[제 7 주 실습]

정렬 – 성능비교

강 지 훈
jhkang@cnu.ac.kr

충남대학교 컴퓨터공학과

중간고사

- 일정
  - 10월 17일 목요일 저녁 7시
- 장소
  - 00반 5412
  - 01반 5411

[문제 7]
Sorting Test

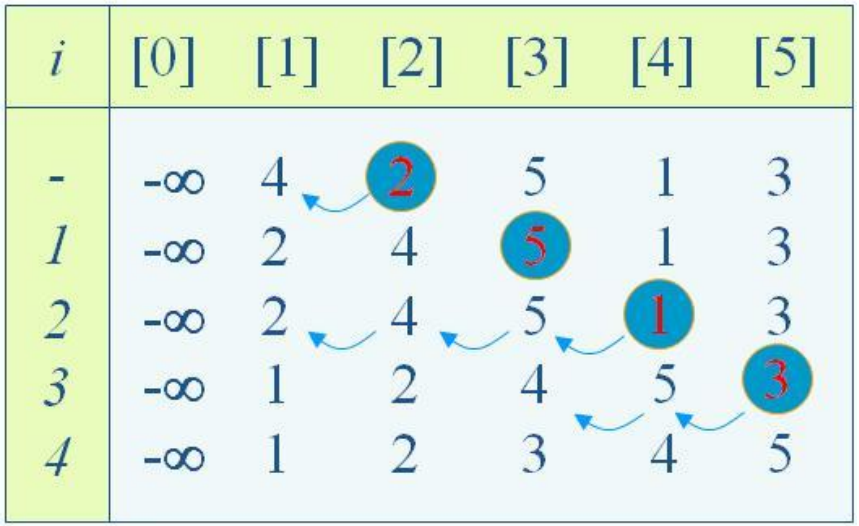
문제 개요

- Insertion, Quick Sort, heap Sort 알고리즘을 이해하고, 이를 구현한다.

- 구현한 Sorting알고리즘의 성능을 측정하는 프로그램을 작성하고 이를 분석한다.

Insertion Sort

알고리즘



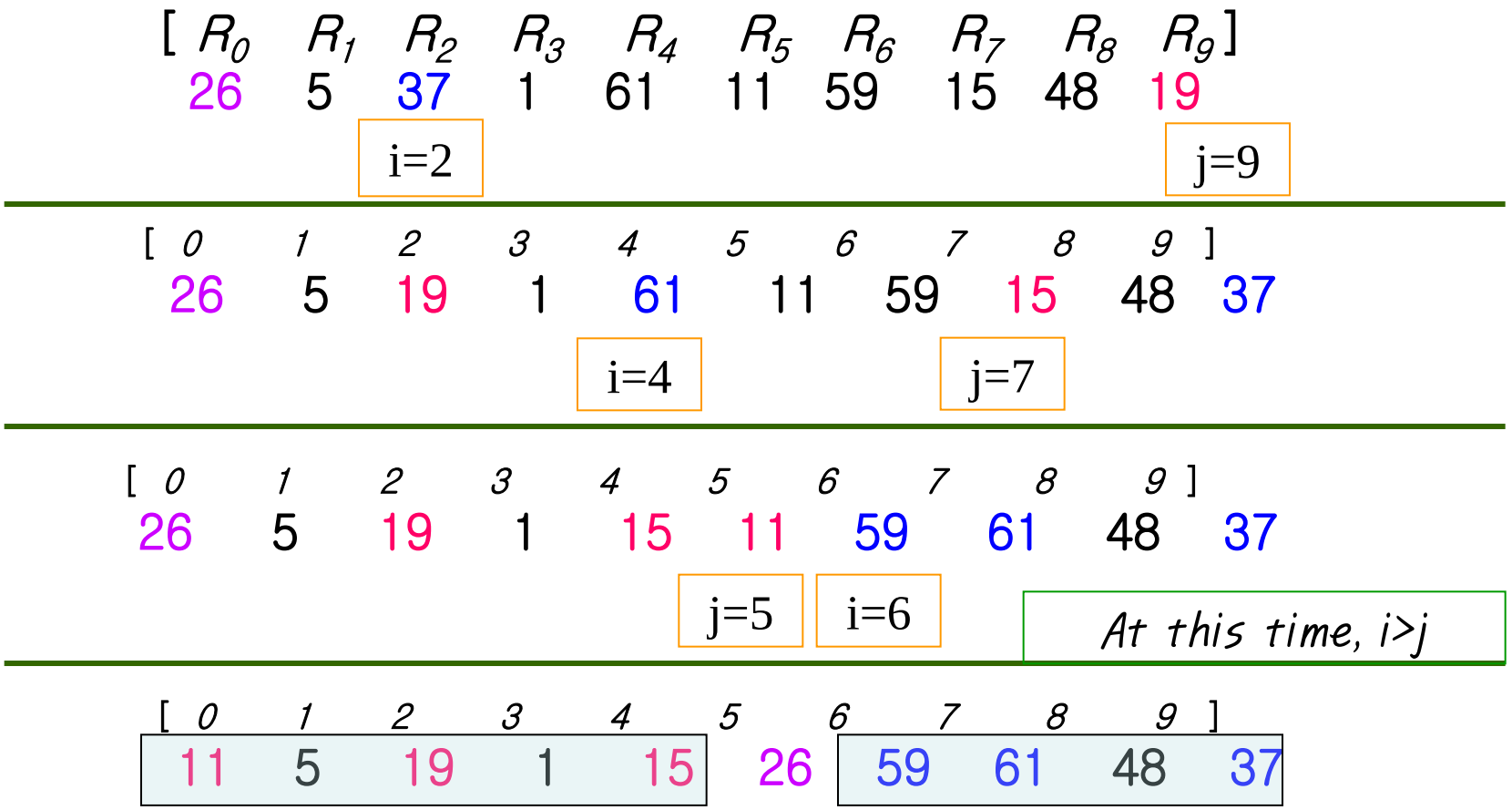
i	[0]	[1]	[2]	[3]	[4]	[5]
-	-∞	4	2	5	1	3
1	-∞	2	4	5	1	3
2	-∞	2	4	5	1	3
3	-∞	1	2	4	5	3
4	-∞	1	2	3	4	5

(R_0, R_1)
(R_0, R_2, R_1)
(R_0, R_2, R_1, R_3)
$(R_0, R_4, R_2, R_1, R_3)$
$(R_0, R_4, R_2, R_5, R_1, R_3)$

- 먼저 4,2를 비교한다. 뒤의 2가 작으므로 앞으로 이동.
- 다음 5도 앞의 값 4와 비교한다. 5가 크므로 이동 없음.
- 1, 3도 같은 방법으로 비교하여 앞의 값이 작을 때까지 이동
- 0번째 배열은 가장 작은 값을 넣어둔다
  (unsigend int의 데이터일 경우 -1)

Quick Sort

알고리즘



Heap Sort

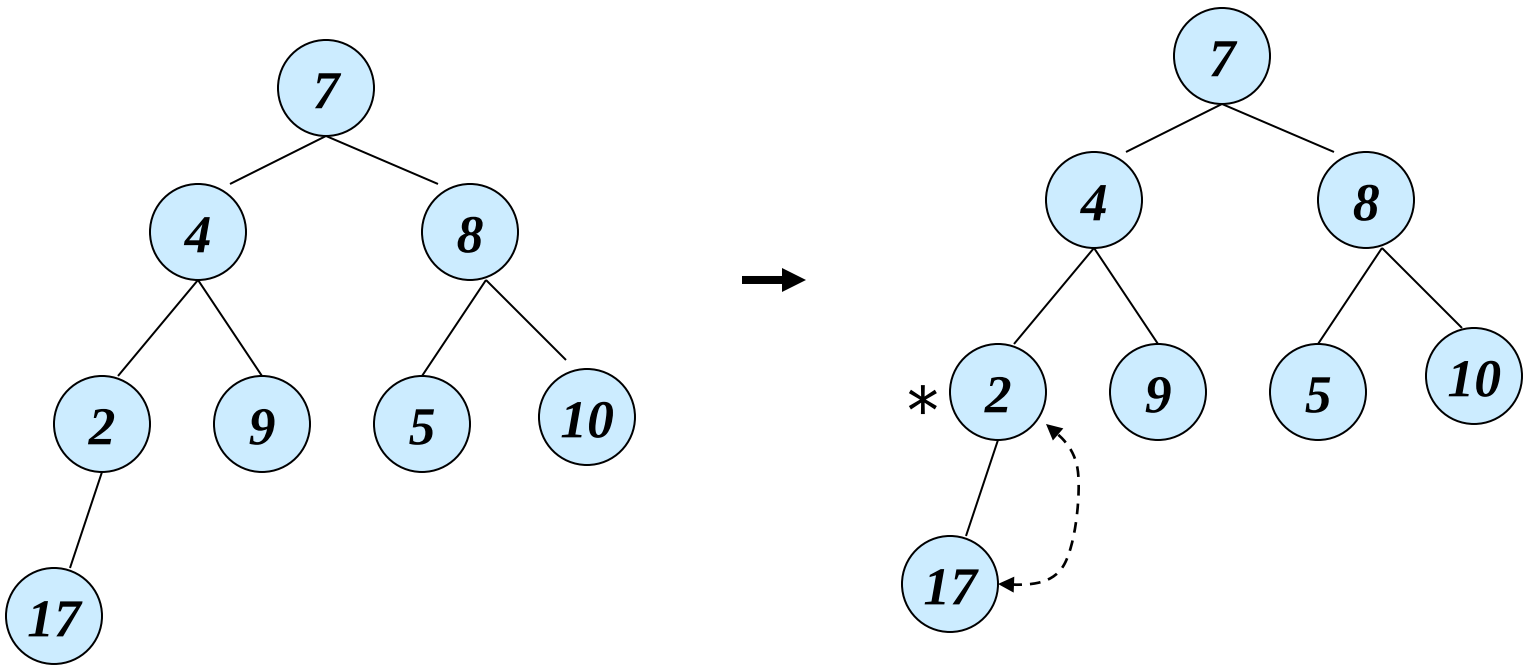
알고리즘

- Heap Sort 알고리즘은 크게 두부분으로 나눌 수 있다.
- 초기화
  - 정렬하려는 데이터배열을 입력받은 후 Heap구조로 초기화 해주어야 한다.
- Sorting(Delete & Adjust)
  - Heap구조로 초기화된 데이터배열을 Delete&Adjust를 통해 정렬된 데이터배열로 Sorting한다.

Heap Sort

초기화 알고리즘

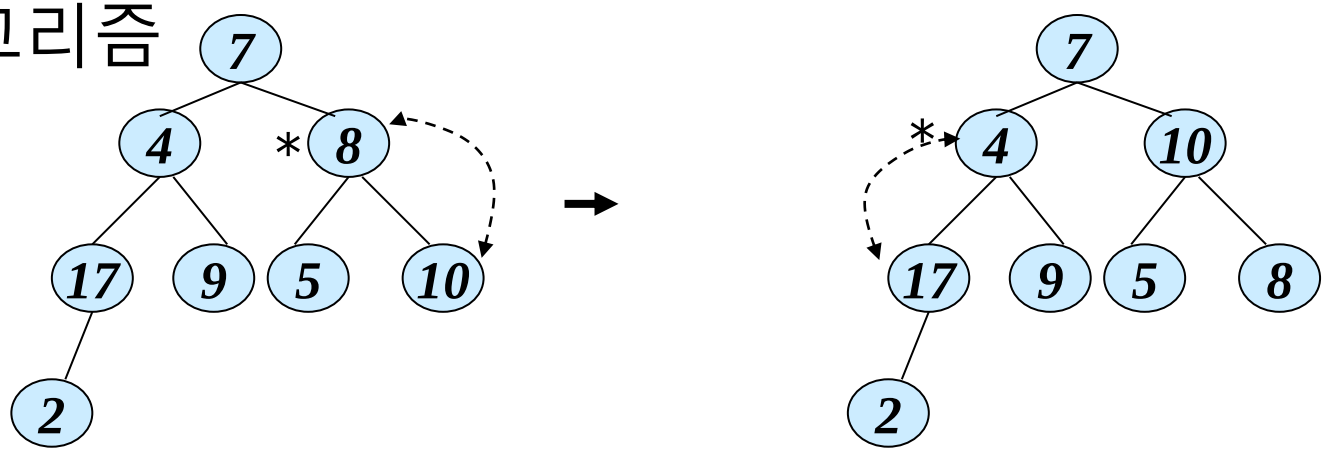
- 무작위로 입력된 값으로 heap을 만든다.



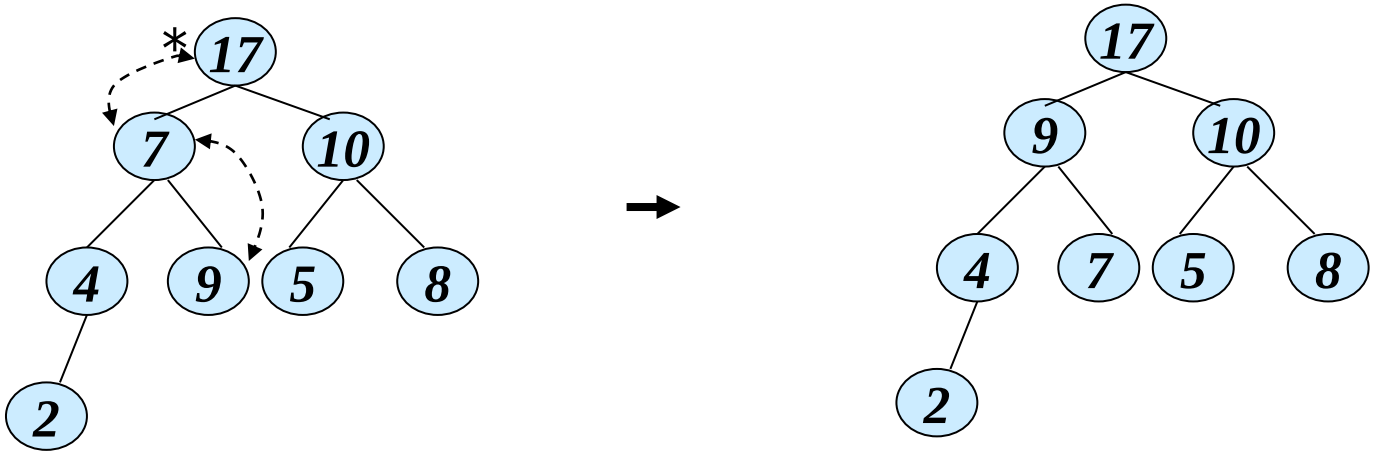
가장 마지막 node의 부모와 비교하여 자식이 더 크면 교환

Heap Sort

초기화 알고리즘



왼쪽자식과 오른쪽자식을 비교하여 더 큰자식과 부모와 비교

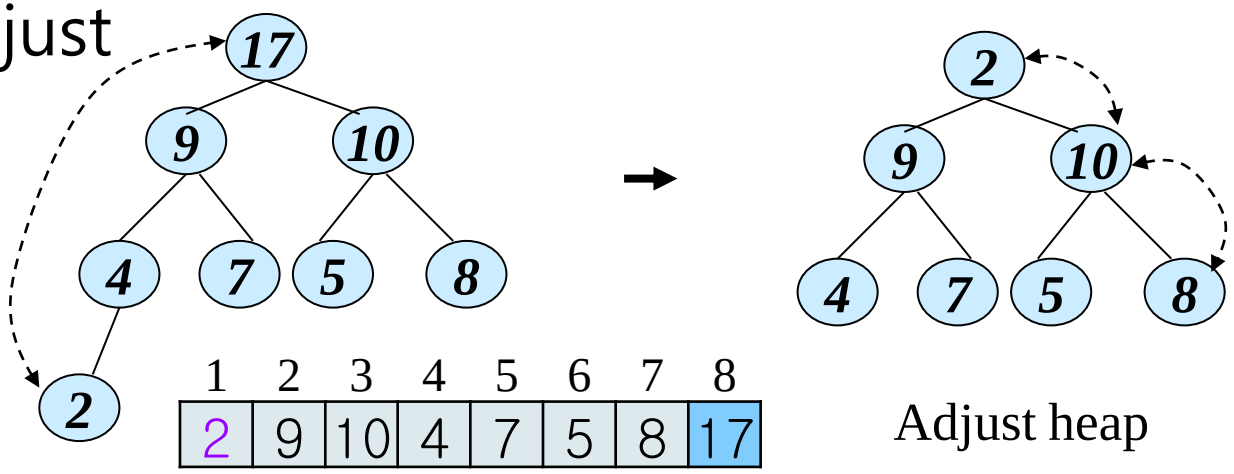


자식과 자리가 바뀌었으면 바뀐곳의 자식 들과 비교한다.

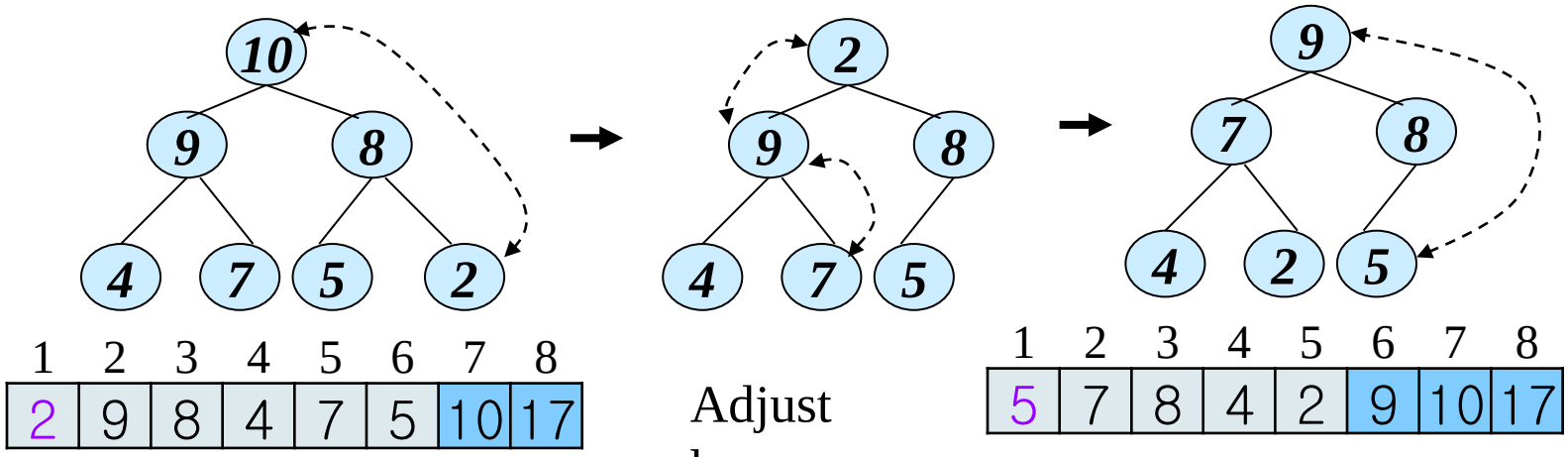
Heap Sort

Delete & Adjust

- delete한 값은 뒤로 보낸다

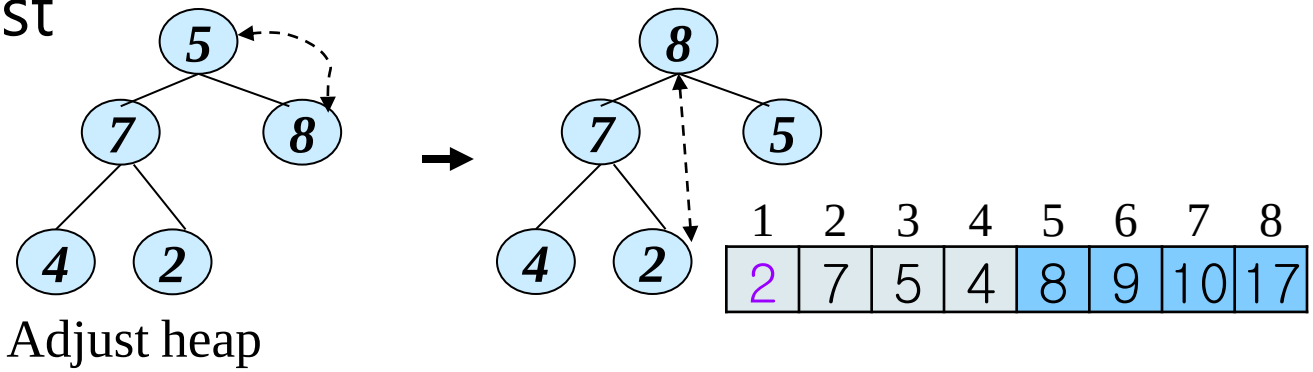


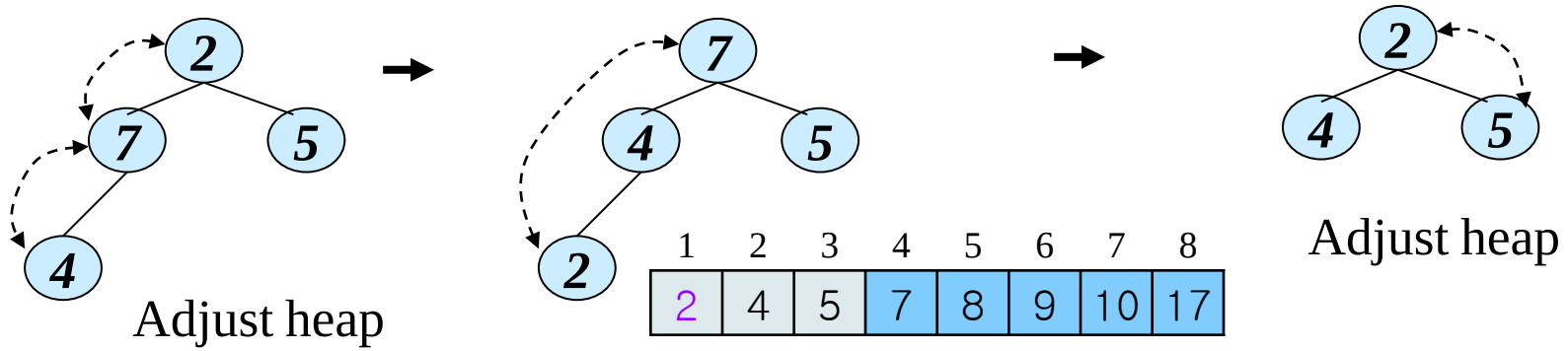
Heap을 만드는 algorism을 이용하기위해 가장 마지막 node를 root로 넣고 원래 root를 삭제하고 자식과 비교한다.

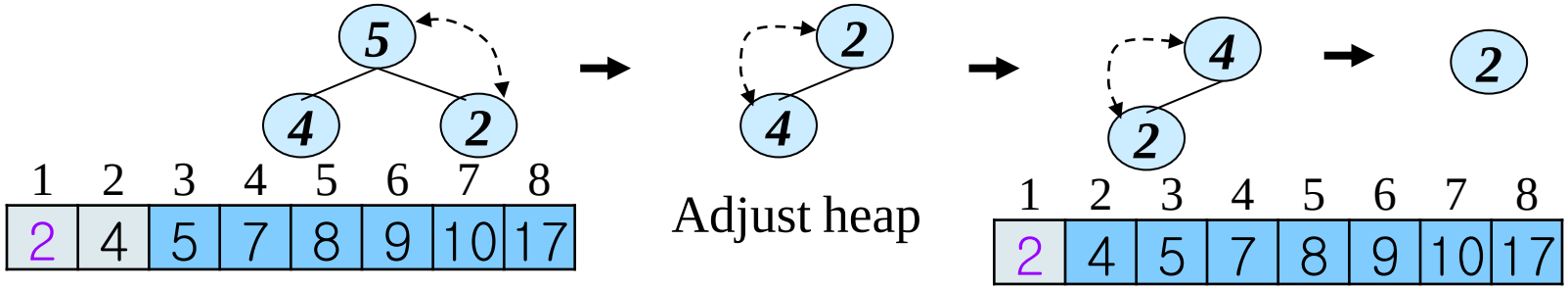


Heap Sort

Delete & Adjust







성능측정

- Insertion Sort, Quick Sort에 대하여 성능을 측정한다.
- 각각의 Sorting알고리즘에 대하여 Sequential, Reverse, Random Data의 경우를 측정한다.
- 각각의 측정에 대하여 Data의 Size를 달리하여 측정한다.

	Sequaltial data						Reverse data						Random data					
Data size ➔	10	20	30	40	50	60	10	20	30	40	50	60	10	20	30	40	50	60
insertion sort																		
Quick sort																		
Heap sort																		

- 위 표와 같이 측정한다. 단, data size에 따른 측정횟수는 위와 같지 않다.

❑출력의 예

```
[1]sequential Data
[2]Reverse Data
[3]Random Data
[4]End
 Select a Sort >>
2
=REVERSE DATA======================
DataSize        Insertion       Quick           Heap
200             27203.004       10256.0185      3929.5515
400             105596.5385     35563.915       11796.12
600             234021.7955     74870.7775      23003.656
800             409290.503      127627.217      35702.3125
1000            639664.8285     194671.549      48083.741
1200            915287.932      275198.8895     61270.9855
1400            1241029.521     369750.6935     74027.6435
1600            1617239.9555    478384.981      86964.833
1800            2043300.171     599316.5655     99290.287
2000            2519252.9365    735209.6795     120485.9165
[1]sequential Data
[2]Reverse Data
[3]Random Data
[4]End
```

이 과제에서 필요한 객체는?

- AppView
- AppController
  - DataGenerator
  - SortTester
- SortTester
  - InsertionSort
  - QuickSort
  - HeapSort

AppControllor의 공개 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public void run()

AppView의 공개 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public AppView()
  - public void showGenerateSequentialMsg()
  - public void showGenerateReverseMsg()
  - public void showGenerateRandomMsg()
  - public void showMsg(String aString)
  - public void showErrorMsg()
  - public void showStartSortingTest()
  - public int inputNumOfDataType()

DataGenerator의 멤버 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public DataGenerator()
  - public void generateSequentialData(int size)
  - public void generateReverseData(int size)
  - public void generateRandomData(int size)
  - public int [] getData(int size)

SortTester의 멤버 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public SortTester()
  - public double testInsertionSort(int [] data, int dataSize)
  - public double testQuickSort(int [] data, int dataSize)
  - public double testHeapSort(int [] data, int dataSize)

InsertionSort의 멤버 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public InsertionSort()
  - public void sort(int [] _data, int size)

- 본 실습 자료에서 따른 구현에 대한 언급은 하지 않으며, 지난 실습 및 수업에서 배운 내용을 토대로 작성한다.

QuickSort의 멤버 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public QuickSort()
  - public void sort(int [] _data, int size)

- 본 실습 자료에서 따른 구현에 대한 언급은 하지 않으며, 지난 실습 및 수업에서 배운 내용을 토대로 작성한다.

HeapSort의 멤버 함수는?

- 사용자에게 필요한 함수 (Public Functions)
  - public HeapSort()
  - public void sort(int [] _data, int size)

- 본 실습 자료에서 따른 구현에 대한 언급은 하지 않으며, 지난 실습 및 수업에서 배운 내용을 토대로 작성한다.

Class “AppControllor“

AppControllor– 비공개 인스턴스 변수

```
public class AppControllor {
    private static final int MAX_DATA_SIZE = 2000;
    private static final int DATA_TERM = 200 ;

    private AppView _appView;

    private int [] _data;
    private DataGenerator _dataGenerator;
    private SortTester _sortTester;

    private double _insertionSortDuration;
    private double _QuickSortDuration;
    private double _HeapSortDuration;

    private int _sortType;
```

AppControllor 의 공개 함수 run()의 구현

```
public void run(){
    this._appView = new AppView();
    _dataGenerator = new DataGenerator();
    this._sortTester = new SortTester();

    this._sortType = this._appView.inputNumOfDataType();
    while (this._sortType != 4) {
        if(_sortType == 1) {
            // Sorted Data Sorting Test
            this._dataGenerator.generateSequentialData(MAX_DATA_SIZE);
            this._appView.showGenerateSequentialMsg();
        }
        else if(_sortType == 2) {
            this._dataGenerator.generateReverseData(MAX_DATA_SIZE);
            this._appView.showGenerateReverseMsg();
        }
        else if(_sortType == 3) {
            this._dataGenerator.generateRandomData(MAX_DATA_SIZE);

        }
        else if(_sortType == 4) {
            this._sortType = -1;
            break;
        }
        else {
            this._appView.showErrorMsg();
            this._sortType = -1;
        }
```

AppControllor 의 공개 함수 run()의 구현

```
    if (this._sortType > 0) {
        this._appView.showStartSortingTest();

        /* 메모리 생성 및 테스트의 안정성을 위하여 가장 첫 성능 측정을 미리 한번 진행한다 */
        this.doTest(DATA_TERM);

        // 실제 테스트 진행
        for (int dataSize = DATA_TERM; dataSize <= MAX_DATA_SIZE; dataSize += DATA_TERM) {
            this.doTest(dataSize);
            String showString = dataSize + "\t\t";
            this._appView.showMsg(showString);
            showString = this._insertionSortDuration + "\t\t";
            this._appView.showMsg(showString);
            showString = this._QuickSortDuration + "\t\t";
            this._appView.showMsg(showString);
            showString = this._HeapSortDuration + "\t\t";
            this._appView.showMsg(showString);
        }
    }
    this._sortType = this._appView.inputNumOfDataType();
}
```

AppControllor 의 Private Method

AppControllor 의 Private Member function의 사용법

```
private void doTest(int dataSize)
{
        this._insertionSortDuration = 0;
        // 삽입정렬 테스트 - 정확성을 위해 여러번 실행하고 실행 횟수만큼 나눈다.
        for (int i = 0; i < MAX_DATA_SIZE; i++)  {
                _data = this._dataGenerator.getData(dataSize);
                this._insertionSortDuration += this._sortTester.testInsertionSort(_data,dataSize);
        }
        this._insertionSortDuration = this._insertionSortDuration / MAX_DATA_SIZE;

        this._QuickSortDuration = 0;
        // 퀵정렬 테스트
        for (int i = 0; i < MAX_DATA_SIZE; i++) {
                _data = this._dataGenerator.getData(dataSize);
                this._QuickSortDuration += this._sortTester.testQuickSort(_data,dataSize);
        }
        this._QuickSortDuration = this._QuickSortDuration / MAX_DATA_SIZE;

        this._HeapSortDuration = 0;
        // 힙정렬 테스트
        for (int i = 0; i < MAX_DATA_SIZE; i++) {
                _data = this._dataGenerator.getData(dataSize);
                this._HeapSortDuration += this._sortTester.testHeapSort(_data, dataSize);
        }
        this._HeapSortDuration = this._HeapSortDuration / MAX_DATA_SIZE;
}
```

Class “AppView“

AppView – 비공개 인스턴스 변수

```
import java.util.Scanner;

public class AppView {
    private Scanner _scanner;
```

AppView의 Public Method

- AppView 의 Public Member function의 사용법과 구현
  - public AppView()
    - 생성자
  - public void showGenerateSequentialMsg()
    - "=SEQUENTIAL DATA=" 를 화면에 내보낸다.
  - public void showGenerateReverseMsg()
    - "=REVERSE DATA=" 를 화면에 내보낸다.
  - public void showGenerateRandomMsg()
    - “=RANDOM DATA=“를 화면에 내보낸다.
  - public void showErrorMsg()
    - "input wrong number"를 화면에 내보낸다.
  - public void showStartSortingTest()
    - "DataSize₩tInsertion₩tQuick₩t₩tHeap"를 화면에 내보낸다.

AppView의 Public Method

- AppView 의 Public Member function의 사용법과 구현
  - public void showMsg(String aString)
    - String 메시지를 하나 받아 화면에 내보낸다.
    - System.out.println이 아닌 System.out.print를 사용하여야 한다.
  - public int inputNumOfDataType()
    - "[1]sequential Data ₩n[2]Reverse Data ₩n[3]Random Data₩n[4]End"를 화면에 내보낸다.
    - " Select a Sort >> "를 화면에 내보낸다.
    - 원소 개수 하나를 입력 받아 반환한다.

Class “DataGenerator”

실험 데이터 생성

난수를 이용하자 !

- 난수 (Random Number)
  - 아무 규칙 없이, 무작위적으로 얻어지는 수
  - 하나의 난수를 얻고 나서, 그 다음 난수를 얻을 때 이전 난수와 어떠한 연관관계도 없이 무작위적으로 얻어져야 한다.

❑난수를 이용한 실험 데이터 생성 [O]

```
import java.util.*;
    ......
    Random random = new Random();
    int data[MaxSize];
    int i = 0 ;

    while ( i < MaxSize ) {
        data[i] = random.nextInt( MAX_SIZE );
        i++ ;
    }
    ......
```

실험 데이터를 무작위로 MaxSize만큼 생성하여 배열 "data[]"에 저장하려고 한다.

난수를 이용한 실험 데이터 생성 [4]

```
import java.util.*;
    ......
    Random random = new Random();
    int data[MaxSize];
    int i = 0 ;

    while ( i < MaxSize ) {
        data[i] = random.nextInt( MAX_SIZE );
        i++ ;
    }
    ......
```

난수 생성을 하는 객체를 생성한다.

"nextInt()"
정수형 난수를 얻는다.

"MAX_SIZE"
0~(Maxsize-1) 사이의 정수를 얻게 해 준다.

DataGenerator– 비공개 인스턴스 변수

```
public class DataGenerator {
    private int [] _dataArray;
    private int _dataSize;
```

Member Functions의 구현

- DataGenerator의 Public Member function의 구현
  - public DataGenerator()
    - _dataArray, _dataSize를 초기화
  - public void generateSequentialData(int size)
  - public void generateReverseData(int size)
  - public void generateRandomData(int size)
    - _dataArray를 size만큼 초기화
    - _dataArray의 0번째를 -1로 초기화
      - Insertion Sort를 위하여 가장 작은 수로 0번째를 초기화한다.
    - _dataSize에 size를 저장
    - 원하는 방식으로 테스트 할 데이터를 생성하여 _dataArray에 값을 넣는다.

Member Functions의 구현

- DataGenerator의 Public Member function의 구현
  - public int [] getData(int size)
    - 복사할 size만큼의 배열 copyArray를 생성한다.
    - 생성된 data배열을 _dataArray로부터 size만큼 '복사'한다.
    - 복사된 copyArray를 리턴한다.

    - 왜 _dataArray 를 바로 사용하지 않고, 복사를 해야할까?

Class “SortTester”

수행시간 측정 방법

시간 비교

- System.nanoTime()
  - 자바에서 현재 시간을 얻어오는 시스템 메소드
  - 단위 : nanosecond
  - Long 타입으로 저장
  - 예시

```
long start = System.nanoTime;
함수 실행();
long end = System.nanoTime();
double time = (double)(end -start);
System.out.println(" 수행 시간 = " + (int)time + " nano sec");
```

  - 자료의 입력 크기가 작을 경우 현재 컴퓨터의 성능이 좋아 millisecond 단위로는 확인이 불가능하여 JDK 1.5이후 추가됨

Member Functions의 구현

- SortTester의 Public Member function의 사용법
  - public SortTester()
    - SortTester의 생성자
  - public double testInsertionSort(int [] data, int dataSize)
    - insertionSort를 실행
  - public double testQuickSort(int [] data, int dataSize)
    - QuickSort를 실행
  - public double testHeapSort(int [] data, int dataSize)
    - HeapSort를 실행

SortTester – 비공개 인스턴스 변수

```
public class SortTester {
    private InsertionSort _insertionSort;
    private QuickSort _quickSort;
    private HeapSort _heapSort;
```

Member Functions의 구현

- SortTester의 Public Member function의 구현
  - public SortTester()
    - _insertionSort, _quickSort, _heapSort를 초기화
  - public double testInsertionSort(int [] data, int dataSize)

```
double insertTime = 0;
long start, end;

start = System.nanoTime();//currentTimeMillis();
this._insertionSort.sort(data, dataSize);
end = System.nanoTime();//.currentTimeMillis();
insertTime = (double) (end - start);

return insertTime;
```

  - public double testQuickSort(int [] data, int dataSize)
  - public double testHeapSort(int [] data, int dataSize)
    - testInsertionSort를 참고하여 작성

Class “InsertionSort”
Class “QuickSort”
Class “HeapSort”

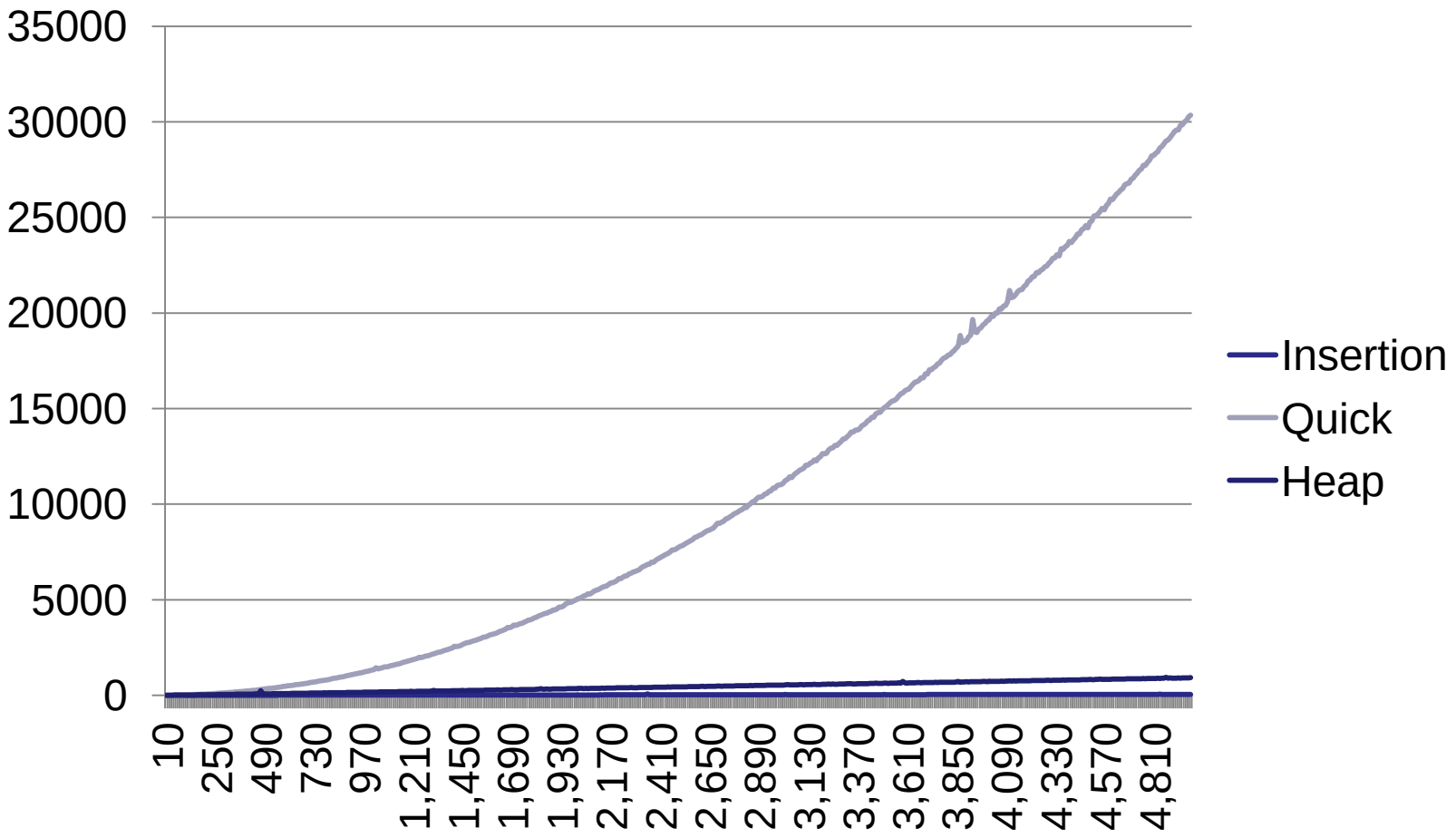
Member Functions의 구현

- InsertionSort, QuickSort, HeapSort의 Public Member function의 구현
  - http://winslab.cnu.ac.kr/lecture/2013/autumn/ds/slides/lec/DS2[02-2]Sorting.pdf
  - http://winslab.cnu.ac.kr/lecture/2013/spring/ds/slides/lec/[DS1-08]Sorting.pdf
  - http://winslab.cnu.ac.kr/lecture/2013/spring/ds/slides/prac/DS1_11.pdf

  - 구현 시 위의 자료를 참고하여 작성 할 것
  - 필요 시 새로운 함수를 생성하여 작성하여도 무관함

성능측정 결과

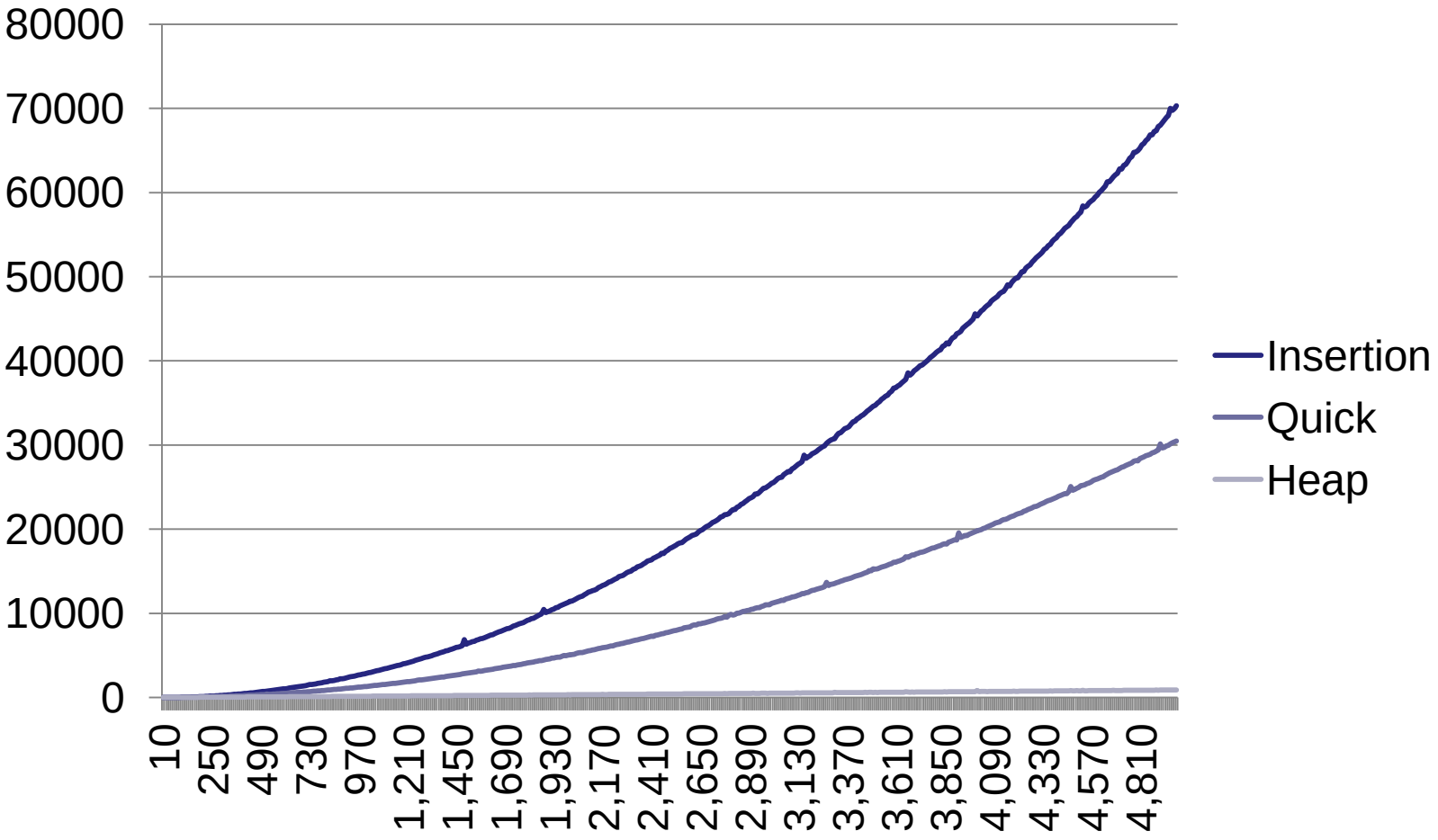
Sequential Data (Data:0~5000, Term:10)



정렬 성능 비교

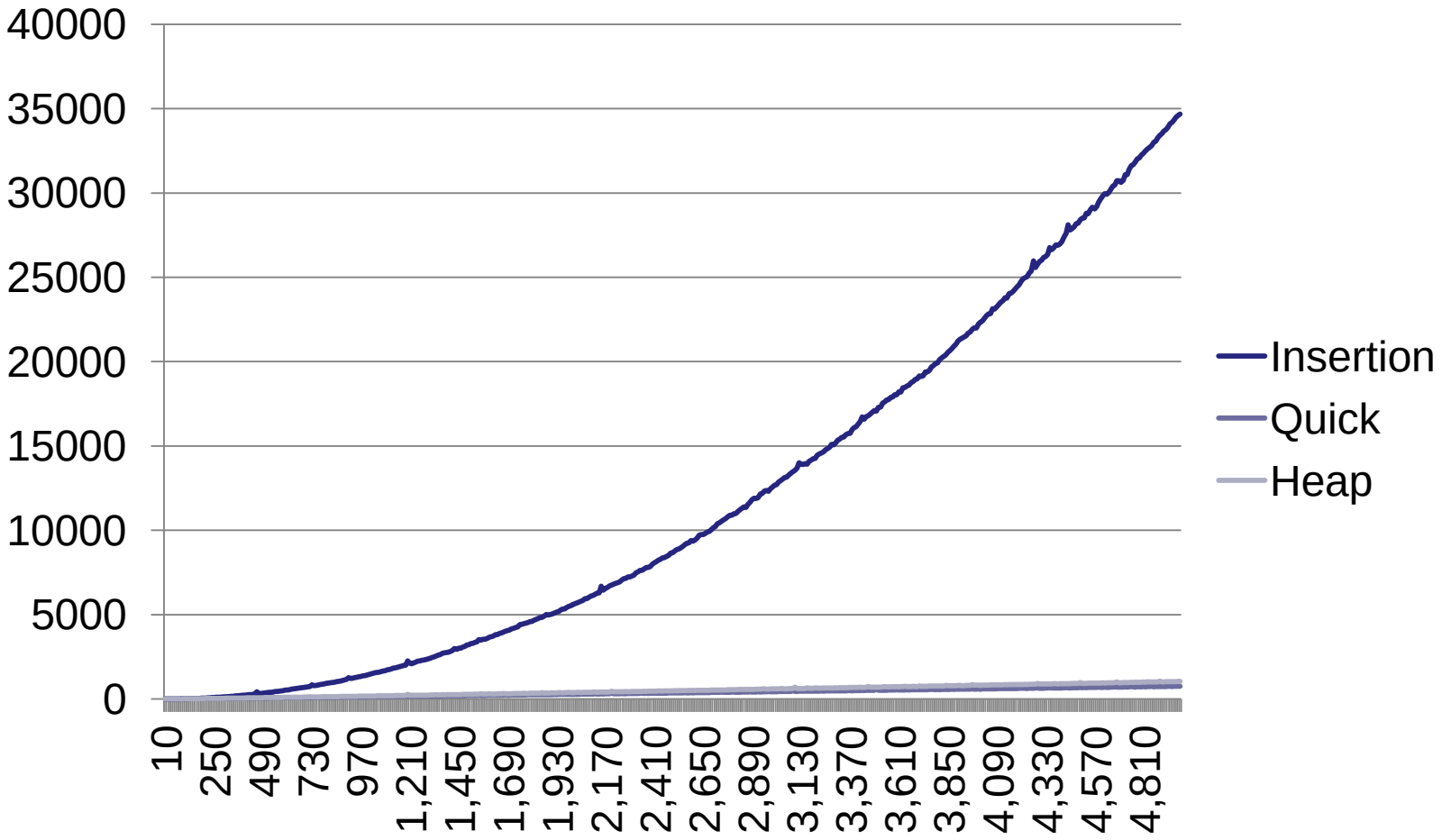
성능측정결과

Reverse Data (Data:0~5000, Term:10)



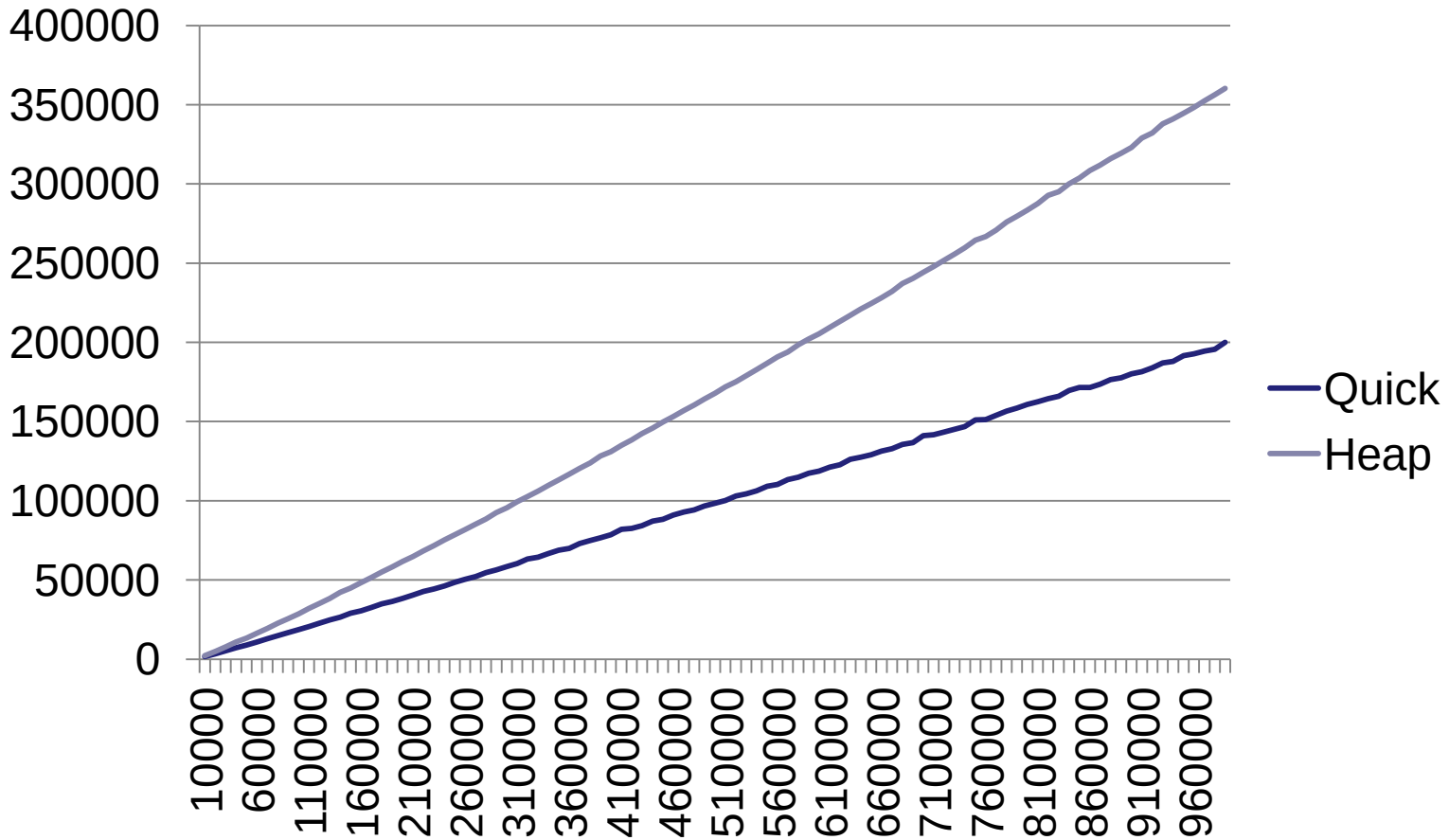
성능측정결과

Random Data (Data:0~5000, Term:10)



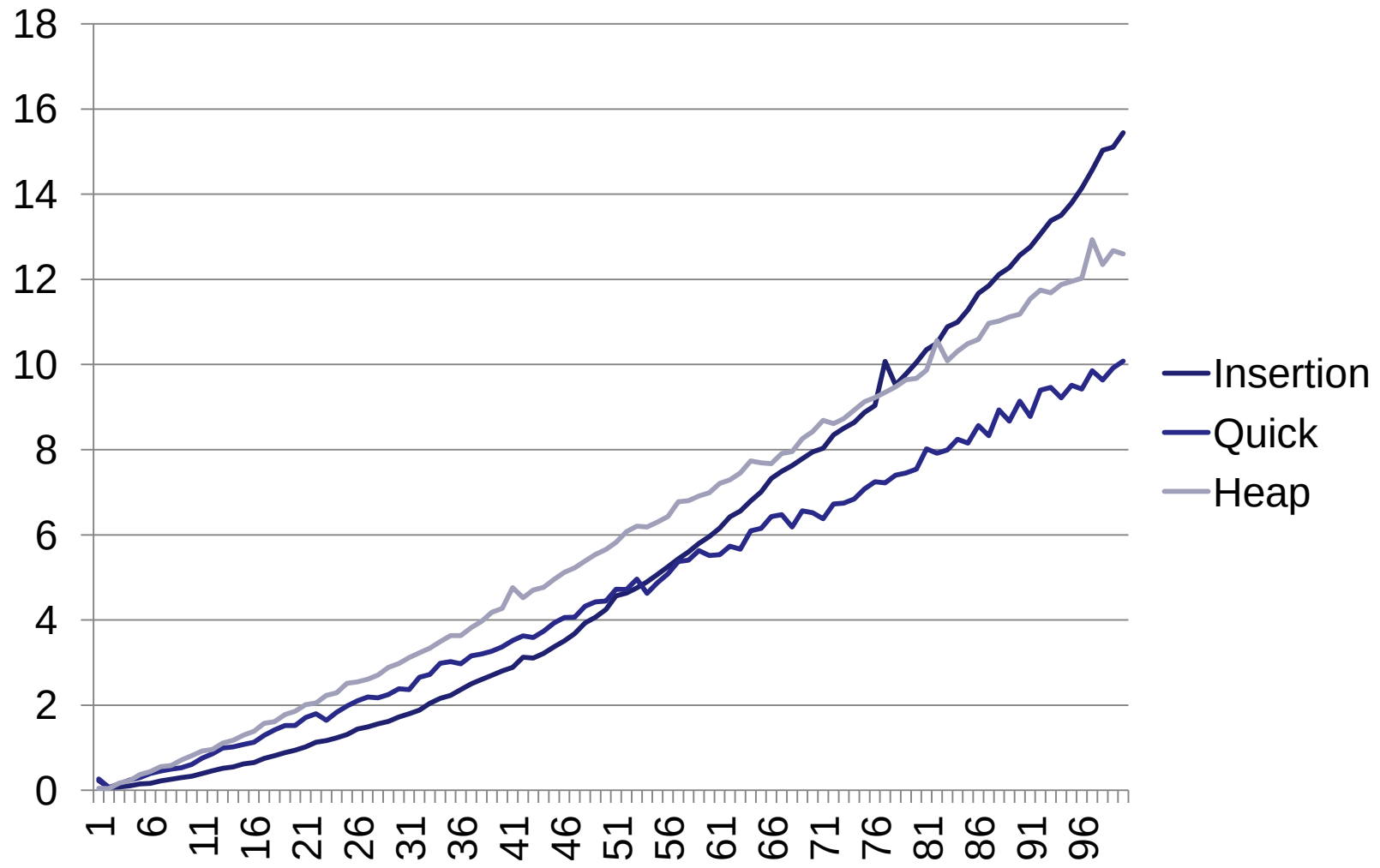
성능측정결과

- Random Data (Data:0~1000000, Term:10000)
  - Quick, Heap Sort만 실행



성능측정결과

Random Data (Data:0~100, Term:1)



정렬 성능 비교

성능측정 결과분석

- 정렬할 데이터의 특성에 따라 각각의 Sorting 알고리즘이 다른 결과가 나온다.
- 각각의 Sorting알고리즘의 데이터의 특성에 따른 시간복잡도를 계산하고, 보고서에 첨부할 것
  - insertion sort
    - Worst Case : $O(n^2)$
    - Best Case : O(n)
  - Quick sort
    - Worst Case : $O(n^2)$
    - Best Case : O(n·logn)
  - Heap sort
    - O(n·logn)

데이터의 특징에 따라 각각의 정렬알고리즘에 시간복잡도가 적용된다.

데이터크기가 작을 경우 Insertion Sort가 빠르다.

[문제 7] 요약

- 성능측정
  - 각 Sorting에 대한 성능 측정 결과를 확인한다.

과제 제출

과제 제출

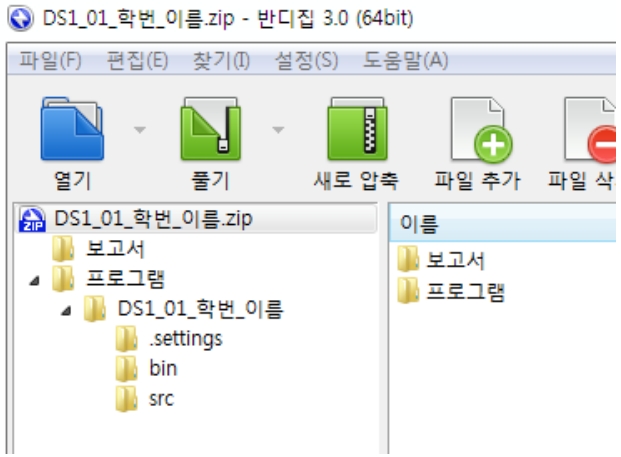
- pineai@cnu.ac.kr
  - 메일 제목 : [0X]DS2_07_학번_이름
    - 양식에 맞지 않는 메일 제목은 미제출로 간주됨
    - 앞의 0X는 분반명 (오전10시 : 00반 / 오후4시 : 01반)

- 제출 기한
  - 10월 22일(화) 23시59분까지
  - 시간 내 제출 엄수

  - 제출을 하지 않을 경우 0점 처리하고, 숙제를 50% 이상 제출하지 않으면 F 학점 처리하며, 2번 이상 제출하지 않으면 A 학점을 받을 수 없다.

과제 제출

- 파일 이름 작명 방법
  - DS2_07_학번_이름.zip
  - 폴더의 구성
    - DS2_07_학번_이름
      - 프로그램
        - 프로젝트 폴더 / 소스
        - 메인 클래스 이름 : DS2_07_학번_이름.java
      - 보고서
        - 이곳에 보고서 문서 파일을 저장한다.
        - 입력과 실행 결과는 화면 image로 문서에 포함시킨다.
        - 문서는 pdf 파일로 만들어 제출한다.



보고서 작성 방법

- 겉장
  - 제목: 자료구조 실습 보고서
  - [제xx주] 숙제명
  - 제출일
  - 학번/이름
- 내용
  1. 프로그램 설명서
     1. 주요 알고리즘 /자료구조 /기타
     2. 함수 설명서
     3. 종합 설명서 : 프로그램 사용방법 등을 기술
  2. 구현 후 느낀 점 : 요약의 내용을 포함하여 작성한다.
  3. 실행 결과 분석
     1. 입력과 출력 (화면 capture : 실습예시와 다른 예제로 할 것)
     2. 결과 분석

     -------- 표지 제외 3장 이내 작성 --------
  4. 소스코드 : 화면 capture가 아닌 소스를 붙여넣을 것

     소스는 장수 제한이 없음.

[제 7 주 실습] 끝