자료구조 실습 보고서

[제 04 주] 가방 성능 비교

제출일	2017/04/02
학 번	201000287
소 속	일어일문학과
이 름	유다훈

201000287 일어일문학과 유다훈

1 프로그램 설명서

1 주요 알고리즘 및 자료구조

- 알고리즘
  - 리스트의 다양한 구현 방법에 따른 차이점을 확인.
  - 입력
    - ◆ 없음
    - ◆ 필요한 데이터는 프로그램에서 생성
  - 출력
    - ◆ 데이터 크기 변화에 따른 삽입하기, 최대값 찾기 측정 결과
      - i 10000 ~ 50000
- 자료구조
  - 코인을 담는 무작위 1 차원 배열
  - 코인을 담을 때 코인의 값을 비교하여 정렬하여 삽입하는 1 차원 배열
  - 코인을 담는 Node
  - Node 가 다른 노드를 연결하며 노드가 추가 될 때마다 크기의 제한없이 계속 증가하는 연결된 체인 형태의 구조(Linked Chain)
  - 새로운 Node 를 삽입할 때 마다 Node 에 담긴 코인값을 비교하여 정렬하여 삽입하는 연결된 체인 형태의 구조
  - 메뉴, 메뉴 종료, 각 자료구조 실행시에 실행을 알리고, 잘못된 메뉴 접근 시 경고를 알리게 해주는 enum

2 함수 설명서

Class	AppController			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	run()	없음	없음	동전 가방 프로그램을 실행시키는 메소드
	showMessage(MessageID aMessageID)	에러메세지 혹은 알림	없음	에러메세지 혹은 알림에 따라 메세지를 출력하는 메소드
	void showTestResults()	없음	없음	데이터 크기, 데이터 삽입시간, 데이터 최대값 출력시간을 appView 의 outputResult 메소드에게 넘겨주는 메소드

Class	AppView			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppView()	없음	없음	생성자 메소드 값을 입력받는 스캐너 생성
	void outputResult(int aTestSize, long aTestInsertTime, long aTestFindMaxTime)	테스트 데이터 크기, 삽입시간, 최대값 찾기 시간	없음	테스트 데이터 크기, 데이터 삽입시간, 데이터 최대값 찾기를 출력해주는 메소드
	void outputMessage(String aMessageString)	출력하고 싶은 문자열	없음	메세지 및 오류출력 메소드.

Class	TestResult			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	TestResult()	없음	없음	테스트 결과값의 초기화
	TestResult(int aTestSize, long aTestInsertTime, long aTestFindMaxTime	테스트 데이터 크기, 데이터 삽입시간, 데이터 최대값 찾는 시간	없음	테스트 결과값을 주어지는 파라미터값으로 설정하는 생성자 메소드
	int testSize()	없음	데이터값	테스트에 쓰인 데이터크기를 리턴
	void setTestSize(int aTestSize)	데이터 크기	없음	테스트 사이즈를 입력받는 크기로 설정
	long testInsertTime()	없음	데이터 삽입 시간	데이터 삽입에 걸린 시간을 반환
	void setTestInsertTime(int aTestInsertTime)	데이터 삽입시간	없음	데이터 삽입 측정 시간을 입력받은 크기로 설정
	long testFindMaxTime()	없음	최대 데이터값 검색 시간	자료구조 내의 최대 데이터 값 검색시간을 반환
	void setTestFindMaxTime(int aTestFindMaxTime)	최대 데이터값 검색 시간	없음	최대 데이터값 검색 시간을 입력받은 시간으로 설정하는 메소드

Class	Coin			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Coin(int givenValue)	코인값	없음	주어지는 값으로 코인의 값을 정하는 생성자 메소드
	int value()	없음	코인값 리턴	코인 값을 반환
	void setValue(int newValue)	저장하고싶은 새로운 코인값	없음	코인값을 새롭게 저장
	boolean equals(Coin aCoin)	코인	입력된 코인의 값과 현재 코인이 값이 같으면 참, 아니면 거짓.	입력받은 코인값과 가방안에 있는 코인들 중 같은 값이 있는지 확인하려할 때 사용.

Class	Node			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Node()	없음	없음	노드를 초기화 시키는 생성자 메소드
	Node(Coin givenCoin)	저장하려는 코인	없음	주어진 코인값을 가지는 노드를 생성하는 생성자 메소드
	Node(Coin givenCoin, Node givenNext)	저장하려는 코인, 다음 노드	없음	주어진 코인을 가지고, 다음 노드를 알고 있는 노드를 생성하는 생성자 메소드
	Coin coin()	없음	코인	노드에 담긴 코인 리턴

	Node next()	없음	노드	현재 노드의 다음 노드값을 리턴
	void setCoin()	없음	없음	코인값을 새롭게 설정
	void setNext()	없음	없음	다음 노드값을 새롭게 설정

Class	UnsortedLinkedBag			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	UnsortedLinkedBag()	없음	없음	가방의 첫 번째 노드와 노드갯수를 초기화시키는 생성자 메소드
	int size()	없음	가방 내에 있는 노드의 갯수 리턴	가방 안에 있는 노드의 갯수를 리턴
	boolean isEmpty()	없음	가방이 비어있으면 참, 아니면 거짓	가방에 노드가 있는지 없는지 검사
	Coin maxElementValue()	없음	가방 안에 들어있는 노드들의 코인값 중 제일 큰 값을 가진 코인	가방 내부의 노드들을 비교해가며 제일 큰 코인값을 가진 노드의 코인을 리턴
	boolean add(Coin anCoin)	코인	true	주어진 코인값을 가지는 노드를 새롭게 생성한 후, 체인 형태의 자료구조에서 맨 앞 공간의 헤드 노드를 뒤로 미루고 추가함. 가방의 맨 앞 쪽.

Class	SortedLinkedBag			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	SortedLinkedBag()	없음	없음	가방의 첫 번째 노드와 노드갯수를 초기화시키는 생성자 메소드
	int size()	없음	가방 내에 있는 노드의 갯수 리턴	가방 안에 있는 노드의 갯수를 리턴
	boolean isEmpty()	없음	가방이 비어있으면 참, 아니면 거짓	가방에 노드가 있는지 없는지 검사
	Coin maxElementValue()	없음	가방 안에 들어있는 노드들의 코인값 중 제일 큰 값을 가진 코인	가방 내부의 제일 마지막 노드의 코인값을 리턴
	boolean add(Coin anCoin)	코인	true	주어진 코인값을 가지는 노드를 새롭게 생성한 후, 체인 형태의 자료구조에서 현재 노드의 코인값과 주어진 코인값을 비교해가며 정렬하여 노드를 삽입하는 메소드

Class	UnsortedArrayBag

	메소드	파라미터 설명	리턴값	메소드 설명
Method	UnsortedArrayBag()	없음	없음	가방을 기본값으로 초기화하는 생성자 메소드
	UnsortedArrayBag(int givenMaxSize)	배열의 최대값	없음	주어진 값으로 배열의 크기를 정하는 생성자 메소드
	int size()	없음	가방 안에 들어간 코인의 갯수 리턴	가방 안의 코인 갯수 리턴
	boolean isEmpty()	없음	가방이 비어있으면 참, 아니면 거짓	가방이 비어있는지 검사
	boolean isFull()	없음	가방이 꽉 차있으면 참 아니면 거짓	가방 내부 배열이 꽉 찼는지 검사
	Coin maxElementValue()	없음	가방 내부 코인들 중의 최대값을 가진 코인	가방 내부의 코인들을 비교해가며 제일 큰 코인값을 가진 코인을 찾아 반환한다.
	boolean add(Coin anCoin)	코인	true	가방이 꽉 안 차있다면 코인값의 크기와 상관없이 순서대로 코인을 삽입

Class	SortedArrayBag			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	SortedArrayBag()	없음	없음	가방을 기본값으로 초기화하는 생성자 메소드
	SortedArrayBag(int givenMaxSize)	배열의 최대값	없음	주어진 값으로 배열의 크기를 정하는 생성자 메소드
	int size()	없음	가방 안에 들어간 코인의 갯수 리턴	가방 안의 코인 갯수 리턴
	boolean isEmpty()	없음	가방이 비어있으면 참, 아니면 거짓	가방이 비어있는지 검사
	boolean isFull()	없음	가방이 꽉 차있으면 참 아니면 거짓	가방 내부 배열이 꽉 찼는지 검사
	Coin maxElementValue()	없음	가방 내부 코인들 중의 최대값을 가진 코인	배열의 맨 뒤의 위치에 제일 큰 값이 위치하므로 최대값 반영
	boolean add(Coin anCoin)	코인	true	넣을 코인값과 가방안에 들어있는 코인 값을 비교하여, 가방 안의 코인값이 크다면 배열의 한 칸씩 뒤로 미루고 밀리고 빈 자리에 넣을 코인을 삽입함. 정렬삽입.

Class	PerformanceMeasurement			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	PerformanceMeasurement()	없음	없음	성능 비교를 위한 데이터들을 기본값으로 초기화하는 생성자 메소드

	PerformanceMeasurement(int givenMaxTestSize, int givenNumberOfTest, int givenFirstTestSize, int givenSizeIncrement)	테스트 최대값, 테스트 횟수, 첫 삽입 데이터 값, 데이터 증가량	없음	주어진 값으로 데이터 값들을 초기화하는 생성자 메소드
	int numberOfTest()	없음	테스트 횟수	테스트가 몇 번 진행되었는지 횟수 리턴
	TestResult[] testResults()	없음	테스트 결과값이 담긴 배열 반환	테스트의 결과값이 담긴 배열을 반환한다.
	void generateData()	없음	없음	0 부터 최대 테스트 데이터 값까지 횟수를 반복하며 코인값으로 될 랜덤한 데이터값을 생성
	void testSortedArrayBag()	없음	없음	정렬 Array 가방에 값을 삽입하고는 시간과최대값을 찾는 시간을 측정하는 메소드
	void testUnsortedArrayBag()	없음	없음	정렬하지 않는 Array 가방에 값을 삽입하는 시간과 최대값을 찾는 시간을 측정하는 메소드
	void testSortedLinkedBag()	없음	없음	정렬 LinkedChain 가방에 값을 삽입하는 시간과 최대값을 찾는 시간을 측정하는 메소드
	void testUnsortedLinkedBag()	없음	없음	정렬하지 않는 LinkedChain 가방에 값을 삽입하는 시간과 최대값을 찾는 시간을 측정하는 메소드

3 종합 설명서

- 각 자료구조 별 구현 방법에 따른 차이점을 확인하고 자료구조 별 성능을 측정하는 프로그램
  - SortedArrayBag, UnsortedArrayBag, SortedLinkedBag, UnsortedLinkedBag 총 네 가지의 자료구조
- 프로그램을 실행하면 프로그램 내부에서 임의의 랜덤한 데이터값을 생성한다.
- PerformanceMeasurement메소드 내부에서 각 자료구조 당 데이터를 삽입하는 데 걸린 시간과 삽입한 데이터들 중 최대값을 가진 데이터를 찾는 데에 걸린 시간을 계산한다.
- 정해진 데이터값 구간에서 데이터 삽입 시간과 최대값 검색 시간을 검색하여 출력한다

  ⇒자료 구조들의 성능 차이를 알 수 있다.

2 프로그램 장단점 분석

- 장점
  - 정렬하는 자료구조와 정렬하지 않는 자료구조 간의 데이터 삽입 및 최대값 검색 시간을 대략적으로 알 수 있다.
  - 배열을 이용하는 ArrayBag 과 노드를 이용하는 LinkedBag 을 동시에 구현해봄으로써 두 자료구조 간의 구현 차이를 알 수 있다.
  - 특정 상황에서 어떠한 자료구조가 더 효율적인지 알 수 있다.

- 단점
  - 시간분석도에 의해 최대 데이터 값이 커지면 커질수록 데이터 삽입 및 최대값 검색 시간이 비례하여 증가한다.
  - 최대 데이터 값이 크면 클수록 결과 출력까지의 시간이 오래걸린다.
  - JAVA 의 특성상 시간 측정이 정확하지 못하다.

3 실행 결과 분석

1 입력과 출력

프로그램 실행 및 종료

```
<<List의 구현에 따른 실행 성능 차이 알아보기>>
[Sorted Array List]
크기: 10000     삽입하기: 1007924  최대값찾기: 835215
크기: 20000     삽입하기: 1801243  최대값찾기: 1680971
크기: 30000     삽입하기: 2675179  최대값찾기: 2629463
크기: 40000     삽입하기: 1977639  최대값찾기: 1938103
크기: 50000     삽입하기: 2393387  최대값찾기: 2426837
[Unsorted Array List]
크기: 10000     삽입하기: 1083801  최대값찾기: 151718924
크기: 20000     삽입하기: 1981747  최대값찾기: 588792986
크기: 30000     삽입하기: 3124443  최대값찾기: 1332704797
크기: 40000     삽입하기: 3045810  최대값찾기: 2291104011
크기: 50000     삽입하기: 3920545  최대값찾기: 3878326829
[Sorted Linked List]
크기: 10000     삽입하기: 924712  최대값찾기: 936334
크기: 20000     삽입하기: 1695742  최대값찾기: 1571132
크기: 30000     삽입하기: 2243376  최대값찾기: 2212929
크기: 40000     삽입하기: 1754399  최대값찾기: 1757662
크기: 50000     삽입하기: 3747331  최대값찾기: 3712272
[Unsorted Linked List]
크기: 10000     삽입하기: 1938044  최대값찾기: 170527748
크기: 20000     삽입하기: 2674895  최대값찾기: 643447944
크기: 30000     삽입하기: 4415003  최대값찾기: 1422724154
크기: 40000     삽입하기: 3436828  최대값찾기: 2687254784
크기: 50000     삽입하기: 4742377  최대값찾기: 6535639045
<<성능 측정을 종료합니다>>
```

출력 - SortedArrayBag

```
[Sorted Array List]
크기: 10000     삽입하기: 1007924   최대값찾기: 835215
크기: 20000     삽입하기: 1801243   최대값찾기: 1680971
크기: 30000     삽입하기: 2675179   최대값찾기: 2629463
크기: 40000     삽입하기: 1977639   최대값찾기: 1938103
크기: 50000     삽입하기: 2393387   최대값찾기: 2426837
```

출력 - UnsortedArrayBag

```
[Unsorted Array List]
크기: 10000     삽입하기: 1083801   최대값찾기: 151718924
크기: 20000     삽입하기: 1981747   최대값찾기: 588792986
크기: 30000     삽입하기: 3124443   최대값찾기: 1332704797
크기: 40000     삽입하기: 3045810   최대값찾기: 2291104011
크기: 50000     삽입하기: 3920545   최대값찾기: 3878326829
```

출력 - SortedLinkedBag

```
[Sorted Linked List]
크기: 10000     삽입하기: 924712   최대값찾기: 936334
크기: 20000     삽입하기: 1695742   최대값찾기: 1571132
크기: 30000     삽입하기: 2243376   최대값찾기: 2212929
크기: 40000     삽입하기: 1754399   최대값찾기: 1757662
크기: 50000     삽입하기: 3747331   최대값찾기: 3712272
```

출력 - UnsortedLinkedBag

```
[Unsorted Linked List]
크기: 10000     삽입하기: 1938044   최대값찾기: 170527748
크기: 20000     삽입하기: 2674895   최대값찾기: 643447944
크기: 30000     삽입하기: 4415003   최대값찾기: 1422724154
크기: 40000     삽입하기: 3436828   최대값찾기: 2687254784
크기: 50000     삽입하기: 4742377   최대값찾기: 6535639045
```

2 결과 분석

- 데이터의 최대 크기는 5 만이며, 1 만씩 증가한다.
  - 데이터 크기가 증가함에 따라 삽입하기와 최대값 찾기 시간은 증가한다.
- SortedArrayBag
  - 시간 복잡도
    - 삽입 시 : O(n)
    - 최대 값 검색 시 : O(1)
      - i 데이터 삽입 시 정렬해가며 삽입하기 때문에 시간이 오래 걸린다.
      - ii 최대 값 검색 시 정렬된 배열에서는 배열의 제일 마지막이 최대 값이므로 검색 시간이 별로 걸리지 않는다.

- UnsortedArrayBag
  - 시간 복잡도
    - 삽입 시 : O(1)
    - 최대 값 검색 시 : O(n)
      i. 데이터 삽입 시 정렬하지 않고 그냥 삽입하기 때문에 시간이 얼마 걸리지 않는다.
      ii. 최대 값 검색 시 정렬되어 있지 않기 때문에 하나하나 비교를 해가며 최대값을 찾아야하기 때문에 데이터 크기에 비례하여 오래 걸린다.
- SortedLinkedBag
  - 시간 복잡도
    - 삽입 시 : O(n)
    - 최대 값 검색 시 : O(n)
      i. 데이터 삽입 시 원래 들어있는 값과 비교 및 정렬해가며 삽입해야 하기 때문에 삽입 시간이 오래 걸린다.
      ii. 최대 값 검색 시 연결 노드리스트들의 맨 마지막에 최대값이 있으나, 해당 노드를 바로 조회할 수 없기 때문에 마지막 노드(next 값이 null)가 나올 때 까지 노드를 검색해야한다.
- UnsortedLinkedBag
  - 시간 복잡도
    - 삽입 시 : O(1)
    - 최대 값 검색 시 : O(n)
      i. 데이터 삽입 시 값을 정렬하지 않고 바로 넣기 때문에 오래 걸리지 않음
      ii. 최대 값 검색 시 맨 마지막의 노드까지 비교를 해야 하기 때문에 데이터 량에 비례하여 검색 시간이 증가한다.
- 출력 결과
  - 삽입
    - 삽입 결과의 경우 UnsortedArray, UnsortedLinked, SortedLinked, SortedArray 순으로 빨라야 하나, JAVA 의 특성상 시간체크가 제대로 되지 않았다.
    - SortedLinked보다 SortedArray가 느려야하는 이유는, Array의 경우 배열을 한 칸 씩 뒤로 밀어주어야 하나, LinkedChain 의 경우 이전노드와 현재노드 사이에 새로운 값을 넣기만 하면 되기 때문.
  - 출력
    - SortedArray, SortedLinkedChain, UnsortedArray, UnsortedLinkedChain 의 순으로 빠르다.
    - SortedArray 의 경우 최대값은 배열의 맨 끝에 있으므로 맨 끝 값만 참조해주면 된다.
    - UnsortedLinkedChain 의 경우 체인의 맨 끝까지 배교해가며 이전노드, 현재노드의 값을 바꾸어가며 검사를 하기 때문에 제일 느리다. 또한 참조해야할 메모리의 크기가 배열에 비해 크다.
- 결론
  - 메모리(입력할 데이터의 크기)가 정해져 있는 경우에는 배열

- 메모리(입력할 데이터의 크기)가 정해져 있지 않은 경우에는 LinkedChain 을 사용해야 하나, 검색속도의 차이가 많이 나기 때문에 배열을 사용하는게 바람직하다.

4 소스 코드

Class	AppController

```
public class AppController {
	private AppView   _appView;
	private PerformanceMeasurement _pm;

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

	public void run() {
		//Start
		this._pm = new PerformanceMeasurement();
		this.showMessage(MessageID.Notice_StartProgram);
		this._pm.generateData();

		//SortedArray
		this.showMessage(MessageID.Notice_SortedArrayStart);
		this._pm.testSortedArrayBag();
		this.showTestResult();

		//UnsortedArray
		this.showMessage(MessageID.Notice_UnsortedArrayStart);
		this._pm.testUnsortedArrayBag();
		this.showTestResult();

		//SortedLinked
		this.showMessage(MessageID.Notice_SortedLinkedStart);
		this._pm.testSortedLinkedBag();
		this.showTestResult();


		//UnsortedLinked
		this.showMessage(MessageID.Notice_UnsortedLinkedStart);
		this._pm.testUnsortedLinkedBag();
		this.showTestResult();



		//end program
		this.showMessage(MessageID.Notice_EndProgram);
	}

	private void showTestResult() {
		TestResult testResults[] = this._pm.testResults();
		for(int index = 0; index < this._pm.numberOfTest(); index++) {
			this._appView.outputResult(testResults[index].testSize(),
						testResults[index].testInsertTime(),
						testResults[index].testFindMaxTime());
		}
	}

	private void showMessage(MessageID aMessageID) {
		switch (aMessageID) {
		case Notice_StartProgram :
```

```
                        this._appView.outputMessage("<<List 의 구현에 따른 실행 성능 차이
알아보기>>₩n");
                        break;
                case Notice_EndProgram :
                        this._appView.outputMessage("<<성능 측정을 종료합니다>>₩n");
                        System.exit(0);
                        break;
                case Notice_UnsortedArrayStart :
                        this._appView.outputMessage("[Unsorted Array List]₩n");
                        break;
                case Notice_SortedArrayStart :
                        this._appView.outputMessage("[Sorted Array List]₩n");
                        break;
                case Notice_UnsortedLinkedStart:
                        this._appView.outputMessage("[Unsorted Linked List]₩n");
                        break;
                case Notice_SortedLinkedStart:
                        this._appView.outputMessage("[Sorted Linked List]₩n");
                        break;
                case Error_WrongMenu:
                        this._appView.outputMessage("<<ERROR: 잘못된 메뉴입니다.>>₩n");
                        break;
                default:
                        break;

                }
        }

}

        }

}
```

Class	AppView

```
import java.util.Scanner;

public class AppView {
        private Scanner _scanner;

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

        public void outputResult(int aTestSize, long aTestInsertTime, long aTestFindMaxTime) {
                System.out.println("크기: " + aTestSize + "       " + "삽입하기: " + aTestInsertTime + "   " +
"최대값찾기: "
                                                                        + aTestFindMaxTime);
        }
        public void outputMessage(String aMessageString) {
                System.out.print(aMessageString);
        }
}
```

Class	SortedArrayBag

```
public class SortedArrayBag {

	private static final int DEFAULT_MAX_SIZE = 100;
	private int _maxSize;
	private int _size;
	private Coin _elements[];

	public SortedArrayBag() { //가방의 기본 생성자.
		this._maxSize = DEFAULT_MAX_SIZE;
		this._elements = new Coin[DEFAULT_MAX_SIZE];
		this._size = 0;

	}
	public SortedArrayBag(int givenMaxSize) { //가방의 생성자
		this._maxSize = givenMaxSize; //주어진 값으로 최대크기를 정함
		this._elements = new Coin[givenMaxSize]; //주어진 값만큼 코인이 들어갈 가방크기 생성
		this._size = 0;

	}
	public int size() {
		return this._size; //코인이 들어가는 사이즈 리턴
	}
	public boolean isEmpty() {
		return (this._size == 0); // 가방이 비었는지 안비었는지
	}
	public boolean isFull() {
		return (this._size == this._maxSize); //가방이 꽉 찼는지 안찼는지
	}

	public boolean add (Coin anCoin) {
		int index = 0 ; //코인이 들어갈 자리를 가리키는 값
		if (anCoin.value() > this.DEFAULT_MAX_SIZE || anCoin.value() < 0) {
			return false;
		}
		if (this.isFull()) {
			return false;
		} else {

			for (int i = 0; i < this._size; i++) { //코인의 갯수까지 반복
				if(this._elements[index].value() > anCoin.value()) //현재값이 코인의
값보다 크다면
					index = i; //지금 현재값이 가르키는 위치를 인덱스값으로
저장
					break; //반복문 탈출
				}

			}

			for(int i = this._size-1; i > index; i-- ) { //i 를 현재 코인갯수보다 1 낮은 값으로.
인덱스값보다 낮아질때까지 하나씩 줄여감
				this._elements[i+1] = this._elements[i]; //현재 코인갯수보다 1 높은
빈칸의 위치에 현재 코인삽입. 한칸씩 뒤로 밀려나기.

		}
```

```
                this._elements[index] = anCoin; //비어있는 현재 코인값에 전달받은 코인값 삽입.
                this._size++; //코인의 총 개수 증가
                return true;

        }

        public Coin maxElement() {
                return this._elements[this._size];
        }
        }
```

Class	SortedLinkedBag

```
public class SortedLinkedBag {
        private static final int MAX_TEST_SIZE = 50000;
        private int _size; //여기서 사이즈는 총 노드의 갯수
        private Node _head;


        public SortedLinkedBag() {
                this._head = null; //초기화실행
                this._size = 0;

        }
        public int size() {
                return this._size; //노드갯수 리턴.
        }
        public boolean isEmpty() {
                return (this._size == 0); //비었는지 안비었는지 체크
        }

        public Coin maxElement() { //최대값 찾기
                Node searchNode = this._head;
                Coin maxCoin = null;


                while(searchNode != null) { //검색할 노드가 없을때까지. 노드가 존재하는한 계속검색
                        maxCoin = searchNode.coin();
                        searchNode = searchNode.next();
                }
                return maxCoin;
        }

        public boolean add(Coin anCoin) { //코인 저장하기
                if(anCoin.value() < this.MAX_TEST_SIZE || anCoin.value() < 0 ){
                        return false;
                } else {
                        Node search = this._head;
                        Node previousNode = null;

                        int i = 0;
                        while (search != null) {
                                if(search.coin().value() > anCoin.value()) {
                                        break;
                                }
```

```
                                        previousNode = search;
                                        search = search.next();
                                }

                                if (search == this._head) {
                                        Node newNode = new Node(anCoin, this._head);
                                        this._head = newNode;
                                } else {
                                        Node newNode = new Node(anCoin, search);
                                        previousNode.setNext(newNode);

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

Class	UnsortedArrayBag

```
public class UnsortedArrayBag {

        private static final int DEFAULT_MAX_SIZE = 100;
        private int _maxSize;
        private int _size;
        private Coin _elements[];


        public UnsortedArrayBag() { //가방의 기본 생성자.
                this._maxSize = DEFAULT_MAX_SIZE;
                this._elements = new Coin[DEFAULT_MAX_SIZE];
                this._size = 0;

        }

        public UnsortedArrayBag(int givenMaxSize) { //가방의 생성자
                this._maxSize = givenMaxSize; //주어진 값으로 최대크기를 정함
                this._elements = new Coin[givenMaxSize]; //주어진 값만큼 코인이 들어갈 가방크기 생성
                this._size = 0;

        }
        public int size() {
                return this._size; //코인이 들어가는 사이즈 리턴
        }
        public boolean isEmpty() {
                return (this._size == 0); // 가방이 비었는지 안비었는지
        }
        public boolean isFull() {
                return (this._size == this._maxSize); //가방이 꽉 찼는지 안찼는지
        }

        public Coin maxElement() { //코인들 중의 최대값 반영
                Coin maxValue = this._elements[0];
                for (int i = 0; i < this.size() ; i++) { //가방안에 들어있는 코인의 갯수만큼 확인
```

```
                        if(maxValue.value() < _elements[i].value()) //최대값보다 i번째 코인의 값이 크다면
                        maxValue = this._elements[i]; //i 번째 코인의 값을 최대값으로 변경
                }
                return maxValue;
        }


        public boolean add(Coin anElement) { // 가방에 코인넣기
                if(this.isFull()) { //만약 꽉 찼으면
                        return false; //넣지 말기
                } else { //꽉안찼으면
                        this._elements[this._size] = anElement; //size 번째 공간에 코인을 넣기
                        this._size++; //사이즈 증가
                        return true;
                }
        }

}
```

Class	UnsortedLinkedBag

```
public class UnsortedLinkedBag {

        private int _size; //여기서 사이즈는 총 노드의 갯수
        private Node _head;

        public UnsortedLinkedBag() {
                this._head = null; //초기화실행
                this._size = 0;

        }
        public int size() {
                return this._size; //노드갯수 리턴.
        }
        public boolean isEmpty() {
                return (this._size == 0); //비었는지 안비었는지 체크
        }


        public Coin maxElement() { //최대값 찾기
                Node searchNode = this._head; //현재 노드는 맨 앞부터
                Coin maxValue = this._head.coin(); //최대값을 저장할 변수

                while(searchNode != null) { //검색할 노드가 없을때까지. 노드가 존재하는한 계속검색
                if (maxValue.value() < searchNode.coin().value()) { //최대값과 현재 찾고있는 노드의 코인값과
비교하여 노드값이 크면
                        maxValue = searchNode.coin(); //맥스값을 현재 코인으로 교체
                }
                searchNode = searchNode.next(); //다음 노드로 이동
                }
                return maxValue;
        }

        public boolean add(Coin anCoin) { //코인 저장하기
```

```
                Node newNode = new Node(anCoin);
                //주어진 코인값의 새로운 노드를 생성
                newNode.setNext(this._head); // 현재 노드의 다음값을 현재 헤드로 지정. 지금 헤드노드는 뒤로 밀려남.
                this._head = newNode; //현재헤드를 지금 헤드로 새로 지정. 노드를 추가할때는 맨 앞에서 추가하므로 원래 있던 헤드는 뒤로 밀려나가는 형식
                this._size++;
                return true;
        }

}
```

Class	PerformanceMeasurement

```
import java.util.Random;

public class PerformanceMeasurement {
        private static final int MAX_TEST_SIZE = 50000;
        private static final int NUMBER_OF_TESTS = 5;
        private static final int FIRST_TEST_SIZE = 10000;
        private static final int SIZE_INCREMENT = 10000;

        private int _maxTestSize;
        private int _numberOfTests;
        private int _firstTestSize;
        private int _sizeIncrement;

        private int [] _data;
        private TestResult [] _testResults;

        public PerformanceMeasurement() {
                this._maxTestSize = this.MAX_TEST_SIZE;
                this._numberOfTests = this.NUMBER_OF_TESTS;
                this._firstTestSize = this.FIRST_TEST_SIZE;
                this._sizeIncrement = this.SIZE_INCREMENT;

                this._data = new int[this.MAX_TEST_SIZE];
                this._testResults = new TestResult[this.NUMBER_OF_TESTS];
        }

        public PerformanceMeasurement(int givenMaxTestSize, int givenNumberOfTest, int givenFirstTestSize, int givenSizeIncrement) {
                this._maxTestSize = givenMaxTestSize;
                this._numberOfTests = givenNumberOfTest;
                this._firstTestSize = givenFirstTestSize;
                this._sizeIncrement = givenSizeIncrement;

                this._data = new int[givenMaxTestSize];
                this._testResults = new TestResult[givenNumberOfTest];

        }

        public int numberOfTest() {
                return this._numberOfTests;
        }
```

```
	public TestResult[] testResults() {
		return this._testResults;
	}

	public void generateData() {
		int i = 0;
		Random random = new Random();

		while (i < this._maxTestSize) {
			this._data[i] = random.nextInt(this._maxTestSize);
			i++;
		}
	}

	public void testSortedArrayBag() {
		SortedArrayBag bag;
		Coin maxCoin;

		int testSize;
		long timeForAdd, timeForMax;
		long start, stop;
		int testDataCount;
		int testCount = 0;

		testSize = this._firstTestSize; // 10000
		while(testCount < this._numberOfTests) {

			bag = new SortedArrayBag(testSize);
			testDataCount = 0;
			timeForAdd = 0;
			timeForMax = 0;

			while(testDataCount < testSize) {
				Coin coin = new Coin(this._data[testDataCount]);
				start = System.nanoTime() ;
				bag.add(coin);
				stop = System.nanoTime() ;
				timeForAdd += (stop - start);

				start = System.nanoTime() ;
				maxCoin = bag.maxElement();
				stop = System.nanoTime() ;
				timeForMax += (stop - start);
				testDataCount ++;

			}
			this._testResults[testCount] = new TestResult(testSize, timeForAdd,
timeForMax);
			testSize += this._sizeIncrement;
			testCount++;

		}

	}
	public void testUnsortedArrayBag() {
		UnsortedArrayBag bag;
		Coin maxCoin;
```

```
                int testSize;
                long timeForAdd, timeForMax;
                long start, stop;
                int testDataCount;
                int testCount = 0;

                testSize = this._firstTestSize; // 10000
                while(testCount < this._numberOfTests) {

                        bag = new UnsortedArrayBag(testSize);
                        testDataCount = 0;
                        timeForAdd = 0;
                        timeForMax = 0;

                        while(testDataCount < testSize) {
                                Coin coin = new Coin(this._data[testDataCount]);
                                start = System.nanoTime();
                                bag.add(coin);
                                stop = System.nanoTime();
                                timeForAdd += (stop - start);

                                start = System.nanoTime() ;
                                maxCoin = bag.maxElement();
                                stop = System.nanoTime();
                                timeForMax += (stop - start);
                                testDataCount ++;

                        }
                        this._testResults[testCount] = new TestResult(testSize, timeForAdd,
timeForMax);
                        testSize += this._sizeIncrement;
                        testCount++;

                }


        }
        public void testSortedLinkedBag() {
                SortedLinkedBag bag;
                Coin maxCoin;

                int testSize;
                long timeForAdd, timeForMax;
                long start, stop;
                int testDataCount;
                int testCount = 0;

                testSize = this._firstTestSize; // 10000
                while(testCount < this._numberOfTests) {

                        bag = new SortedLinkedBag();
                        testDataCount = 0;
                        timeForAdd = 0;
                        timeForMax = 0;

                        while(testDataCount < testSize) {
                                Coin coin = new Coin(this._data[testDataCount]);
                                start = System.nanoTime() ;
```

```
                                        bag.add(coin);
                                        stop = System.nanoTime();
                                        timeForAdd += (stop - start);

                                        start = System.nanoTime();
                                        maxCoin = bag.maxElement();
                                        stop = System.nanoTime() ;
                                        timeForMax += (stop - start);
                                        testDataCount ++;

                                }
                                this._testResults[testCount]    =    new    TestResult(testSize,    timeForAdd,
timeForMax);
                                testSize += this._sizeIncrement;
                                testCount++;

                        }


        }
        public void testUnsortedLinkedBag() {
                        UnsortedLinkedBag bag;
                        Coin maxCoin;

                        int testSize;
                        long timeForAdd, timeForMax;
                        long start, stop;
                        int testDataCount;
                        int testCount = 0;

                        testSize = this._firstTestSize; // 10000
                        while(testCount < this._numberOfTests) {

                                bag = new UnsortedLinkedBag();
                                testDataCount = 0;
                                timeForAdd = 0;
                                timeForMax = 0;

                                while(testDataCount < testSize) {
                                        Coin coin = new Coin(this._data[testDataCount]);
                                        start = System.nanoTime() ;
                                        bag.add(coin);
                                        stop = System.nanoTime() ;
                                        timeForAdd += (stop - start);

                                        start = System.nanoTime() ;
                                        maxCoin = bag.maxElement();
                                        stop = System.nanoTime() ;
                                        timeForMax += (stop - start);
                                        testDataCount ++;

                                }
                                this._testResults[testCount]    =    new    TestResult(testSize,    timeForAdd,
timeForMax);
                                testSize += this._sizeIncrement;
                                testCount++;

                        }
```

```
		}


}
```

Class	Coin

```
public class Coin {
		/*코인 클래스*/
		private int _value;

		public Coin() {
			//기본생성자
		}
		public Coin(int givenValue) {
			this._value   = givenValue; //코인값을 저장
		}

		public int value() {
			return this._value; //코인값 리턴
		}


		public void setValue(int newValue) {
			this._value = newValue; //코인값을 새로저장
		}

		public boolean equals(Coin aCoin) {
			return (this._value == aCoin.value());
		}

}
```

Class	Node

```
public class Node {

		private Coin _coin; //노드에 담기는 코인
		private Node _next; //다음 노드


		public Node(){ //노드 초기화
			this._coin = null;
			this._next = null;
		}

		public Node(Coin givenCoin) { //노드의 코인값을 설정
			this._coin = givenCoin;
			this._next = null;
		}
```

```
	public Node(Coin givenCoin, Node givenNext) { //노드의 코인과 다음노드값을 설정
		this._coin = givenCoin;
		this._next = givenNext;
	}

	public Coin coin() {


		return this._coin; //코인값 리턴
	}

	public Node next() {
		return this._next; //다음노드값 리턴

	}
	public void setCoin (Coin newCoin) {
		this._coin = newCoin; //코인값 새로 설정

	}
	public void setNext (Node newNext) {
		this._next = newNext; //다음 노드값 새로 설정
	}


}
```