자료구조 실습 보고서

[제 12 주] 정렬 알고리즘들의 성능 비교

제출일	2017/05/29
학 번	201000287
소 속	일어일문학과
이 름	유다훈

1 프로그램 설명서

1 주요 알고리즘 및 자료구조

- 알고리즘
  - 입출력
    - ◆ 필요한 데이터는 프로그램에서 생성
    - ◆ 성능 검증 결과를 출력
      - i 데이터 크기 변화에 따른 성능 측정 결과
      - ii 1000 에서부터 10000 까지 1000 단위씩 증가.
    - ◆ 삽입 정렬, 퀵정렬
- 자료구조
  - 오름차순 데이터 리스트
  - 내림차순 데이터 리스트
  - 무작위 데이터 리스트

1 함수 설명서

Class	AppController			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppController()	없음	없음	_appView 변수와 _manager 변수를 초기화 하는 생성자 메소드
	run()	없음	없음	성능비교 프로그램을 실행하는 실행 메소드
	void showTableTitle(ListOder anOrder)	어떠한 데이터 리스트를 사용하는지 알려주는 enum	없음	어떠한 데이터 리스트가 사용되었는지 알려주는 메세지를 호출지시하는 메소드
	void showTableHead()	없음	없음	삽입 정렬과 퀵 정렬의 데이터 구분
	void showMeasuerdResult()	없음	없음	테스트 결과를 1000 단위로 출력 지시
	void measureAndShowResultFor(ListOrder anOrder)	어떠한 데이터 리스트를 사용하는지 알려주는 enum	없음	해당 데이터 리스트를 사용하여 진행된 테스트 결과 출력

Class	AppView			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppView()	없음	없음	생성자 메소드 값을 입력받는 스캐너 생성
	void output(String aString)	문자열	없음	전달받은 문자열을 출력한다.
	void outputLine(String aString)	문자열	없음	전달받은 문자열을 출력하고 행을 바꾼다.

Class	final class DataGenerater()

	메소드	파라미터 설명	리턴값	메소드 설명
Method	static Integer[] ascendingOrderList(int aSize)	배열의 최대값	배열	전달받은 수만큼 배열의 크기를 정하고 배열의 크기만큼 데이터를 만들어서 저장한다.
	static Integer[] descendingOrderList(int aSize)	배열의 최대값	배열	전달받은 수만큼의 배열의 크기를 정하고, 배열의 크기만큼 데이터를 만들어 거꾸로 저장한다.
	static Integer[] randomOrderList(int aSize)	배열의 최대값	배열	전달받은 수만큼의 배열의 크기를 정하고, 배열의 크기만큼 데이터를 만들어 랜덤하게 저장한다.

Class	abstract class Sort<E>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	protected void swap(E[] aList, int t, int j)	배열, 정수형 변수 2 개	없음	전달받은 정수형 변수위치에 저장된 데이터 위치를 서로 바꿈
	public abstract boolean sort(E[] aList, int aSize)	배열, 배열의 최대길이	없음	정렬을 담당하는 메소드. abstract 선언을 하여 자식 메소드에서 구현을 강제시킴.

Class	InsertionSort <E extends Comparable <E>> extends Sort<E>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	boolean sort(E[]aList, int aSize)	배열, 배열크기	잘 삽입되었으면 true, 아니면 false	배열 내의 원소들의 값을 비교하여 최소값을 구한 후, 그 값을 맨 앞으로 보내고 삽입 정렬을 실행한다.

Class	QuickSort<E extends Comparable<E>> enxtends Sort<E>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	int pivot(E[] aList, int left, int right)	배열, 배열의 첫부분인덱스, 배열의 끝부분인덱스	피봇값	정렬 시 기준값이 되는 피봇값을 정하는 메소드
	int partition(E[] aList, int left, int right)	배열, 배열의 첫 부분 인덱스, 배열의 끝 부분 인덱스	중간값	재귀적 퀵정렬을 하기위해 필요한 중간값을 만든다.
	void quickSortRecursively(E[] aList, int left, int right)	배열, 배열의 첫 부분 인덱스, 배열의 끝 부분 인덱스	없음	파티션을 이용하여 배열을 두 군데로 나눈 후 나누어진 부분에 대하여 다시 퀵정렬을 재귀적으로 실행하는 메소드
	boolean sort(E[] aList, int aSize)	배열과 배열의 크기	제대로 정렬을 실행하고 true 리턴.	배열 내의 원소들의 값을 비교하여 최대값을 구한 후, 그 값을 맨 뒤로

			그렇지 않으면 false	보내고 퀵정렬을 실행한다.

Class	ParameterSet			
Method	메소드 이름	파라미터 설명	리턴값	메소드 설명
	ParameterSet (int givenStartingSize, int givenNumberOfSizeIncresingSteps, int givenIncrementSize)	시작하는 데이터값, 데이터 증가횟수, 증가값	없음	실험에 필요한 데이터의 시작값과 데이터의 증가 횟수, 증가값을 초기화 하는 생성자 메소드.
	int startingSize()	없음	시작값	실험에 사용하는 데이터의 시작값 리턴
	void setStartingSize(int startingSize)	시작값	없음	실험에 사용하는 데이터의 시작값 설정
	int numberOfSizeIncreasingSteps()	없음	증가횟수	실험에 사용하는 데이터의 증가 횟수 리턴
	void setNumberOfSizeIncreasingSteps(int numberOfSizeIncreasingSteps)	증가횟수	없음	실험에 사용하는 데이터의 증가 횟수 설정
	int incrementSize()	없음	증가값	실험에 사용하는 데이터의 1 회당 증가 값 리턴
	void setIncrementSize(int incrementSize)	증가값	없음	실험에 사용하는 데이터의 1 회당 증가값 설정
	int maxDataSize()	없음	최대 데이터 사이즈	실험에서 사용하는 데이터의 최대 데이터값 리턴

Class	Experiment			
Method	메소드 이름	파라미터 설명	리턴값	메소드 설명
	Experiment(ParameterSet givenParameterSet	실험에 필요한 정보값들	없음	실험에 필요한 정보값들을 초기화하는 생서자 메소드
	Integer[] copyListOfGivenSize(Integer[] aList, int copiedSize)	배열, 배열의 사이즈	배열	주어진 배열을 또다른 배열로 복사하는 메소드
	long durationOfSingleSort(Sort<Integer> aSort, Integer[] aList	정렬방법, 배열	실험시간	주어진 정렬방법으로 배열을 정렬하여 배열을 정렬하는데 걸린 시간을 리턴하는 메소드
	long[] durationOfSort(Sort<Integer> aSort, Integer[] experimentList)	정렬 방법, 배열	실험시간	주어진 정렬 방법으로 배열을 정렬하여 걸린 시간을 데이터 증가 값별로 저장한 값들을 리턴.

Class	ExperimentManager			
Method	메소드 이름	파라미터 설명	리턴값	메소드 설명
	ExperimentManager()	없음	없음	실험에 필요한 정보값들을 초기화하도록 지시하는

				생성자 메소드
	ParameterSet parameterSet()	없음	실험에 필요한 정보값	실험에 사용되는 데이터의 정보값을 리턴하는 메소드
	void prepareExperimentLists()	없음	없음	실험에 사용될 데이터 리스트들을 생성하여 준비하는 메소드
	void setParameterSetWithDefaults()	없음	없음	실험에 필요한 데이터의 정보값을 초기화하는 메소드
	Integer[] experimentListOfOrder(ListOrder anOrder)	어떠한 데이터 리스트를 사용하는지 알려주는 enum	배열	입력받은 데이터 리스트에 따라 해당하는 데이터 리스트가 저장된 배열을 리턴하는 메소드
	void prepareExperiment(ParameterSet aParameterSet)	실험에 쓰일 데이터의 정보값	없음	실험에 쓰일 데이터를 생성하도록 지시하는 메소드
	long measuredResultForInsertionSortAt(int sizeStep)	실험의 증가값	배열	실험의 증가값에 해당하는 인덱스의 배열을 리턴한다
	long measuredResultForQuickSortAt(int sizeStep)	실험의 증가값	배열	실험의 증가값에 해당하는 인덱스의 배열을 리턴한다
	void performExperiment(ListOrder anOrder)	어떠한 데이터 리스트를 사용하는지 알려주는 enum	없음	실험을 실행한다. 삽입정렬과 퀵정렬에 해당하는 값을 별도의 배열에 저장한다.

Class	Timer			
Method	메소드 이름	파라미터 설명	리턴값	메소드 설명
	void start()	없음	없음	정렬을 시작할 때 나노초를 측정하는 메소드
	void stop()	없음	없음	정렬이 끝날 때 나노초를 측정하는 메소드
	long duration()	없음	정렬을 실행하는 데에 걸린 시간	정렬을 하는 데에 걸린 시간은 리턴하는 메소드

2 종합 설명서

- 프로그램을 실행하면 코드의 짜여진 순서에 따라 오름차순, 내림차순, 무작위 리스트순으로 정렬을 실행한다
- 정렬은 insertion sort 와 quick sort 순으로 진행된다.
- 정렬을 실행 한 후 정렬이 끝날 때까지 측정한 실행시간을 출력한다.
- 삽입 정렬과 퀵 정렬의 시간 차이를 알아본다.

2 프로그램 장단점 분석

- 장점
  - 데이터를 자동으로 생성하여 삽입 정렬과 퀵 정렬을 사용해볼 수 있다.
  - 오름차순, 내림차순, 무작위 로 생성된 데이터에 대해 어느 정렬이 더 효율적으로 정렬하는지 알 수 있다.
  - 같은 정렬 기능을 부모 클래스 sort 로 선언하여 사용하므로 인해 불필요한 코드를 작성하지 않아도 된다.
  - 객체를 생성하지 않아도 되는 클래스와 메소드를 final 키워드와 static 키워드를 이용해 사용해 볼 수 있다.
  - enum 클래스에서 메소드와 static 객체를 사용할 수 있다.

- 단점
  - 삽입 정렬의 경우는 내림차순, 퀵 정렬의 경우 내림차순, 오름차순의 경우 정렬 시간이 제일 길게 걸린다.
  - 상속 개념을 이해해야한다.
  - 프로그램 내부 메소드 간의 연결관계를 이해하는데 어렵다.

3 실행 결과 분석

1 입력과 출력

프로그램 실행

```
<< 정렬 성능 비교 프로그램을 시작합니다. >>

>> 2가지 정렬의 성능 비교: 삽입, 퀵 <<
```

입출력 1-1 - 오름차순 데이터를 사용하여 실행한 정렬 시간 측정

```
>오름차순데이터를 사용하여 실행한 측정
         <Insertion Sort>     <Quick Sort>
[ 1000]              7178          1253198
[ 2000]             11632           4974744
[ 3000]             17183         10848460
[ 4000]             28749         19538248
[ 5000]             28266         30022679
[ 6000]             32802         43132908
[ 7000]             38552         69222905
[ 8000]             43640         86537555
[ 9000]             48951        103185971
[10000]             54549        134069787
```

입출력 1-2 - 내림차순 데이터를 사용하여 실행한 정렬 시간 측정

```
>내림차순데이터를 사용하여 실행한 측정
         <Insertion Sort>     <Quick Sort>
[ 1000]           1121619          1327943
[ 2000]           4371446           5262727
[ 3000]           9853508         11887036
[ 4000]          16882581         23424151
[ 5000]          27513178         34610287
[ 6000]          43451598         47296370
[ 7000]          60073705         67235230
[ 8000]          73565860         93236638
[ 9000]          91621227        110340903
[10000]         117458861        131749904
```

입출력 1-3 - 무작위 데이터를 사용하여 실행한 정렬 시간 측정

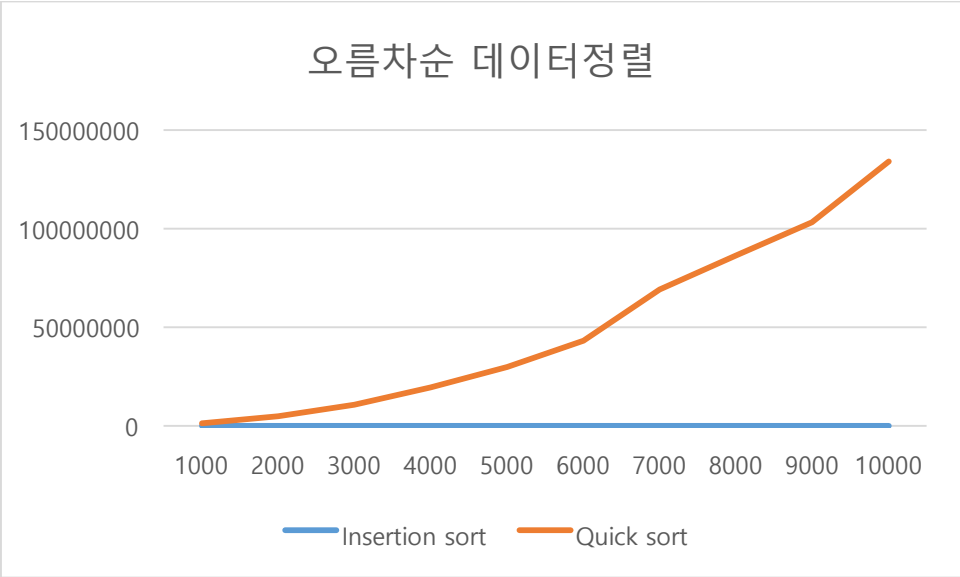
```
>무작위데이터를 사용하여 실행한 측정
        <Insertion Sort>     <Quick Sort>
[ 1000]          657068           122921
[ 2000]         2442799           203006
[ 3000]         5371681           315368
[ 4000]         9462402           441369
[ 5000]        14603489           555841
[ 6000]        21093390           688879
[ 7000]        29824699           813108
[ 8000]        40521365           935744
[ 9000]        49896361          1989670
[10000]        59347790          2437312
```

프로그램 종료

```
<< 정렬 성능 비교 프로그램을 종료합니다 >>
```

2 결과 분석

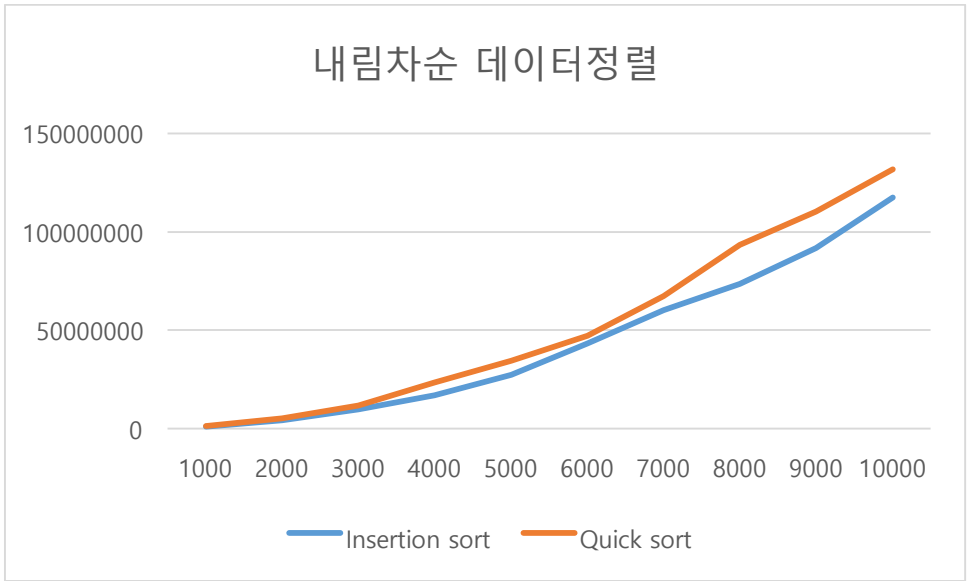
- 오름차순 데이터 정렬



- 오름차순 데이터 정렬 결과 삽입 정렬이 퀵 정렬에 비해 월등이 빠른 것을 확인할 수 있다.
- 이 정렬 프로그램은 정렬 대상 데이터를 오름차순으로 정렬하는 프로그램이므로, 이미 정렬 되어있는 것을 다시 정렬하는 것이다.

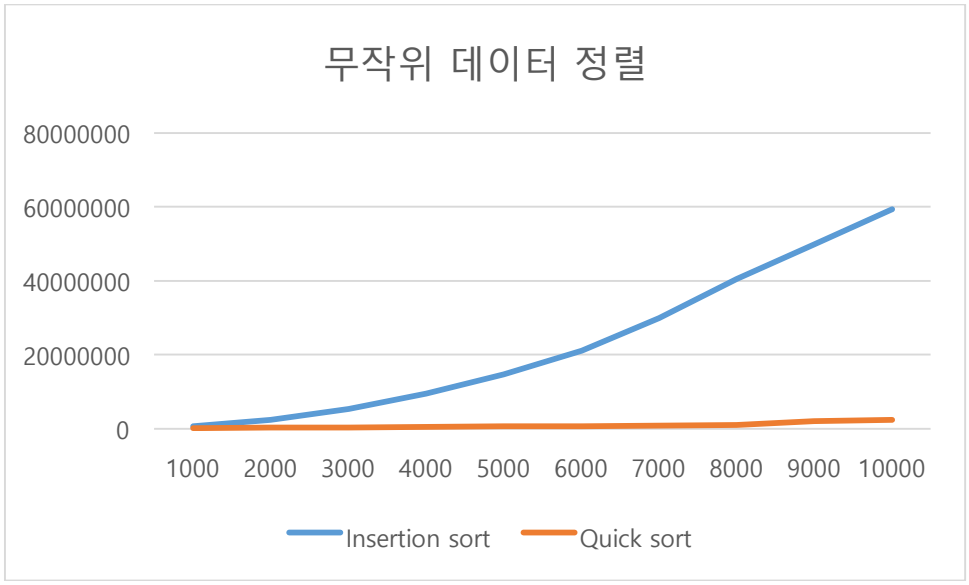
- 이미 정렬 되어있는 데이터를 다시 같은 방법으로 정렬하는 것은 퀵 정렬을 사용할 때에 매우 좋지 않은 효율을 나타냈다.

- 내림차순 데이터 정렬



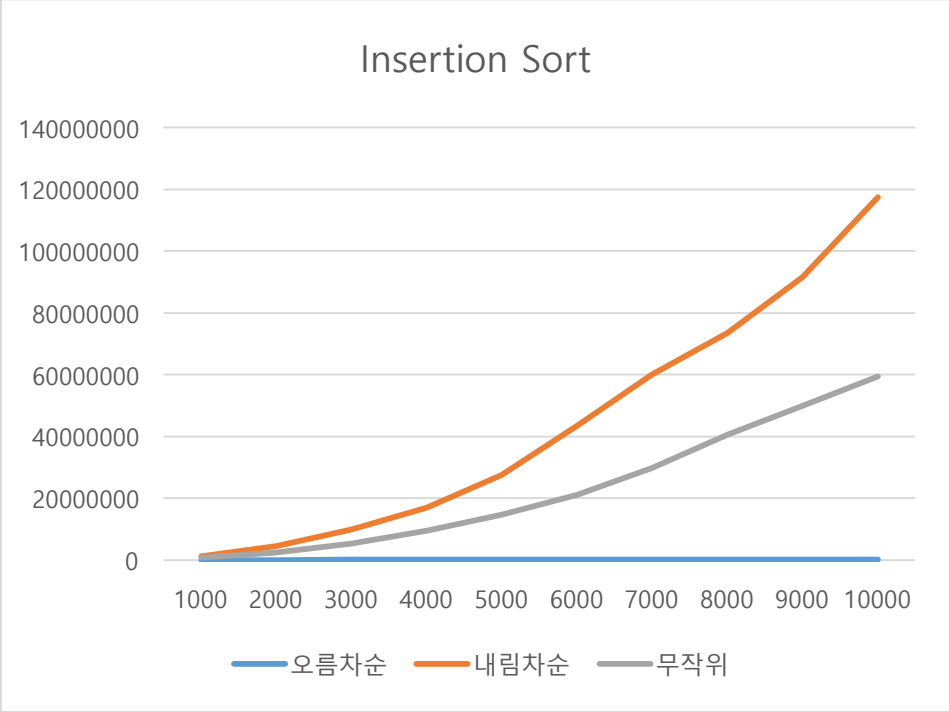
- 내림차순 데이터를 오름차순 데이터로 정렬하였을 때, 삽입 정렬이 퀵 정렬보다 약간 빠른 것을 확인할 수 있다.

- 무작위 데이터 정렬



- 무작위 데이터를 오름차순으로 정렬했을 때, 퀵 정렬이 삽입 정렬보다 월등하게 빠른 것을 확인할 수 있다.

- Insertion Sort



- 삽입 정렬을 오름차순, 내림차순, 무작위 순의 데이터들을 오름차순 리스트로 정렬한 결과 오름차순이 제일 빠르며, 그 다음으로 무작위, 내림차순 인 것을 확인할 수 있다.
- 이미 오름차순으로 정렬되어 있는 오름차순 데이터를 제외한다면, 무작위 데이터를 정렬할 때 삽입정렬이 빠른 것을 확인할 수 있다.

- Quick Sort

- 퀵 정렬을 오름차순, 내림차순, 무작위 순의 데이터들을 오름차순 리스트로 정렬한 결과, 무작위 순의 데이터를 오름차순으로 정렬하는 속도가 월등히 빠르며 오름차순과 내림차순 데이터를 정렬하는 것은 두 데이터 리스트다 비슷한 속도를 보이는 것을 확인할 수 있다.

- 결론
  - 오름차순 데이터를 정렬할 때에는 삽입 정렬이 가장 빠르다.
  - 내림차순 데이터를 정렬할 때에는 삽입 정렬이 근소한 차이로 빠르다.
  - 무작위순 데이터를 정렬할 때에는 퀵 정렬이 가장 빠르다.

  - 삽입 정렬은 오름차순, 무작위순, 내림차순 순으로 빠르다.
  - 퀵 정렬은 무작위순이 제일 빠르며 오름차순과 내림차순은 비슷한 속도를 보여준다.
  - 대부분의 데이터는 무작위 데이터를 오름차순으로 정렬해야 할 경우가 많으므로, 퀵 정렬이 삽입 정렬보다 유용하게 사용할 수 있다.