댓값이나 최솟값을 쉽게 추출할 수 있는 자료구조이다. 힙의 종류로는 최소 힙과 최대 힙이 있는데 이번 과제에서는 루트 값이 최솟값 이이고 그 밑으로 루트값보다 다 작은값을 가지므로 최소 힙을 구현한 것이다. 먼저 downheap과정에서 한 노드에 대해 $\log n$ 만큼 내려갈 수 있고 내려가면서 2번의 비교를 진행하므로 $2\log n$ 번이 필요하다. 즉 n 개의 데이터에 대해 $2\log n$ 번의 비교를 진행하면서 즉 $2n\log n$ 번의 비교연산을 진행하게 된다. 즉, 힙 정렬의 시간 복잡도는 $2n\log n + n$ 이다. 빅이로 표현하면 $O(n\log n)$ 이다. 최악의 경우도 같은 시간복잡도를 가진다. 힙정렬의 장점은 부가적인 메모리가 전혀 필요없고 퀵정렬과 비교했을 때 어떤 입력 데이터에 대해서도 최대복잡도가 $O(n\log n)$ 이라는 것이 보장된다는 점이 있다. 단점으로는 실제로 프로그램에서 실행 할 경우 평균적인 계산 속도에서는 퀵 정렬 보다 늦다는 단점이 있다.

이번 과제에서는 힙정렬을 직접 구현해보았는데 다음과 같이 구현하였다.

```
public void adjust(int[] newData, int root, int n) {
    int child, rootkey;
    int temp = newData[root]; // 임시 루트값
    rootkey = newData[root]; // 루트키값
    child = 2 * root; // 처음에 8
    while (child <= n) {

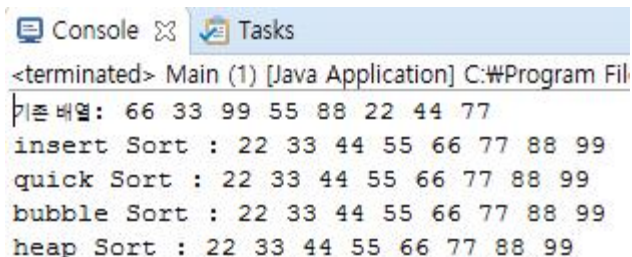
        if (child < n && newData[child] < newData[child + 1]) { //오른쪽 자식이있고 왼쪽자식이 오른쪽 자식보다 작을때
            temp = newData[++child];
        } else { // 왼쪽 자식밖에 없거나 (자수가1) 왼쪽자식이 오른쪽 자식보다 클 때
            temp = newData[child];
        }

        if (rootkey < temp) { //rootkey값과 child값을 비교해서 child값이 더 큰 경우 (교환)
            int tmp2 = newData[child / 2]; //tmp2에 자식의 부모값을 저장
            newData[child / 2] = newData[child]; // 자식의 부모값에 자식의 값을 저장
            newData[child] = tmp2; //자식 값은 자식의 부모값을 저장
        }
        child = child * 2; //child를 부모의 값으로 바꿔줌
    }
}
```

먼저 자식 값이 n보다 작을 때까지 반복된다. while문 안에서는 오른쪽 자식이있고 왼쪽 자식이 오른쪽 자식보다 작을 때 temp에 오른쪽 자식 값을 저장한다. 그리고 아닌 경우는(왼쪽 자식밖에 없거나 왼쪽 자식이 오른쪽 자식보다 클 때) 왼쪽 자식값을 temp에 저장해준다.

그 후 , rootkey값이 temp보다(temp에 현재 왼쪽,오른쪽 자식값중 더 큰 값이 저장되었다) 작다면 위에서 temp에 저장된 해당 자식값과 부모 노드값을 swap해주면 된다. 그 후 child를 2를 곱해주어 child의 상위값 즉 부모노드를 child로 해주어 반복을 돌려준다

<실행 결과>



```
<terminated> Main (1) [Java Application] C:\Program Fil
기존 배열: 66 33 99 55 88 22 44 77
insert Sort : 22 33 44 55 66 77 88 99
quick Sort : 22 33 44 55 66 77 88 99
bubble Sort : 22 33 44 55 66 77 88 99
heap Sort : 22 33 44 55 66 77 88 99
```

오름차순으로 정렬이 잘 되었음을 볼 수 있다.