자료구조 실습 보고서
[제 14 주] 우선순위큐 – 성능비교

제출일 : 2015 – 06 – 07

20140239 5 이승희

1. 프로그램 설명서

1) 주요 알고리즘 / 자료구조 / 기타

: Max Priority Queue 를 Sorted Array, Sorted Linked List, Max Heap 의 세 가지로 구현한다. Sorted Array 는 Increasing order 이고 Sorted Linked List 는 Decreasing order 로 구현한다.

2) 함수 설명서

(1) AppController

- public void showAll() //
- public void run() //
- private void reset() //
- private void showSize() //
- private void maxValue() //
- private void removeMax() //
- private void add(int input) //
- private void showMessage(MessageID aMessageID) //

(2) AppView

- public char inputCharacter() //
- public int inputNumber() //
- public void outputAdd(Comparable anElement) //
- public void outputSize(int size) //
- public void outputMaxValue(int maxValue) //
- public void outputRemovedMax(int maxValue) //
- public void outputelement(Comparable anElement) //

(3) HeapPriorityQueueprivate

- public class PriorityQueueIterator<T> implements Iterator //
- public PriorityQueueIterator<T> priorityQueue() //

(4) PriorityQueue

- public boolean isEmpty() //
- public boolean isFull() //
- public T max() //
- public int size() //
- public Boolean add(T anElement) //
- public T removeMax() //

(5) SortedArrayPriorityQueue

(6) SortedLinkedPriorityQueue

(7) Iterator

- public Boolean hasNext() //

- public T next() //

3) 종합 설명서

: 키보드에서 해야할 일을 입력받아 그에 따른 일을 수행한다.

- 'I' : 키보드에서 숫자를 입력 받아 Priority Queue 에 삽입한다. 정수 값을 입력 받기 전에 검사하여 Priority Queue 가 full 이면 오류 메시지를 내보낸다. Priority Queue 가 Full 이 아니면, 출력 예에서와 같이 정수 값을 입력하여 삽입한다.
- 'm' : 현재 Priority Queue 가 가지고 있는 최대값을 출력한다. Priority Queue 가 empty 이면, 오류 메시지를 출력한다.
- 'd' : Priority Queue 에서 최대값을 삭제한다. 삭제 전에 큐가 비어있는지 확인하여, empty 이면 오류 메시지를 출력한다.
- 'v' : Priority Queue 의 내용을 보여준다. 출력하기 전에 큐가 비어있는지 확인하여 비어있으면 오류 메시지를 출력한다.
- 'x' : Priority Queue 의 원소를 차례로 삭제하면서 출력한다. 큐가 비어있는지 확인하여 비어있으면 오류 메시지를 출력한다.
- 'r' : 난수를 10 개를 생성하여 삽입한다. Priority Queue 가 full 일 때까지만 삽입한다. 그러므로 삽입 전에 full 검사를 한다. 삽입이 끝난 후에는 삽입된 원소의 개수를 출력한다.
- 'n' : Priority Queue 가 가지고 있는 원소의 개수를 출력한다.
- 'q' : 프로그램을 종료한다.

최대 트리는 각 노드의 키 값이 자식 노드의 키 값보다 작지 않은 트리이다. 이러한 성질을 만족하는 트리는 루트 노드가 가장 큰 키 값을 갖게 된다. 최대 히븐 최대 트리가 완전이진트리의 모습을 한 트리이다. 최대 힙은 최대 트리이므로, 당연히 루트가 가장 큰 키 값을 갖는다.

2. 프로그램 장단점 분석

: 프로그램을 통해 최대 힙의 형태를 갖는 트리의 출력을 확인할 수 있고 알고리즘을 이해할 수 있다. 시간을 알 수 없다는 점이 단점이라고 할 수 있다.

3. 실행 결과 분석

1) 입력과 출력

(1) Sorted Array Priority Queue 로 구현

```
<우선순위 큐를 시작합니다>

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : i
- 삽입할 정수값을 입력하시오: 50
50이 정상적으로 입력 되었습니다.

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : r
65이 정상적으로 입력 되었습니다.
74이 정상적으로 입력 되었습니다.
39이 정상적으로 입력 되었습니다.
79이 정상적으로 입력 되었습니다.
78이 정상적으로 입력 되었습니다.
26이 정상적으로 입력 되었습니다.
86이 정상적으로 입력 되었습니다.
88이 정상적으로 입력 되었습니다.
69이 정상적으로 입력 되었습니다.
90이 정상적으로 입력 되었습니다.
- 임의의 원소가 10 개 입력되었습니다.
```

```
[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : v
= Priority Queue의 내용 =
26 39 50 65 69 74 78 79 86 88 90

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : d
- 최대값 90이 삭제되었습니다.
[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : m
- Priority Queue의 최대값은 88 입니다.

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : n
- Priority Queue 에는 10개의 원소가 들어 있습니다.
```

```
[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : x
= 삭제된 원소들 =
88 86 79 78 74 69 65 50 39 26
= 삭제 종료 - Priority Queue 비었습니다.

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : q
<우선순위 큐가 끝났습니다>
```

(2) Sorted Linked List Priority Queue 로 구현

```
<우선순위 큐를 시작합니다>

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : i
- 삽입할 정수값을 입력하시오: 70
70이 정상적으로 입력 되었습니다.

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : r
95이 정상적으로 입력 되었습니다.
95이 정상적으로 입력 되었습니다.
29이 정상적으로 입력 되었습니다.
75이 정상적으로 입력 되었습니다.
54이 정상적으로 입력 되었습니다.
49이 정상적으로 입력 되었습니다.
18이 정상적으로 입력 되었습니다.
76이 정상적으로 입력 되었습니다.
70이 정상적으로 입력 되었습니다.
93이 정상적으로 입력 되었습니다.
- 임의의 원소가 10 개 입력되었습니다.
[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : m
- Priority Queue의 최대값은 95 입니다.

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : d
- 최대값 95이 삭제되었습니다.
```

```
[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : v
= Priority Queue의 내용 =
95 93 76 75 70 70 54 49 29 18

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : n
- Priority Queue 에는 10개의 원소가 들어 있습니다.
[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : x
= 삭제된 원소들 =
95 93 76 75 70 70 54 49 29 18
= 삭제 종료 - Priority Queue 비었습니다.

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : q
<우선순위 큐가 끝났습니다>
```

(3) Heap Priority Queue 로 구현

```
<우선순위 큐를 시작합니다>

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : i
- 삽입할 정수값을 입력하시오: 88
88이 정상적으로 입력 되었습니다.

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : r
84이 정상적으로 입력 되었습니다.
47이 정상적으로 입력 되었습니다.
41이 정상적으로 입력 되었습니다.
34이 정상적으로 입력 되었습니다.
57이 정상적으로 입력 되었습니다.
60이 정상적으로 입력 되었습니다.
86이 정상적으로 입력 되었습니다.
32이 정상적으로 입력 되었습니다.
10이 정상적으로 입력 되었습니다.
22이 정상적으로 입력 되었습니다.
- 임의의 원소가 10 개 입력되었습니다.
[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : v
= Priority Queue의 내용 =
88 86 60 84 34 47 57 41 32 10 22

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : d
- 최대값 88이 삭제되었습니다.
```

```
[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : m
- Priority Queue의 최대값은 86 입니다.

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : n
- Priority Queue 에는 10개의 원소가 들어 있습니다.
[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : x
= 삭제된 원소들 =
86 84 60 57 47 41 34 32 22 10
= 삭제 종료 - Priority Queue 비었습니다.

[다음 중 해야 할 일의 코드를 선택하시오]
i : 입력
m : 최대값 보기
d : 최대값 삭제
v : Priority Queue 내용 보기
x : 모든 원소 차례대로 삭제하여 출력
r : 난수를 생성하여 10개 입력
n : 원소의 개수 보기
q : 프로그램 종료
? 해야할 일의 코드를 치시오 : q
<우선순위 큐가 끝났습니다>
```

2) 결과분석

: Sorted Array 는 Increasing order, Sorted Linked List 는 Decreasing order 로 두 가지 모두 삽입, 삭제의 시간 복잡도가 동일하다. Max heap 은 아래의 표와 같이 시간복잡도가 다른 두 가지보다 작기 때문에 더 빠르다고 할 수 있다. 세 가지 모두 출력하는데에 반복자 iterator 를 사용하였기 때문에 시간복잡도는 **Θ(n)**이 소요된다.

Representation	Insert	Delete Max
Unsorted Array	$\Theta(1)$	$\Theta(n)$
Unsorted Linked List	$\Theta(1)$	$\Theta(n)$
Sorted Array	$O(n)$	$\Theta(1)$
Sorted Linked List	$O(n)$	$\Theta(1)$
Max Heap	$O(\log_2 n)$	$O(\log_2 n)$

4. 소스코드

1) AppController

```
import java.util.Random;

public class AppController {
    private AppView _appView;
    HeapPriorityQueue<Integer> _priorityQueue;

    // SortedArrayPriorityQueue<Integer> _priorityQueue;

    // SortedLinkedPriorityQueue<Integer> _priorityQueue;

    public AppController() {
        this._appView = new AppView();
    }

    public void showAll() {
        if (this._priorityQueue.isEmpty())
            this.showMessage(MessageID.Error_Empty);
        else {
            this.showMessage(MessageID.Notice_ShowStart);
            HeapPriorityQueue.PriorityQueueIterator iterator =
                    (HeapPriorityQueue.PriorityQueueIterator) this._priorityQueue
                    .priorityQueue();
            // SortedArrayPriorityQueue.PriorityQueueIterator iterator =
            // (SortedArrayPriorityQueue.PriorityQueueIterator)
            // this._priorityQueue
            // .priorityQueue();
            // SortedLinkedPriorityQueue.PriorityQueueIterator iterator =
            // (SortedLinkedPriorityQueue.PriorityQueueIterator)
            // this._priorityQueue
            // .priorityQueue();
            while (iterator.hasNext()) {
                this._appView.outputElement((Comparable) iterator.next());
            }
            System.out.println();
        }
    }
```

```
public void run() {
    this._priorityQueue = new HeapPriorityQueue<Integer>();
    // this._priorityQueue = new SortedArrayPriorityQueue<Integer>();
    // this._priorityQueue = new SortedLinkedPriorityQueue<Integer>();
    char command = 0;
    int input;

    this.showMessage(MessageID.Notice_StartProgram);

    while (command != 'q') {
        this.showMessage(MessageID.Notice_Menu);
        command = this._appView.inputCharacter();
        if (command == 'i') {
            input = this._appView.inputNumber();
            this.add(input);
        } else if (command == 'r') {
            Random random = new Random();
            int i = 0;
            for (i = 0; i < 10; i++) {
                if (this._priorityQueue.isFull())
                    break;
                else {
                    Integer newData = (int) (Math.random() * 100);
                    this.add(newData);
                }
            }
            System.out.println("- 임의의 원소가 " + i + " 개 입력되었습니다. ");
        } else if (command == 'v') {
            this.showAll();
        } else if (command == 'd') {
            this.removeMax();
        } else if (command == 'm') {
            this.maxValue();
        } else if (command == 'n') {
            this.showSize();
        } else if (command == 'x') {
            this.reset();
        } else if (command == 'q')
            this.showMessage(MessageID.Notice_EndProgram);
        else
            this.showMessage(MessageID.Error_WrongMenu);

    }
}

private void reset() {
    if (this._priorityQueue.isEmpty())
        this.showMessage(MessageID.Error_Empty);
    else {
        this.showMessage(MessageID.Notice_StartRemoveAll);
        for (int i = this._priorityQueue.size(); i > 0; i--) {
            this._appView.outputElement(this._priorityQueue.removeMax());
        }
        System.out.println();
        this.showMessage(MessageID.Notice_EndRemoveAll);
    }
}

private void showSize() {
    this._appView.outputSize(this._priorityQueue.size());
}

private void maxValue() {
    if (this._priorityQueue.isEmpty())
        this.showMessage(MessageID.Error_Empty);
    else
        this._appView.outputMaxValue((Integer) this._priorityQueue.max());
}
```

```
    private void removeMax() {
        if (this._priorityQueue.isEmpty())
            this.showMessage(MessageID.Error_Empty);
        else
            this._appView.outputRemovedMax(this._priorityQueue.removeMax());
    }

    private void add(int input) {
        if (this._priorityQueue.isFull())
            this.showMessage(MessageID.Error_InputFull);
        else {
            this._priorityQueue.add(input);
            this._appView.outputAdd(input);

        }
    }
    private void showMessage(MessageID aMessageID) {
        switch (aMessageID) {
        case Notice_StartProgram:
            System.out.println("<우선순위 큐를 시작합니다> ");
            break;
        case Notice_Menu:
            System.out.println();
            System.out.println("[다음 중 해야 할 일의 코드를 선택하시오]");
            System.out.println("i : 입력 ");
            System.out.println("m : 최대값 보기 ");
            System.out.println("d : 최대값 삭제 ");
            System.out.println("v : Priority Queue 내용 보기 ");
            System.out.println("x : 모든 원소 차례대로 삭제하여 출력 ");
            System.out.println("r : 난수를 생성하여 10개 입력 ");
            System.out.println("n : 원소의 개수 보기 ");
            System.out.println("q : 프로그램 종료 ");
            break;
        case Notice_EndProgram:
            System.out.println("<우선순위 큐가 끝났습니다> ");
            break;
        case Notice_ShowStart:
            System.out.println("= Priority Queue의 내용 =");
            break;
        case Notice_StartRemoveAll:
            System.out.println("= 삭제된 원소들 =");
            break;
        case Notice_EndRemoveAll:
            System.out.println("= 삭제 종료 - Priority Queue 비었습니다. ");
            break;
        case Error_WrongMenu:
            System.out.println("- Error : 잘못된 입력입니다.");
            break;
        case Error_InputFull:
            System.out.println("ERROR : Priority Queue가 꽉 차서 입력할 수 없습니다.");
            break;
        case Error_Empty:
            System.out.println("ERROR : Priority Queue는 비어있습니다.");
            break;
        default:
            break;
        }
    }
}
```

2) AppView

```
import java.util.Scanner;

public class AppView {
    private Scanner _scanner;

    public AppView() {
        this._scanner = new Scanner(System.in);
    }

    public char inputCharacter() {
        System.out.print("? 해야할 일의 코드를 치시오 : ");
        return this._scanner.next().charAt(0);
    }

    public int inputNumber() {
        System.out.print("- 삽입할 정수값을 입력하시오: ");
        return this._scanner.nextInt();
    }

    public void outputAdd(Comparable anElement) {
        System.out.println(anElement + "이 정상적으로 입력 되었습니다.");

    }

    public void outputSize(int size) {
        System.out.println("- Priority Queue 에는 " + size + "개의 원소가 들어 있습니다.");

    }

    public void outputMaxValue(int maxValue) {
        System.out.println("- Priority Queue의 최대값은 " + maxValue + " 입니다.");
    }

    public void outputRemovedMax(int maxValue) {
        System.out.println("- 최대값 " + maxValue + "이 삭제되었습니다.");
    }

    public void outputElement(Comparable anElement) {
        System.out.print(anElement + " ");
    }

}
```

3) DS1_14_201402395_이승희

```
public class DS1_14_201402395_이승희 {

    public static void main(String[] args) {
        AppController appController=new AppController();
        appController.run();
    }

}
```

4) HeapPriorityQueue

```java
public class HeapPriorityQueue<T> implements PriorityQueue<T> {
    private static final int DEFAULT_CAPACITY = 100;
    private static final int ROOT = 1;
    private int _maxSize;
    private int _size;
    private T[] _tree;

    public HeapPriorityQueue() {
        this._maxSize = this.DEFAULT_CAPACITY;
        this._size = 0;
        this._tree = (T[]) new Object[this._maxSize + 1];
    }

    @Override
    public boolean isEmpty() {
        return this._size == 0;
    }

    @Override
    public boolean isFull() {
        return this._size == this._maxSize;
    }

    @Override
    public T max() {
        if (this.isEmpty())
            return null;
        else
            return (T) this._tree[this.ROOT];
    }

    @Override
    public int size() {
        return this._size;
    }

    @Override
    public boolean add(T anElement) {
        if (this.isFull())
            return false;
        else {
            if (!(anElement instanceof Comparable))
                throw new IllegalArgumentException();
            this._size++;
            int i = this._size;
            Comparable<T> x = (Comparable<T>) anElement;
            while ((i > ROOT) && (x.compareTo((T) this._tree[i / 2]) > 0)) {
                this._tree[i] = this._tree[i / 2];
                i /= 2;
            }
            this._tree[i] = anElement;
            return true;
        }
    }
```

```
    @Override
    public T removeMax() {
        if (this.isEmpty())
            return null;
        T rootElement = (T) this._tree[ROOT];
        this._size--;
        if (this._size > 0) {
            // 삭제한 후에 적어도 하나의 원소가 남아 있다.
            // 그러므로 마지막 위치 (this._size+1)의 원소를 떼어내어,
            // Root 위치 (1) 로부터 아래쪽으로 새로운 위치를 찾아 내려간다.

            T lastElement = (T) this._tree[this._size + 1];
            int parent = ROOT;
            int biggerChild;
            while ((parent * 2) <= this._size) {
                // child가 존재. left, right 중에서 더 key값을 갖는 child를 biggerChild로 설정
                biggerChild = parent * 2;
                Comparable<T> comparable = (Comparable<T>) this._tree[biggerChild];
                if ((biggerChild < this._size)
                        && (comparable.compareTo(this._tree[biggerChild + 1]) < 0)) {
                    biggerChild++;
                }
                Comparable<T> comparable2 = (Comparable<T>) lastElement;
                if (comparable2.compareTo((T) this._tree[biggerChild]) >= 0)
                    break;
                this._tree[parent] = this._tree[biggerChild];
                parent = biggerChild;

            }
            this._tree[parent] = lastElement;
        }
        return rootElement;
    }
    public PriorityQueueIterator<T> priorityQueue() {
        return new PriorityQueueIterator();
    }

    public class PriorityQueueIterator<T> implements Iterator {
        private int _nextPosition;

        private PriorityQueueIterator() {
            this._nextPosition = 0;
        }

        public boolean hasNext() {
            return this._nextPosition < size();
        }

        public T next() {
            if (this._nextPosition == size())
                return null;
            else {
                T nextElement = (T) _tree[this._nextPosition + 1];
                this._nextPosition++;
                return nextElement;
            }
        }
    }
}
```

5) Iterator

```
public interface Iterator<T> {
    public boolean hasNext();
    public T next();
}
```

6) MessageID

```
public enum MessageID {
    // Message IDs for Notices:
    Notice_StartProgram,
    Notice_EndProgram,
    Notice_Menu,
    Notice_RandomAdd,
    Notice_ShowStart,
    Notice_ShowEnd,
    Notice_StartRemoveAll,
    Notice_EndRemoveAll,
    // Message IDs for Errors:
    Error_WrongMenu,
    Error_InputFull,
    Error_Empty
}
```

7) Node

```
public class Node<T extends Comparable>implements Comparable {
    private T _element;
    private Node<T> _next;

    public Node() {
        this._element = null;
        this._next = null;
    }

    public Node(T anElement) {
        this._element = anElement;
    }

    public Node(T anElement, Node<T> aNode) {
        this._element = anElement;
        this._next = aNode;
    }

    public T element() {
        return this._element;
    }

    public Node<T> next() {
        return this._next;
    }

    public void setElement(T anElement) {
        this._element = anElement;
    }

    public void setNext(Node<T> aNode) {
        this._next = aNode;
    }

    @Override
    public int compareTo(Object arg0){
        return (_element.compareTo(arg0));
    }


}
```

8) PriorityQueue

```
public interface PriorityQueue<T> {
    public boolean isEmpty();

    public boolean isFull();

    public T max();

    public int size();

    public boolean add(T anElement);

    public T removeMax();

}
```

9) SortedArrayPriorityQueue

```
public class SortedArrayPriorityQueue<T> implements PriorityQueue<T> {
    private static final int DEFAULT_MAX_SIZE = 50;
    private int _maxSize;
    private int _size;
    private T[] _element;

    public SortedArrayPriorityQueue() {
        this._maxSize = this.DEFAULT_MAX_SIZE;
        this._size = 0;
        this._element = (T[]) new Comparable[this._maxSize];
    }

    public SortedArrayPriorityQueue(int aMaxSize) {
        this._maxSize = aMaxSize;
        this._size = 0;
        this._element = (T[]) new Comparable[this._maxSize];
    }

    @Override
    public boolean isEmpty() {
        return this._size == 0;
    }

    @Override
    public boolean isFull() {
        return this._size == this._maxSize;
    }

    @Override
    public T max() {
        if (this.isEmpty())
            return null;
        else
            return (T) this._element[this._size - 1];
    }
    @Override
    public int size() {
        return this._size;
    }

    @Override
    public boolean add(T anElement) {
        if (this.isFull())
            return false;
        else {
            if (this._size == 0)
                this._element[0] = (T) anElement;

            else {
                int i = 0;
                while (this._element[i] != null) {
                    if (((Comparable) this._element[i]).compareTo(anElement) > 0)
                        break;
                    else
                        i++;
                }
                for (int j = this._size; j > i; j--)
                    this._element[j] = this._element[j - 1];
                this._element[i] = (T) anElement;

            }
            this._size++;
            return true;
        }
    }
```

```
    @Override
    public T removeMax() {
        if (isEmpty()) {
            return null;
        } else {
            T removedElement = this._element[this._size - 1];
            this._element[this._size - 1] = null;
            this._size--;
            return removedElement;
        }
    }

    public PriorityQueueIterator<T> priorityQueue() {
        return new PriorityQueueIterator();
    }

    public class PriorityQueueIterator<T> implements Iterator {
        private int _nextPosition;

        private PriorityQueueIterator() {
            this._nextPosition = 0;
        }

        public boolean hasNext() {
            return this._nextPosition < size();
        }

        public T next() {
            if (this._nextPosition == size())
                return null;
            else {
                T nextElement = (T) _element[this._nextPosition];
                this._nextPosition++;
                return nextElement;
            }
        }
    }
}
```

10) SortedLinkedPriorityQueue

```
public class SortedLinkedPriorityQueue<T> implements PriorityQueue<T> {
    private static final int DEFAULT_MAX_SIZE = 50;
    private int _maxSize;
    private int _size;
    private Node _head;

    public SortedLinkedPriorityQueue() {
        this._head = null;
        this._size = 0;
        this._maxSize = DEFAULT_MAX_SIZE;
    }

    public SortedLinkedPriorityQueue(int aMaxSize) {
        this._head = null;
        this._size = 0;
        this._maxSize = aMaxSize;
    }

    @Override
    public boolean isEmpty() {
        return this._size == 0;
    }

    @Override
    public boolean isFull() {
        return this._size == this._maxSize;
    }

    @Override
    public T max() {
        if (this.isEmpty())
            return null;
        else
            return (T) this._head.element();
    }
```

```
    @Override
    public int size() {
        return this._size;
    }

    @Override
    public boolean add(T anElement) {
        if (this.isFull()) {
            return false;
        } else {
            Node searchNode = this._head;
            Node previousNode = null;
            Node newNode = new Node((Comparable) anElement);

            while (searchNode != null) {
                if (searchNode.compareTo(anElement) < 0) {
                    break;
                } else {
                    previousNode = searchNode;
                    searchNode = searchNode.next();
                }
            }

            if (searchNode == this._head) {
                newNode.setNext(this._head);
                this._head = newNode;
            } else {
                newNode.setNext(searchNode);
                previousNode.setNext(newNode);
            }
            this._size++;
            return true;
        }
    }
    @Override
    public T removeMax() {
        if (this.isEmpty())
            return null;
        else {
            Node removedNode = this._head;
            this._head = this._head.next();
            this._size--;
            return (T) removedNode.element();
        }
    }
    public PriorityQueueIterator<T> priorityQueue() {
        return new PriorityQueueIterator();
    }

    public class PriorityQueueIterator<T> implements Iterator {
        private Node _nextPosition;

        private PriorityQueueIterator() {
            this._nextPosition = _head;
        }

        public boolean hasNext() {
            return (this._nextPosition != null);
        }

        public T next() {
            if (this._nextPosition == null) {
                return null;
            } else {
                T t = (T) _nextPosition.element();
                this._nextPosition = _nextPosition.next();
                return t;
            }
        }
    }
}
```