알고리즘 실습(1주차)

201000287 일어일문학과 유다훈

1. 과제 설명 및 해결 방법

1. 외부파일로부터 무작위 순서의 데이터값을 10개, 100개, 1000개, 10000개를 입력 받는다.
2. 해당 데이터를 Linked List를 생성하고, 노드 1개당 데이터 1개씩 삽입한다.
3. 삽입 정렬을 이용하여 Linked List를 오름차순으로 정렬한다
4. 오름차순으로 정렬된 데이터 값들을 외부 파일로 출력한다.
5. 삽입 정렬을 할 때 걸린 시간을 측정한다.

   삽입 정렬 시에 시간 측정은 milisecond 단위로 측정한다.

2. 주요 부분 코드 설명(알고리즘 부분)

1. Linked List 생성

```
while((buffer = br2.readLine()) != null) {
    Node newNode = new Node(Integer.parseInt(buffer));
    if(headNode == null) { //헤드노드가 없을 때 첫 헤드노드 설정
        newNode.setNext(headNode);
        headNode = newNode;
    } else { //헤드가 있을 때 추가되는 데이터의 생성
        headNode.setBack(newNode);
        newNode.setNext(headNode);
        headNode = newNode;
    }
}
```

- 외부 파일로부터 데이터를 한 줄씩 읽어온다.
- 읽을 데이터가 있는 동안 while문을 계속 반복시킨다.
  - LinkedList는 첫 헤드노드를 가지고 다음 노드를 찾아가는 자료구조이기 때문에, 첫 헤드노드가 없을 때는 그 케이스에 대한 처리를 해줘야한다.
  - 노드를 삽입할때 원래 있던 노드는 뒤로 밀리는 형식으로 생성해나가기 때문에, 새 노드의 다음노드로 현재의 헤드노드를 지정해주고, 새롭게 헤드노드를 지금 삽입하는 새로운 노드로 지정해준다.
  - 헤드가 이미 있을 때에는 뒤로 밀릴 헤드노드의 이전 노드를 찾아가기 위해 setBack()이라는 메소드를 통해 앞의 노드의 주소를 지정해준다.
  - 삽입되는 새로운 노드의 다음 노드값을 지정해주고, 헤드노드로 새롭게 입력되는 노드를 지정해준다.
  - 이렇게 한다면 노드를 단방향이 아닌 양방향으로 참조할 수 있게 된다.

2. DummyNode 생성 및 지정

```
/* 헤드노드의 앞부분을 처리해주기 위한 더미노드 생성 */
Node dummyNode = new Node();
headNode.setBack(dummyNode);
dummyNode.setNext(headNode);
headNode = dummyNode;
```

- 헤드노드에 데이터가 들어있지만, 삽입 정렬의 n번째 원소와 (n-1)개의 모든 원소를 비교하는 특성 상, 뒤로 가다보면 null값을 참조하게 되어 NullPointer에러를 발생한다.
- 이 것을 방지하기 위해 원소 값을 0으로 가지고 있는 더미노드를 헤드로 지정해준다.
- 이렇게 한다면 실제 데이터는 headNode의 다음값부터 저장된다.

3. LinkedList의 삽입 정렬

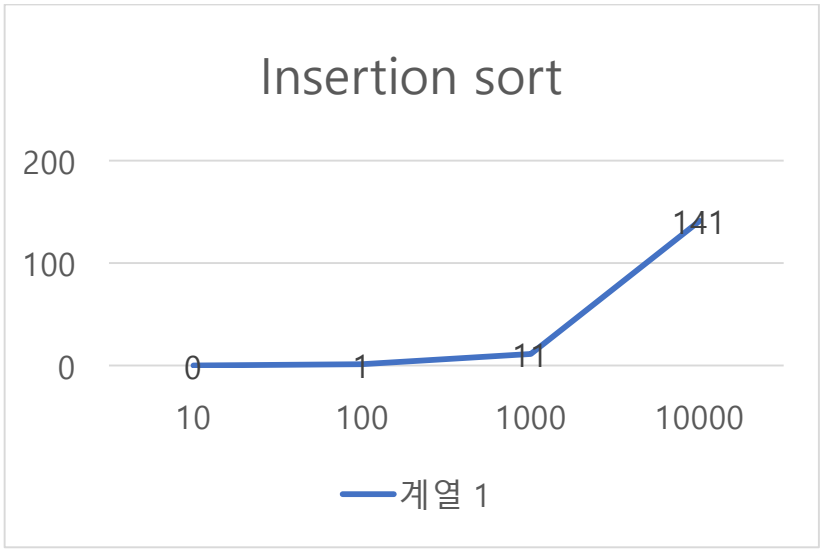
```
/*Insertion sort*/

Node nextNode = headNode.next().next(); //while문 제어를 위한 노드 생성
long startTime = System.currentTimeMillis(); //정렬 시작시간
while(nextNode != null) {


    int keyValue = nextNode.element();
    Node previousNode = nextNode.back();
    while( (previousNode != null) && (previousNode.element() > keyValue)) {
        if(previousNode.back() != null) {
            previousNode.next().setElement(previousNode.element());
            previousNode = previousNode.back();
        }
    }
    previousNode.next().setElement(keyValue);
    nextNode = nextNode.next();
}
long endTime = System.currentTimeMillis(); //정렬 종료 시간

long time = (endTime - startTime);
System.out.println(time); //정렬시간 출력

/* Insertion sort END */
```

- while문 제어를 위한 노드를 만들어준다. 삽입 정렬의 특성상 시작 노드는 2번째 노드부터 시작한다. 여기서 더미노드가 끼어있기 때문에 세번째 노드를 nextNode로 지정한다.
- while문이 실행되기 전과 실행된 후의 시간을 측정한다. 일단 시작 시간을 선언한다.
- nextNode값이 null이 아닌 동안 while문을 계속 반복한다.
    - 일단 원소를 정렬하기 위한 기준값을 위해 key값으로 현재 nextNode의 원소값을 저장한다.
    - 현재 노드와 이전 노드들과 비교를 해야하기 때문에 이전노드를 지정한다. back() 메소드를 이용하여 이전 노드를 참조한다.
    - 이전 노드의 값이 null이 아니면서 동시에 이전 노드의 원소값이 키 값보다 큰 경우 동안 내부 반복문을 실행한다.
    - 이전 노드의 원소값이 키 값보다 크다면, 앞으로 옮겨 자리를 바꿔주어야 한다. 이전노드의 다음노드의 원소값으로, 이전노드의 원소값을 지정한다.

- 이전노드(previousNode)를 이전노드의 이전 노드로 지정한다.
- 노드를 뒤로 이동하여 반복비교를 한다. 키값이 큰 경우에는 반복문은 끝나고 멈춘 노드에 키값을 넣어주어 정렬을 한다.
- 다음 키값 지정을 위해 nextNode값을 nextNode값의 다음 노드값을 지정해준다.
- 계속 반복 진행하여 마지막 노드까지 끝나고, nextNode가 null이 된다면 모든 데이터를 정렬한 것이므로 while문을 종료한다.
- 종료시간을 선언하고, 종료시간에서 시작시간을 빼어 걸린 시간을 출력한다.

3. 결과(시간 복잡도 포함)

- 10개 정렬 시 : 0 Milisec (정렬 아이템이 별로 없어 측정 불가)
- 100개 정렬 시 : 1 Milisec
- 1000개 정렬 시 : 11 Milisec
- 10000개 정렬 시 : 141 Milisec



- 삽입 정렬의 시간복잡도는 O(n^2)이다.
- 갯수가 늘어남에 따라 대각선 방향으로 증가하는 것을 확인할 수 있다.
- 정렬 데이터의 수가 크지 않아 정확하게 확인할 수 없지만, 수업시간에 배웠던 삽입정렬의 그래프 모양과 비슷하게, 정렬시간이 급격하게 늘어나지는 않으나 대각선 방향으로 상승하는 것을 확인할 수 있다.