알고리즘

- 01. Sort (Bubble, Selection, Insertion) -

※ 과제 목표 및 해결 방법

- 이중 연결 리스트로 구현.
- Bubble, Selection, Insertion Sort를 구현하여 입출력 파일 방식으로 결과를 도출하여라.

※ 주요 부분 코드 설명 (알고리즘 부분 코드 캡쳐)

```
public class Bubble_Sort {
    public Object Bubble(DoublyLinkedList bubble_list) {
        for(int i = 0; i < bubble_list.size(); i++) {
            for (int j = 0; j < bubble_list.size()-1; j++) {
                if((int)bubble_list.get(j) > (int)bubble_list.get(j+1)) {
                    bubble_list.add(j+2, bubble_list.get(j));
                    bubble_list.remove(j);
                }
            }
        }
        return bubble_list;
    }

    public void print(DoublyLinkedList bubble_list) {
        for (int i = 0; i < bubble_list.size(); i++) {
            System.out.println(bubble_list.get(i));
        }
    }

}
```

[그림1. Bubble_Sort]

➔ 버블 정렬은 매번 연속된 두 개의 값을 비교하여, 정한 기준의 값을 뒤로 넘기며 정렬하는 방법이다. 오른차순으로 정렬하고자 할 경우, 비교시마다 큰 값이 뒤로 이동하여, 1바퀴를 돌고 오면 가장 큰 값이 맨 뒤에 저장된다.
맨 마지막에는 비교한느 수들 중 가장 큰 값이 저장 되기 때문에, (전체 데이터의 크기 – 현재까지 순환한 바퀴 수)만큼만 반복하면 된다.
① 처음 값과 다음 값의 크기를 비교를 한다.
② 조건문이 성립하면 큰 값을 뒤로 보내기 위해 j+2의 위치에 j의 값을 추가해준다.
③ j위치에 있는 값을 추가해주었기 때문에 삭제를 해준다.
④ j의 값이 (리스트 크기 – 1)까지 반복하면 다시 처음부터 비교를 한다.
⑤ 모든 정렬을 마치면 bubble_list를 반환해준다.

```
public class Selection_Sort {
    public Object Selection(DoublyLinkedList select_list) {
        int min_value;

        for(int i = 0; i < select_list.size()-1; i++) {
            min_value = i;
            for(int j = i+1; j < select_list.size(); j++) {
                if((int) select_list.get(min_value) > (int) select_list.get(j)) {
                    min_value = j;
                }
            }
            select_list.add(i, select_list.get(min_value));
            select_list.remove(min_value+1);
        }
        return select_list;
    }

    public void print(DoublyLinkedList select_list) {
        for (int i = 0; i < select_list.size(); i++) {
            System.out.println(select_list.get(i));
        }
    }
}
```

[그림2. Selection_Sort]

➔ 선택 정렬은 현재 위치에 들어갈 값을 찾아 정렬하는 것이다. 현재 위치에 저장 될 값의 크기가 작냐, 크냐에 따라 최소 선택 정렬과 최대 선택 정렬로 구분 할 수 있다. 전자는 오름차순으로 정렬될 것이고, 후자는 내림차순으로 정렬될 것이다.
① 첫 번째에 위치한 값을 최소값으로 지정한다.
② 지정한 최소값과 리스트의 값들을 비교해서 지정한 최소값이 크면 현재의 j값을 최소값으로 변경해준다.
③ 변경한 최소값을 I의 위치에 추가해준다.
④ 추가를 한 후 기존의 위치해 있던 값을 삭제해준다.
⑤ 모든 정렬을 마치면 select_list를 반환해준다.

```java
public class Insertion_Sort {
    public Object Insertion(DoublyLinkedList insert_list) {
        for (int i = 1; i < insert_list.size(); i++) {
            int standard_value = (int) insert_list.get(i);
            int a = i-1;

            while (standard_value < (int) insert_list.get(a) && a >= 0 ) {
                insert_list.add(a, standard_value);
                insert_list.remove(a+2);
                a--;
            }
        }
        return insert_list;
    }

    public void print(DoublyLinkedList insert_list) {
        for (int i = 0; i < insert_list.size(); i++) {
            System.out.println(insert_list.get(i));
        }
    }
}
```

[그림3. Insertion_Sort]

➔ 삽입 정렬은 현재 위치에서, 그 이하의 배열들을 비겨하여 자신이 들어갈 위치를 찾아, 그 위치에 삽입하는 정렬이다.
① 삽입 정렬은 두 번째부터 시작한다. 현재의 값을 저장을 해준다. 비교할 값의 위치를 (현재 – 1)로 설정한다.
② 현재의 값과 비교할 값을 비교한다.
③ 현재의 값이 더 작으면 a의 위치에 현재의 값을 추가해준다.
④ 추가한 후 a+2의 위치의 값을 삭제해준다.
⑤ a의 값을 1감소해준다. a의 값이 0보다 작아지면 반복문을 종료한다.
⑥ 모든 정렬을 마치면 insert_list를 반환해준다.

```java
public static void main(String[] args) {
    DoublyLinkedList bubble_list = new DoublyLinkedList();
    Bubble_Sort bubble_sort = new Bubble_Sort();

    try {
        // 파일 객체 생성
        File read_file = new File("C:\\Users\\ssey0\\OneDrive\\바탕 화면\\3학년 2학기\\알고리즘\"
        // 입력 스트림 생성
        FileReader filereader = new FileReader(read_file);
        BufferedReader bufReader = new BufferedReader(filereader);
        String line = "";
        int line_num = 1;

        line = bufReader.readLine();
        bubble_list.addFirst(Integer.parseInt(line));

        while((line = bufReader.readLine()) != null) {
            bubble_list.add(line_num, Integer.parseInt(line));
            line_num++;
        }

        bufReader.close();

        long start = System.currentTimeMillis(); //시작하는 시점 계산

        bubble_sort.Bubble(bubble_list); // 실행시간을 측정하고싶은 코드

        long end = System.currentTimeMillis(); //프로그램이 끝나는 시점 계산

        File writer_file = new File("C:\\Users\\ssey0\\OneDrive\\바탕 화면\\3학년 2학기\\\
        FileWriter filewriter = new FileWriter(writer_file);
        BufferedWriter bufWriter = new BufferedWriter(filewriter);

        for(int i=0; i < bubble_list.size(); i++) {
            bufWriter.write(String.valueOf(bubble_list.get(i)));
            bufWriter.newLine();
        }
        bufWriter.write("실행 시간 : " + ( end - start )/1000.0 + "초");
        bufWriter.close();

    }catch (IOException e) {
        System.out.println(e);
    }
```

[그림4. Main]

① 각각의 test 케이스의 파일을 한 줄씩 읽으면서 이중연결 리스트를 만들었다. **(100개, 1000개, 10000개의 데이터 값을 변경해줘야 한다.)**
② 각 정렬마다 리스트를 생성하면 시간 측정을 위한 함수를 실행하고 각 정렬을 실행한다.
③ 실행을 마치고 txt파일로 정렬한 값을 파일에 입력해준다.
④ 정렬한 값의 입력을 마치고 마지막으로 실행시간을 입력해주고 파일을 닫는다.

※ 실행 결과(시간 복잡도 포함)

버블, 삽입, 선택의 결과를 분석한 결과 100개, 1000개의 데이터에서는 정렬 시간에는 별 차이가 보이지 않는 것을 확인 할 수 있었다. 그러나 10000개의 데이터를 정렬할 경우 비교적 버블이 가장 많은 시간이 소요되었다. 삽입과 선택 두 개의 정렬을 비교했을 때를 보면 삽입 정렬이 좀 더 빠르다는 것을 알 수 있었습니다. 3개의 방식 최악의 경우 모두 시간복잡도는 O(n^2)이지만, 삽입 정렬의 경우 이미 정렬되어 있는 경우 한 번씩 밖에 비교하지 않아 시간복잡도는 O(n)을 가진다. 하지만 이 테스트 케이스에서는 이미 정렬이 되어 있는 것은 아니라서 O(n)보다는 좀 더 많은 시간복잡도를 가지지 않았을까 생각이 든다.

버블

[그림5. Bubble_result]

선택

[그림6. Select_result]

삽입

[그림7. Insert_result]