

11

자료구조 2

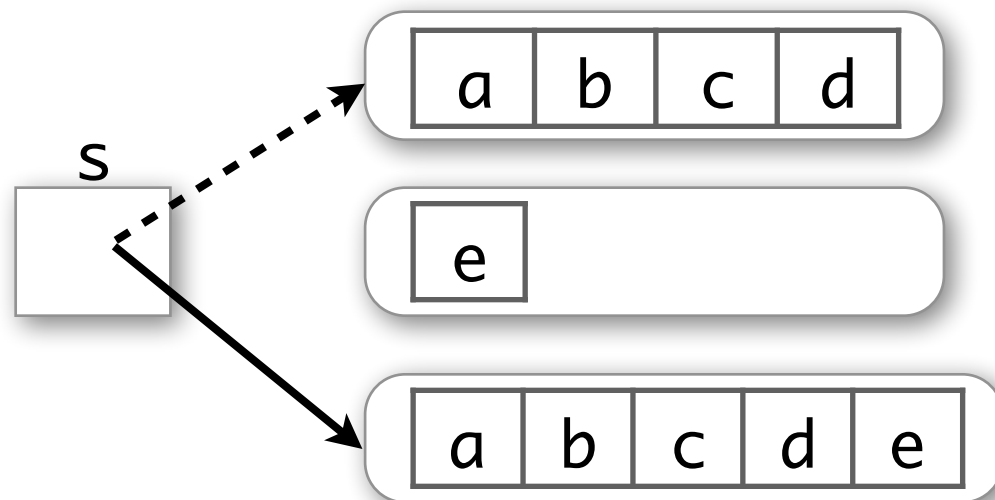
강의 내용

- 데이터를 저장 및 관리하기 위한 다양한 자료구조
 - 문서, 파일, 통칭(Generic)을 이용한 값 모음(Collection)
 - 예외 처리
- 자료구조에 자주 쓰이는 연산: 탐색과 정렬
 - 효율적인 탐색: 이진 탐색
 - 간단하지만 비효율적인 정렬: 삽입 정렬
 - 좀 더 복잡하지만 효율적인 정렬: 병합 정렬
- 효율적이란? 프로그램 계산 복잡도

문자열(String)은 변경불가

- 문자열은 한 번 만들어지면 변경할 수 없다.
- 연산을 적용하면 새로운 문자열을 만드는 것이다.
- 변경을 원한다면 `java.lang.StringBuffer` 를 사용하라.

```
String s = "abcd";  
s = s + "e";
```



문자열 장난 첫 번째, 토큰으로 나누기

- 많은 경우 문자열을 의미 있는 것들로 나누고 싶을 때가 많다.
 - 예, "910359,이육세,남"
→ "910359" "이육세" "남"
- 토큰 (token) 또는 어휘 (token)
 - 의미 있는 단어, 즉, 문자열
 - 분리자(delimiter)로 토큰이 나열된 자료가 많다.

java.util.StringTokenizer

○ 문자열 토큰화 라이브러리

```
String s = "910359,이육세,남";  
StringTokenizer t = new StringTokenizer(s, ",");  
t.nextToken(); // "910359"  
t.nextToken(); // "이육세"  
t.nextToken(); // "남"  
t.nextToken(); // NoSuchElementException
```

분리자

- 분리자를 여러 개 지정 가능

```
String s = "$13.46";  
StringTokenizer t = new StringTokenizer(s, "$.");  
t.nextToken(); // "13"  
t.nextToken(); // "46"  
t.nextToken(); // NoSuchElementException
```

다른 유용한 메소드

<code>class StringTokenizer</code>	
<u>생성자</u>	
<code>new StringTokenizer (String text, String delim)</code>	text로부터 분리자들 delim으로 문자열 토큰화기 생성
<u>메소드</u>	
<code>nextToken(): String</code>	문자열을 보고 분리자들을 지우고, 분리자가 포함되지 않은 길이가 0초과인 가장 긴 문자열을 찾아 지우고 결과로 반환
<code>nextToken(String new_delimiters): String</code>	nextToken()과 같으나 분리자를 새로 지정
<code>hasMoreTokens(): boolean</code>	토큰이 남았는지 여부 반환
<code>countTokens(): int</code>	토큰이 몇 개 남았는지 반환

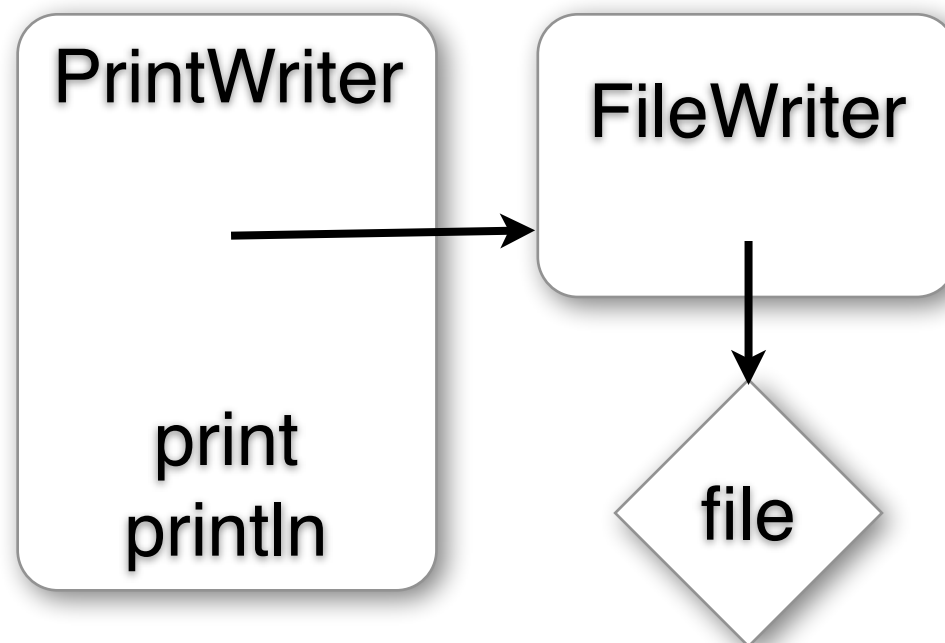
파일 (Files)

- 파일(file): 컴퓨터의 보조 기억장치에 저장된 기호의 나열
 - 문자 파일 (character file): 문자들의 나열
 - 바이너리 파일 (binary file): 0과 1의 나열
- 파일을 사용하려면
 - 열어서(open) 보거나 쓰고 다 사용했으면 닫아야(close)한다.
- 파일을 열 때 보기만 할 건지 쓰기만 할 건지 다 할 건지 지정해야 한다.
 - 입력 (input) 파일: 보기만 하는 파일
 - 출력 (output) 파일: 쓰기만 하는 파일

파일 출력 객체 만들기

- **FileWriter 객체**
 - 파일의 주소 보유
 - 파일에 쓰기 메소드 보유
 - 생성시 파일이 쓰기 모드로 열린다. 없으면 생성.
- **PrintWriter 객체**
 - **FileWriter 객체 보유**
 - **print, println을 포함해 여러 메소드 제공**

```
PrintWriter ofile =  
    new PrintWriter  
        (new FileWriter("file.txt"));
```



예제, 간단한 출력

```
import java.io.*; IO 라이브러리 사용
```

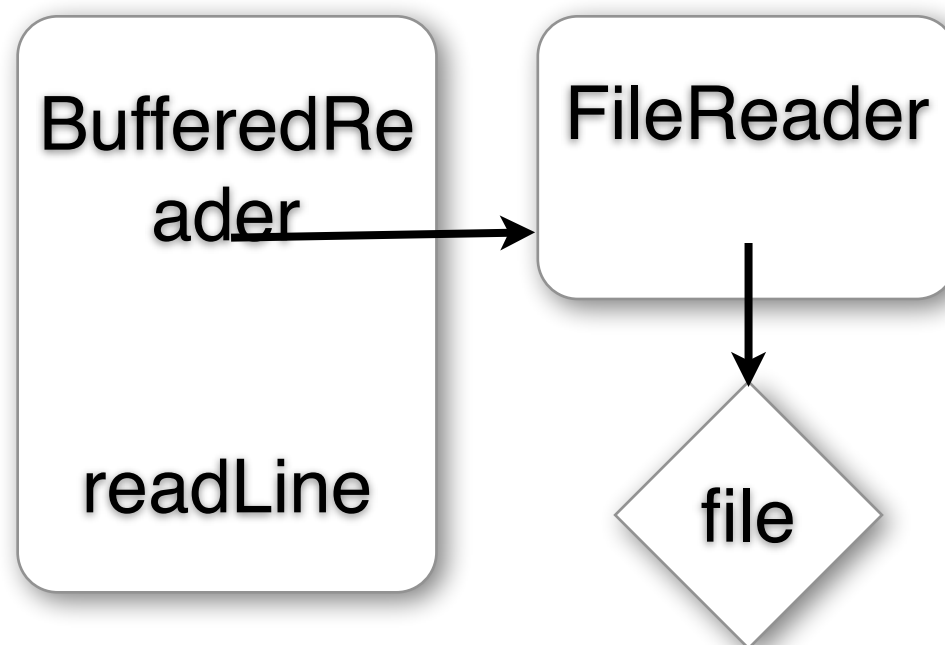
```
public class Output1 {
    public static void main(String[] args) throws IOException {
        PrintWriter outfile =
            new PrintWriter(new FileWriter("test.txt"));
        outfile.println("Hello to you!");
        outfile.print("How are");
        outfile.println(" you?");
        outfile.println(47+2);
        outfile.close();
    }
}
```

IOException이 발생할 수 있다.

파일 입력 객체 만들기

- FileReader 객체
 - 파일의 주소 보유
 - 파일 읽기 메소드 보유
 - 생성시 파일이 읽기 모드로 열린다. 없으면 예외 발생.
- BufferedReader 객체
 - FileReader 객체 보유
 - readLine을 포함해 여러 메소드 제공

```
BufferedReader ifile =  
    new BufferedReader  
        (new FileReader("file.txt"));
```

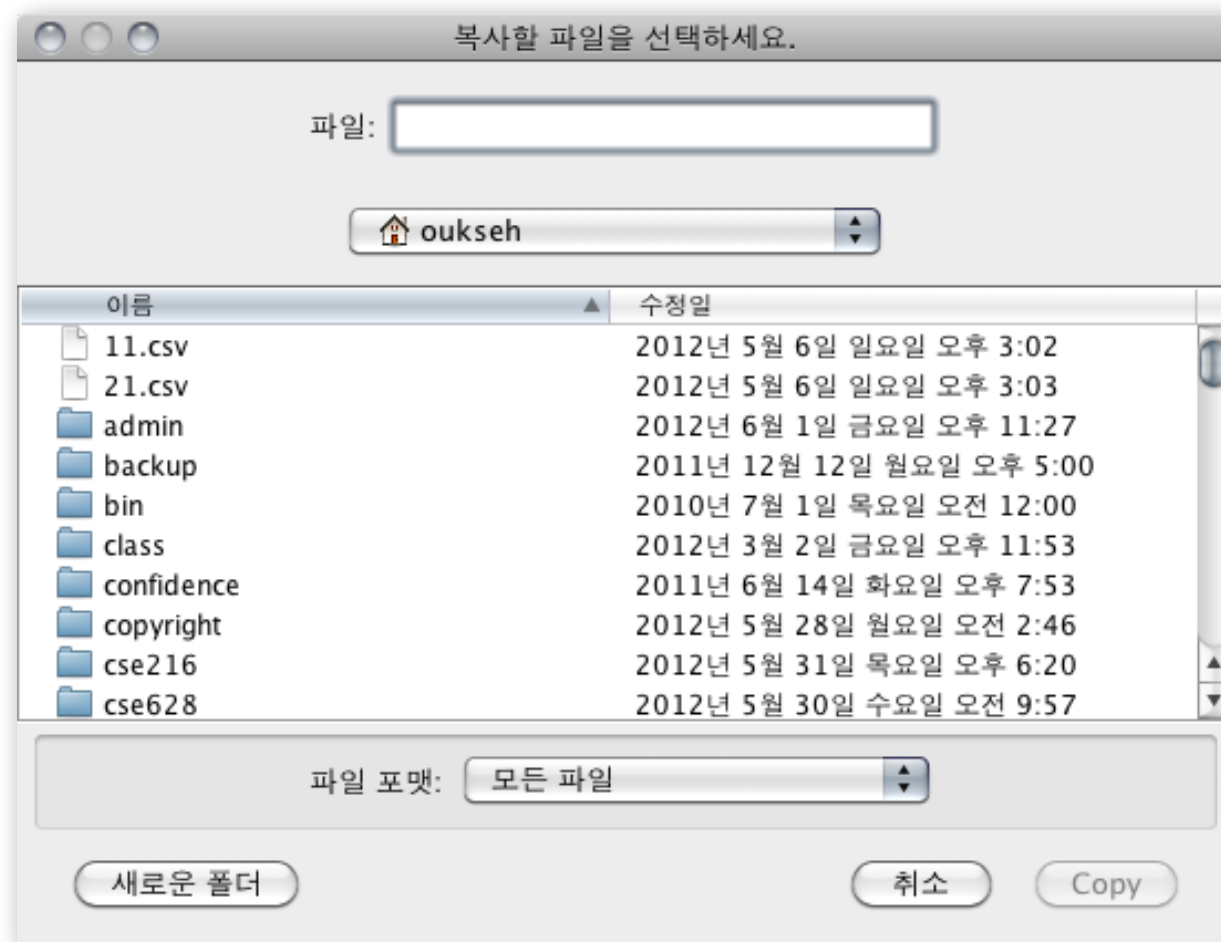


예제, 파일 복사

```
import java.io.*; import javax.swing.*;

public class CopyFile {
    public static void main(String[] args) throws IOException {
        String f = JOptionPane.showInputDialog("Input filename, please: ");
        BufferedReader infile = new BufferedReader(new FileReader(f));
        PrintWriter outfile = new PrintWriter(new FileWriter(f + ".out"));
        while (infile.ready()) {
            outfile.println(infile.readLine());
        }
        infile.close();
        outfile.close();
    }
}
```

파일 선택 GUI



예, 파일 선택해서 복사

```
import java.io.*; import javax.swing.*;

public class CopyFile {
    public static void main(String[] args) throws IOException {
        JFileChooser chooser = new JFileChooser();
        chooser.setDialogTitle("복사할 파일을 선택하세요.");
        int result = chooser.showDialog(null, "Copy");
        if(result != JFileChooser.APPROVE_OPTION)
            System.exit(0);
        String f = chooser.getSelectedFile().toString();
        BufferedReader infile = new BufferedReader(new FileReader(f));
        PrintWriter outfile = new PrintWriter(new FileWriter(f + ".out"));
        while (infile.ready()) {
            outfile.println(infile.readLine());
        }
        infile.close();
        outfile.close();
    }
}
```

특별한 녀석들: 표준 입출력

- System.out: 표준 출력
 - System.out.println
- System.in: 표준입력
 - 불행히도 System.in은 InputStream 클래스로 한 번에 한 문자만 입력받는 메소드 정도 밖에 없다.
 - System.in을 편하게 쓰려면 BufferedReader로 변환 필요

```
BufferedReader keyboard =  
    new BufferedReader(new InputStreamReader(System.in));  
String s = keyboard.readLine();
```

예제, 급여 처리

이순신 | 31 | 20250
이육세 | 42 | 24500
홍길동 | 18 | 18000
!

이순신 | 627750
이육세 | 1029000
홍길동 | 324000
!

- 입력 파일을 읽는 PayrollReader 클래스를 작성하라.
 - 이름, 일한시간, 시급
- 출력 파일을 쓰는 PayrollWriter 클래스를 작성하라.
 - 이름, 급여
- 입력 파일로부터 출력파일을 만들어라.

명세

class PayrollReader	
<u>Methods</u>	
getNextRecord(): boolean	파일에서 다음 레코드를 읽는다. 제대로 읽었으면 참을 아니면 거짓을 반환한다.
nameOf(): String	현재 레코드의 고용인 이름을 반환한다.
hoursOf(): int	현재 레코드의 고용인이 일한 시간을 반환한다.
payrateOf(): double	현재 레코드의 고용인의 시급을 반환한다.
close()	파일을 닫는다.

구현

```
import java.io.*; import java.util.*;

public class PayrollReader {
    private BufferedReader infile;
    private String EOF = "!";
    private String name;
    private int hours, payrate;

    public PayrollReader(String file_name) {
        infile = new BufferedReader(new FileReader(file_name));
    }
    public String nameOf() { return name; }
    public int hoursOf() { return hours; }
    public int payrateOf() { return payrate; }
    public void close() { infile.close(); }
```

구현

```
public boolean getNextRecord() {  
    if(!infile.ready()) return false;  
    String line = infile.readLine();  
    StringTokenizer t = new StringTokenizer(line, "|");  
    String s = t.nextToken().trim();  
    if(s.equals(EOF) || t.countTokens() != 2) return false;  
    name = s;  
    hours = new Integer(t.nextToken().trim()).intValue();  
    payrate = new Integer(t.nextToken().trim()).intValue();  
    return true;  
}  
}
```

제어기

```
import javax.swing.*;

public class Payroll {
    private static void processPayroll(String in, String out) {
        PayrollReader reader = new PayrollReader(in);
        PayrollWriter writer = new PayrollWriter(out);
        while(reader.getNextRecord()) {
            int pay = reader.hoursOf() * reader.payrateOf();
            writer.printCheck(reader.nameOf(), pay);
        }
        reader.close(); writer.close();
    }
    public static void main(String[] args) {
        String in_name =
        JOptionPane.showInputDialog("Plead type input payroll name: ");
        String out_name =
        JOptionPane.showInputDialog("Plead type output payroll name: ");
        if(in_name != null && out_name != null)
            processPayroll(in_name, out_name);
    }
}
```

안전한 코딩

- getNextRecord에서 문제가 발생하였다면?
 - 파일을 읽지 못한다면?
 - 파일이 열렸는데 시스템 문제로 인해 계속 읽을 수 없을 수도 있다.
 - 레코드가 원하는 형식이 아니면?
 - 급여 파일이 깨졌을 수도 있다.
- 예상되는 위기 대처법
 - 파일에 문제가 있다면 false를 반환
 - 레코드가 원하는 형식이 아니면 다음 레코드를 읽음
 - 어쨌든 사용자가 알 수 있도록 로그(log)는 남겨야 한다!

오류를 어떻게 알 수 있는가?

- `infile.readLine()` 에서 못 읽으면 `IOException` 예외 발생
 - 프로그램을 죽일 순 없다.
 - 예외를 잡아서 로그를 남기고 `false`를 반환해야 한다.
- `t.countToken() != 2` 이면 레코드 형식 오류
 - 로그를 남기고 다음 줄로 다시 시도

예외 잡기

- try 몸체에서 실행하다가 예외가 발생하면,
 - 그 아래 문장들은 실행되지 않고 밖으로 탈출
- catch 구문에서 예외를 잡는다.
 - 단 제한된 타입의 예외만
 - catch 구문을 실행하고 정상적인 수행을 계속

```
try {  
    명령문;  
    ...  
}  
catch (타입 변수) {  
    명령문;  
    ...  
}
```

전혀 안전하지 않은 코딩

```
public boolean getNextRecord() {  
    if(!infile.ready()) return false;  
    String line = infile.readLine();  
    StringTokenizer t = new StringTokenizer(line, "|");  
    String s = t.nextToken().trim();  
    if(s.equals_EOF) || t.countTokens() != 2) return false;  
    name = s;  
    hours = new Integer(t.nextToken().trim()).intValue();  
    payrate = new Integer(t.nextToken().trim()).intValue();  
    return true;  
}  
}
```


파일을 읽다가 오류난 경우를 잡자

```
public boolean getNextRecord() {
    boolean result = false;
    try {
        if(infile.ready()) return false;
        String line = infile.readLine();

        ...
        if(!s.equals EOF) && t.countTokens() == 2) {
            ...
            result = true;
        }
    }
    catch (IOException e) {
        System.out.println("PayrollReader error: " + e.getMessage());
    }
    return result;
}
```

형식이 이상한 경우를 잡자

```
public boolean getNextRecord() {
    boolean result = false;
    try {
        if(infile.ready()) return false;
        String line = infile.readLine();

        ...
        if(!s.equals(EOF)) {
            if(t.countTokens() == 2) {
                ...
                result = true;
            }
            else {
                System.out.println("PayrollReader: bad record format: "
                                   + line + " Skipping record");
                result = getNextRecord();
            }
        }
    }
    catch (IOException e) {
        System.out.println("PayrollReader error: " + e.getMessage());
    }
    return result;
}
```

오류 처리를 한 곳에 모으자

```
public boolean getNextRecord() {
    boolean result = false;
    try {
        ...
        if(!s.equals(EOF)) {
            if(t.countTokens() == 2) {
                ...
                result = true;
            }
            else throw new RuntimeException(line);
        }
    }
    catch (IOException e) {
        System.out.println("PayrollReader error: " + e.getMessage());
    }
    catch (RuntimeException e) {
        System.out.println("PayrollReader error: bad record format: "
            + e.getMessage() + " Skipping record");
        result = getNextRecord();
    }
    return result;
}
```

예외도 객체다

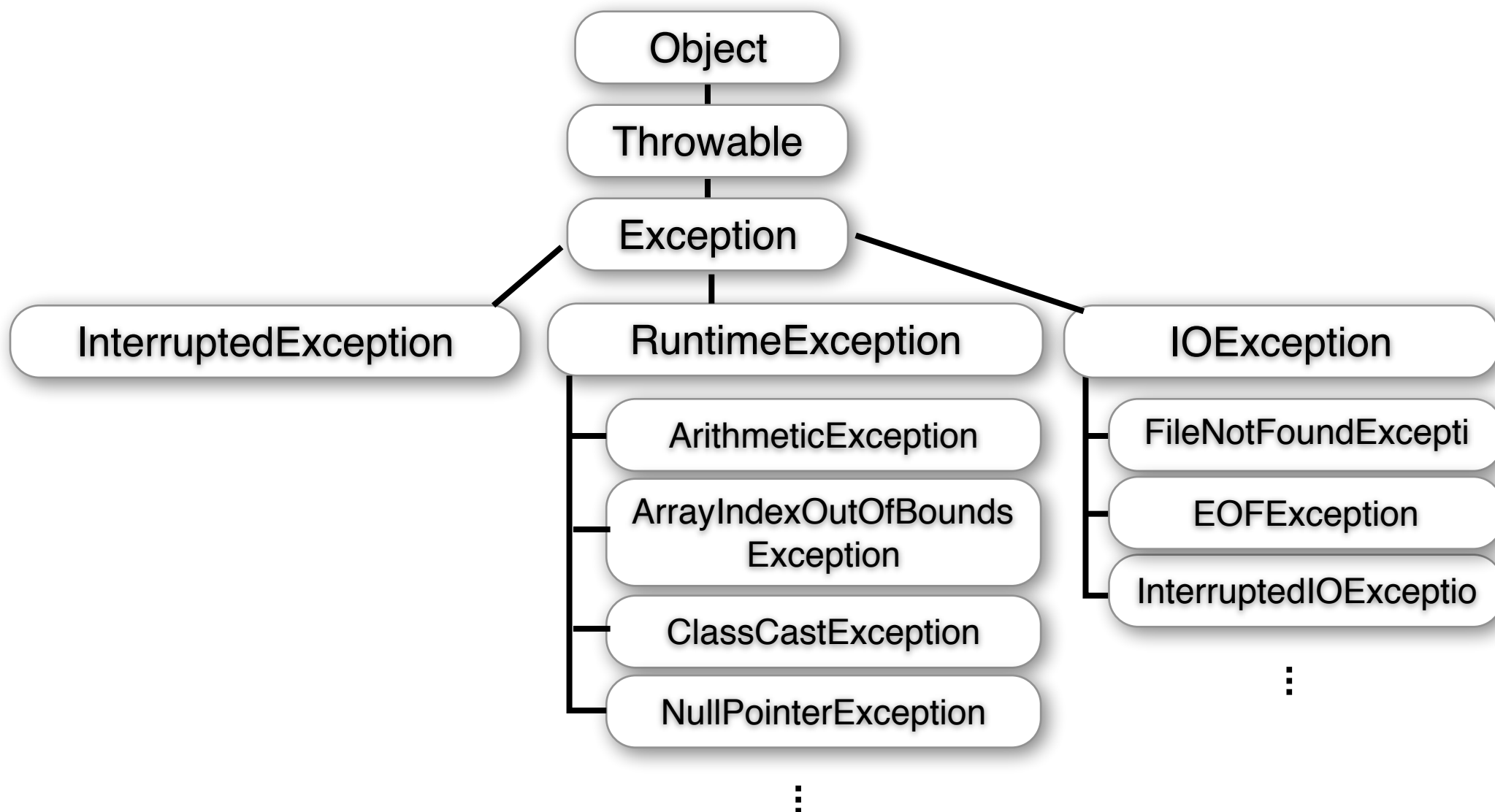
- 예외 객체는 오류에 대한 정보를 가진 객체다.
- 오류에 대한 설명
- 오류가 발생한 위치, 추가로 탈출 경로

```
Exception in thread "main" java.io.FileNotFoundException: /Volumes/Pamela/
Users/oukseh/11.csv (Permission denied)
    at java.io.FileInputStream.open(Native Method)
    at java.io.FileInputStream.<init>(FileInputStream.java:120)
    at java.io.FileInputStream.<init>(FileInputStream.java:79)
    at java.io.FileReader.<init>(FileReader.java:41)
    at CopyFile.main(CopyFile.java:11)
```

예외의 메소드

<code>getMessage(): String</code>	오류에 대한 정보
<code>toString(): String</code>	예외 객체를 문자열로 변환. 일반적으로 예외 클래스 이름과 오류에 대한 정보를 포함
<code>printStackTrace() ()</code>	예외가 언제 발생했고, 어디를 따라 탈출했는지 정보를 콘솔에 출력

예외도 클래스 구조도를 가지고 있다



catch에서 지정하는 예외

- `catch(<타입> e) { ... }`는
 - <타입>의 하위 타입만 잡는다.
 - `Exception`으로 하면 모든 예외
 - `RuntimeException`으로 하면 모든 실행시간 예외
 - `IOException`으로 하면 파일처리 예외

예, 잘할때까지 정수 입력 받기

```
private int readAnIntFrom(BufferedReader view) throws IOException {  
    int num;  
    try {  
        System.out.print("Type an int: ");  
        String s = view.readLine();  
        num = new Integer(s).intValue();  
    }  
    catch (Exception e) {  
        System.out.println("Error: " + e.getMessage()  
            + " not an integer; try again.");  
        num = readAnIntFrom(view); // restart return num;  
    }  
}
```

Exception 보다는 RuntimeException

RuntimeException 보다는 NumberFormatException

throws 절

- 예외도 프로그램을 죽인다.
 - 오류 발생보다는 좋지만 프로그램을 비정상 종료한다는 점에서 역시 위험한 녀석이다.
- Java의 해결책
 - RuntimeException을 제외한 모든 예외에 대해
 - 모든 메소드마다
 - 발생가능한 예외를 명시해야 한다.
 - 사용자가 인지할 수 있도록.

예외도 잘 써야 보약

```
public class ExceptionExample {
    public ExceptionExample() { }
    public void f() {
        try { g(); }
        catch (RuntimeException e) { System.out.println("caught at f"); }
        System.out.println("f completes");
    }
    public void g()
    {
        try {
            PrintWriter outfile = new PrintWriter(new FileWriter("text.out"));
            try { outfile.println( h() ); }
            catch (NullPointerException e)
            { System.out.println("null pointer caught at g"); }
        }
        catch (IOException e)
        { System.out.println("io error caught at g"); }
        System.out.println("g completes");
    }
    private int h() {
        int[] r = new int[2];
        return r[3]; // ArrayIndexOutOfBoundsException!
    }
}
```

f를 호출하면 무슨 일이 일어날까?

여러 예외를 구분짓고 싶다

- 계산A, 계산B를 하다가 각각 오류가 발생했을 때 처리 루틴을 별도로 작성하고 싶다.
 - `try { ... } catch (?) { ... }`
- 방법1: 예외 정보로 구분
 - `throw new RuntimeException("A");`
 - `throw new RuntimeException("B");`
 - `try { ... } catch (RuntimeException e) {
String m = e.getMessage();
if(m.equals("A")) ... if(m.equals("B")) ...`

여러 예외를 구분짓고 싶다

○ 방법2: 사용자 정의 예외 사용

- `class MyExceptionA extends Exception {}`
- `class MyExceptionB extends Exception {}`
- `throw new MyExceptionA();`
- `throw new MyExceptionB();`
- `try { ... }`
 `catch (MyExceptionA e) { ... }`
 `catch (MyExceptionB e) { ... }`

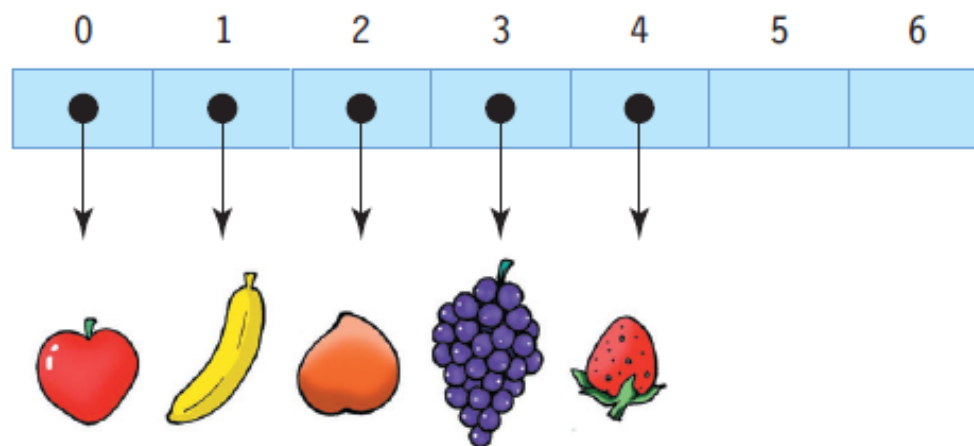
자바 객체 모음 (Collection)

객체 모음 (Collection)의 개념

- 요소(element) 객체들의 저장소
 - 요소의 갯수에 따라 크기 자동 조절
 - 요소의 삽입, 삭제에 따라 요소의 위치 자동 이동
 - 고정 크기의 배열을 다루는 어려움 해소
- 다양한 객체들의 삽입, 삭제, 검색 등 관리 용이

객체 모음 (Collection)의 개념

배열(array)



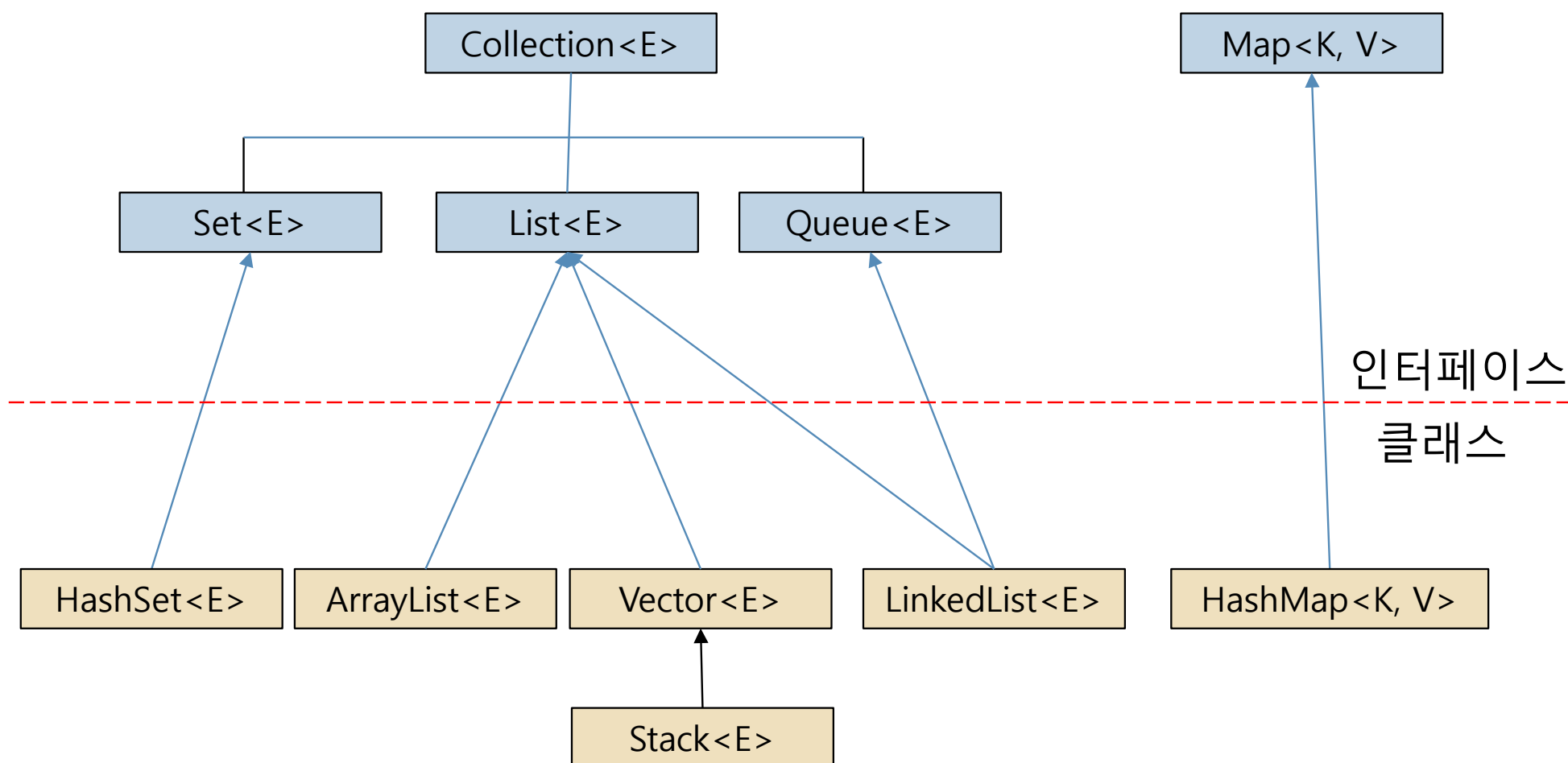
- 고정 크기 이상의 객체를 관리할 수 없다.
- 배열의 중간에 객체가 삭제되면 응용프로그램에서 자리를 옮겨야 한다.

컬렉션(collection)



- 가변 크기로서 객체의 개수를 염려할 필요 없다.
- 컬렉션 내의 한 객체가 삭제되면 컬렉션이 자동으로 자리를 옮겨준다.

컬렉션 인터페이스와 클래스



컬렉션의 특징

- 통칭 (Generic) 기법으로 구현
 - 특정 타입만 다루지 않고 여러 종류의 타입으로 변신할 수 있도록 클래스나 메소드를 일반화 시키는 방법
- 예: 벡터 Vector <E>
 - E 에 구체적인 타입을 주어 그 타입만 다루는 벡터로 활용
 - Vector <Integer>, Vector<String>
- 컬렉션의 요소는 객체만 가능
 - int, double 등 기본타입은 구체화 불가. 대신 Wrapper 클래스들 사용 (Integer, Double)

벡터 <E>의 특성

- 배열의 길이 제한 극복
 - 요소의 개수가 넘치면 자동으로 길이 조절
- 요소 객체들을 삽입, 삭제, 검색하는 컨테이너
 - 삽입, 삭제에 따라 자동으로 요소의 위치 조정
- 벡터의 맨 뒤, 중간에 객체 삽입 가능
- 임의의 위치에 있는 객체 삭제 가능

```

import java.util.Vector;
public class VectorEx {
    public static void main(String[] args) {
        int sum = 0;
        // 정수 값만 다루는 벡터 생성
        Vector<Integer> v = new Vector<Integer>();
        v.add(5); v.add(4); v.add(-1); // 5, 4, -1 삽입
        v.add(2, 100); // 4와 -1 사이에 정수 100 삽입
        System.out.println("벡터 내의 요소 객체 수 : " + v.size());
        System.out.println("벡터의 현재 용량 : " + v.capacity());
        // 모든 요소 정수 출력하기
        for(int i=0; i<v.size(); i++) {
            int n = v.get(i); // 벡터의 i 번째 정수
            System.out.println(n);
        }
        // 벡터 속의 모든 정수 더하기 int sum = 0;
        for(int i=0; i<v.size(); i++) {
            int n = v.elementAt(i); // 벡터의 i 번째 정수
            sum += n;
        }
        System.out.println("벡터에 있는 정수 합 : " + sum);
    }
}

```

```

import java.util.Vector;
class Point {
    private int x, y;
    public Point(int x,int y) {this.x = x; this.y = y;}
    public String toString() {
        return "(" + x + "," + y + ")";
    }
}

public class PointerVectorEx {
    public static void main(String[] args) {
        Vector<Point> v = new Vector<Point>();
        // 3 개의 Point 객체 삽입
        v.add(new Point(2, 3));
        v.add(new Point(-5, 20));
        v.add(new Point(30, -8));
        v.remove(1); // 인덱스 1의 Point(-5, 20) 객체 삭제
        // 벡터에 있는 Point 객체 모두 검색하여 출력
        for(int i=0; i<v.size(); i++) {
            Point p = v.get(i); // 벡터의 i 번째 Point 객체 얻어내기
            System.out.println(p); // p.toString()을 이용하여 객체 p 출력 }
        }
    }
}

```

HashMap <K, V>

- 키(key)와 값(value)의 쌍으로 구성되는 요소를 다루는 컬렉션
 - K: 키로 사용할 요소의 타입
 - V: 값으로 사용할 요소의 타입
 - 키와 값이 한쌍으로 삽입
 - 값을 검색하기 위해 반드시 키 이용
- 삽입 및 검색 빠름
 - put(), get() : 요소 삽입 및 검색

```

import java.util.*;
public class HashMapDicEx {
    public static void main(String[] args) {
        // 영어 단어와 한글 단어의 쌍을 저장하는 HashMap 컬렉션 생성
        HashMap<String, String> dic = new HashMap<String, String>();
        // 3 개의 (key, value) 쌍을 dic에 저장
        dic.put("baby", "아기"); // "baby"는 key, "아기"는 value
        dic.put("love", "사랑");
        dic.put("apple", "사과");
        // dic 해시맵에 들어 있는 모든 (key, value) 쌍 출력
        // 모든 키를 Set 컬렉션에 받아옴
        Set<String> keys = dic.keySet();
        // Set에 접근하는 Iterator 리턴.
        Iterator<String> it = keys.iterator();
        while(it.hasNext()) {
            String key = it.next(); // 키
            String value = dic.get(key); // 값
            System.out.print("(" + key + "," + value + ")");
        }
        System.out.println();
    }
}

```

```
// 영어 단어를 입력받고 한글 단어 검색
Scanner scanner = new Scanner(System.in);
for(int i=0; i<3; i++) {
    System.out.print("찾고 싶은 단어는?");
    String eng = scanner.next();
    //해시맵에서 '키' eng의 '값' kor 검색
    String kor = dic.get(eng);
    if(kor == null)
        System.out.println(eng + "는 없는 단어 입니다.");
    else System.out.println(kor);
}
}
}
```

탐색 및 정렬

순차 탐색 (Sequential Search)

- 리스트를 왼쪽부터 오른쪽으로 순차적으로 탐색하며 주어진 키와 동일한 값의 인덱스 반환.

```
public static int search(String key, String[] a)
{
    for (int i = 0; i < a.length; i++)
        if ( a[i].equals(key) == 0 ) return i;
    return -1;
}
```

효율적?









- 1천만명의 고객을 가진 카드회사가 1초당 1000 건의 거래를 처리하고 싶다. 이를 순차 탐색을 사용하는 랩탑 컴퓨터 한대에서 처리할 수 있을까?
- 랩탑 컴퓨터는 편의상 1천만명의 고객 정보를 담을 수 있는 충분한 크기의 하드 용량을 가졌다고 가정
- 초당 2천만번의 비교 연산 가능하다고 가정

효율적?

- 1천만명의 고객을 가진 카드회사가 1초당 1000 건의 거래를 처리하고 싶다. 이를 순차 탐색을 사용하는 랩탑 컴퓨터 한대에서 처리할 수 있을까? **불가능!**
- 랩탑 컴퓨터는 편의상 1천만명의 고객 정보를 담을 수 있는 충분한 크기의 하드 용량을 가졌다고 가정
- 초당 2천만번의 비교 연산 가능하다고 가정
1000건의 거래 처리를 위해 500초 소요.
 $(10^7 \times 10^3 / (2 \times 10^7) = 500)$
즉, 500대가 필요.

이진 탐색 (Binary Search)

- 먼저 리스트를 정렬
- 주어진 키의 위치를 알아 내기 위해 스무고개 놀이
 - 가운데 위치한 키 비교
 - 만약 같으면, 종료
 - 아니면, 왼쪽 절반 혹은 오른쪽 절반에서 탐색

<i>interval</i>	<i>size</i>	<i>Q</i>	<i>A</i>
	128	< 64?	false
	64	< 96?	true
	32	< 80?	true
	16	< 72?	false
	8	< 76?	false
	4	< 78?	true
	2	< 77?	false
	1	= 77	

```

public class BinarySearch {
    public static int search(String key, String[] a) {
        return search(key, a, 0, a.length);
    }
    // 불변식: a[lo] <= key <= a[hi-1].
    public static int search(String key, String[] a, int lo, int hi)
    {
        if (hi <= lo) return -1;
        int mid = lo + (hi - lo) / 2;
        int cmp = a[mid].compareTo(key);
        if (cmp > 0) return search(key, a, lo, mid);
        else if (cmp < 0) return search(key, a, mid+1, hi);
        else return mid;
    }
    public static void main(String[] args) {
        String[] arr = new String[4];
        arr[0] = "a"; arr[1] = "b"; arr[2] = "c"; arr[3] = "d";
        System.out.println(search("c", arr));
    }
}

```

a.compareTo(b): a가 b보다 “작으면” -1, 같으면 0, “크면” -1 반환

몇번의 연산 필요?

- 크기 N 배열에 이진 탐색: 1번 비교
후 크기 $N/2$ 의 배열에 이진 탐색

$$N \rightarrow N/2 \rightarrow N/4 \rightarrow \dots \rightarrow 1$$

$$\begin{aligned}
 &1 \\
 &2 \rightarrow 1 \\
 &4 \rightarrow 2 \rightarrow 1 \\
 &8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
 &16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
 &32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
 &64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
 &128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
 &256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
 &512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \\
 &1024 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1
 \end{aligned}$$

- 즉, 비교 횟수는 어떤 숫자를 1로 만들때까지 2로 나누는 횟수와 동일:
 $\log_2 N$

효율적?

- 1천만명의 고객을 가진 카드회사가 1초당 1000 건의 거래를 처리하고 싶다. 이를 이진 탐색을 사용하는 랩탑 컴퓨터 한대에서 처리할 수 있을까? **충분!**
- 랩탑 컴퓨터는 편의상 1천만명의 고객 정보를 담을 수 있는 충분한 크기의 하드 용량을 가졌다고 가정
- 초당 2천만번의 비교 연산 가능하다고 가정
1000건의 거래 처리를 위해 0.001초 소요.
 $(\log_2(10^7) \times 10^3) / (2 \times 10^7) = 0.001$

정렬 (Sorting)

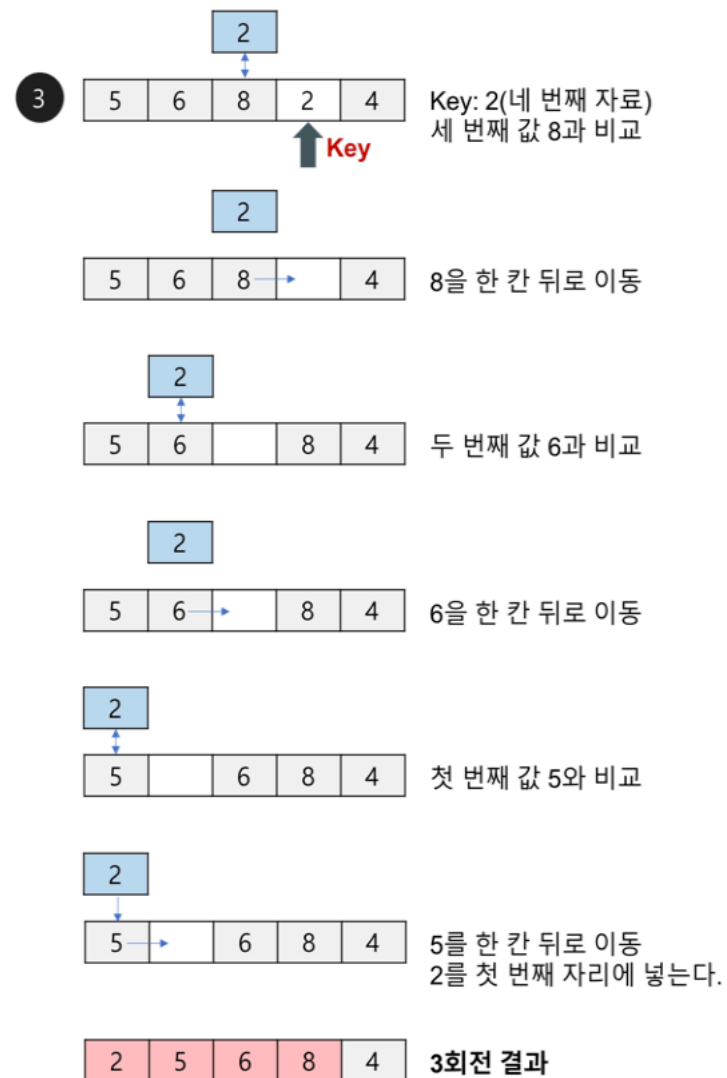
- 리스트에 있는 N 개의 요소를 오름차순 혹은 내림차순으로 재배치
 - 사용 예: 이진 탐색, 통계, 데이터베이스, 바이오 인포매틱스, 컴퓨터 그래픽스, 복잡한 수리과학 계산 등...
- 다양한 알고리즘
 - 삽입 정렬, 버블 정렬, 선택 정렬, 병합 정렬, ...

삽입 정렬 (Insertion Sort)

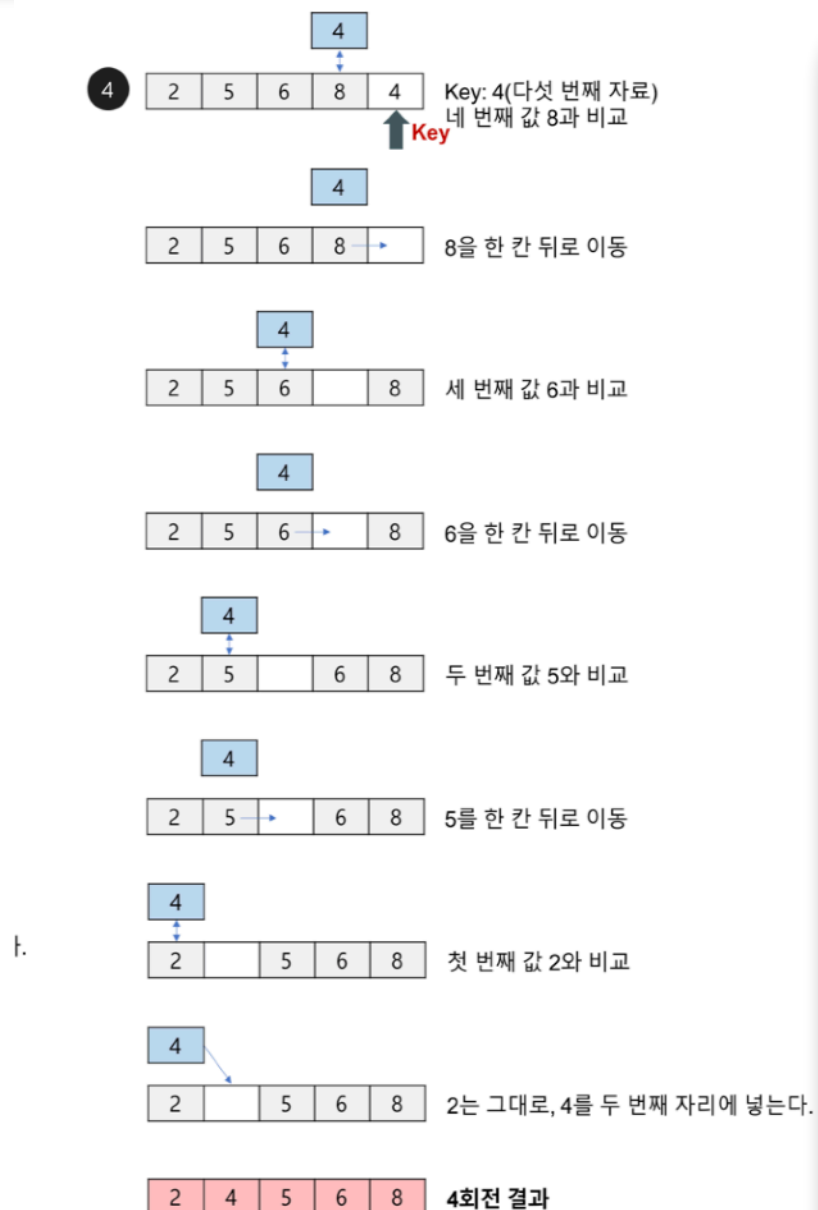
- 손 안의 카드를 정렬하는 방법과 유사
 - 새로운 카드를 기존의 정렬된 카드 사이의 올바른 자리를 찾아 삽입
 - 새로 삽입될 카드의 수 만큼 반복하면 전체 카드가 정렬됨
- 리스트의 모든 요소를 앞에서 부터 차례대로 이미 정렬된 배열 부분과 비교하여 자신의 위치를 찾아 삽입

초기상태

8	5	6	2	4
---	---	---	---	---



출처: <https://gmlwjd9405.github.io/2018/05/06/algorithm-insertion-sort.html>



오름차순
완성상태

2 4 5 6 8

```
public class Insertion {
    public static void sort(String[] a) {
        int N = a.length;
        for (int i = 1; i < N; i++)
            for (int j = i; j > 0; j--)
                if (a[j-1].compareTo(a[j]) > 0)
                    exch(a, j-1, j);
                else break;
    }
    private static void exch(String[] a, int i, int j) {
        String swap = a[i];
        a[i] = a[j];
        a[j] = swap;
    }
    public static void main(String[] args) {
        String[] arr = new String[4];
        arr[0] = "d"; arr[1] = "a";
        arr[2] = "c"; arr[3] = "b";
        sort(arr);
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

몇번의 연산 필요?

- 입력 리스트의 성질에 따라 다름 (길이: N)
 - 최선의 경우: 이미 오름차순으로 정렬되어있음
요소 이동없이 $N-1$ 번의 비교로 끝
 - 최악의 경우: 내림차순으로 정렬
외부 루프 매 반복마다 i 번의 비교 수행
 $(n-1) + (n-2) + \dots + 2 + 1 = n(n-1)/2$

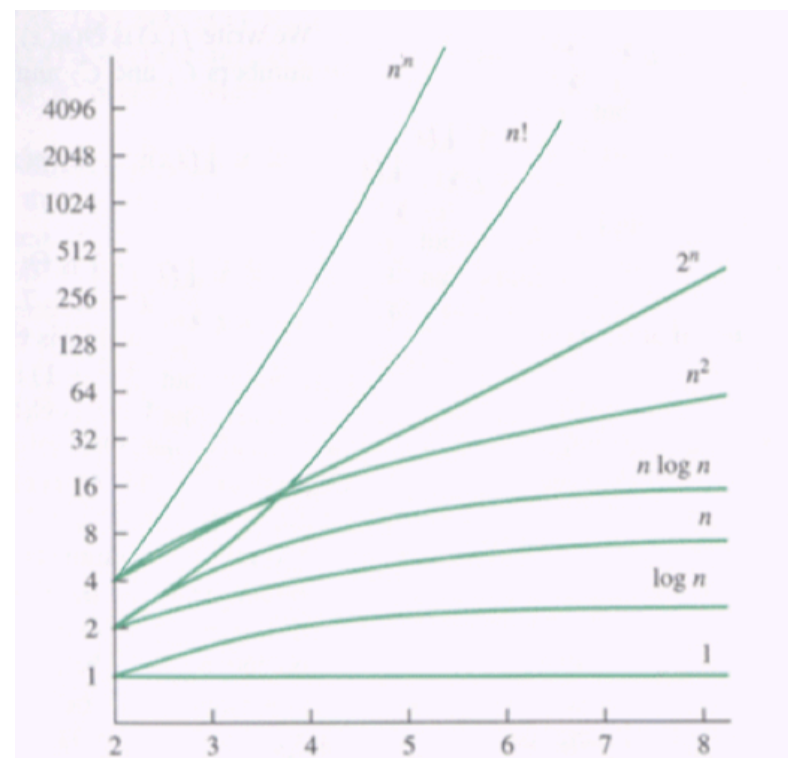
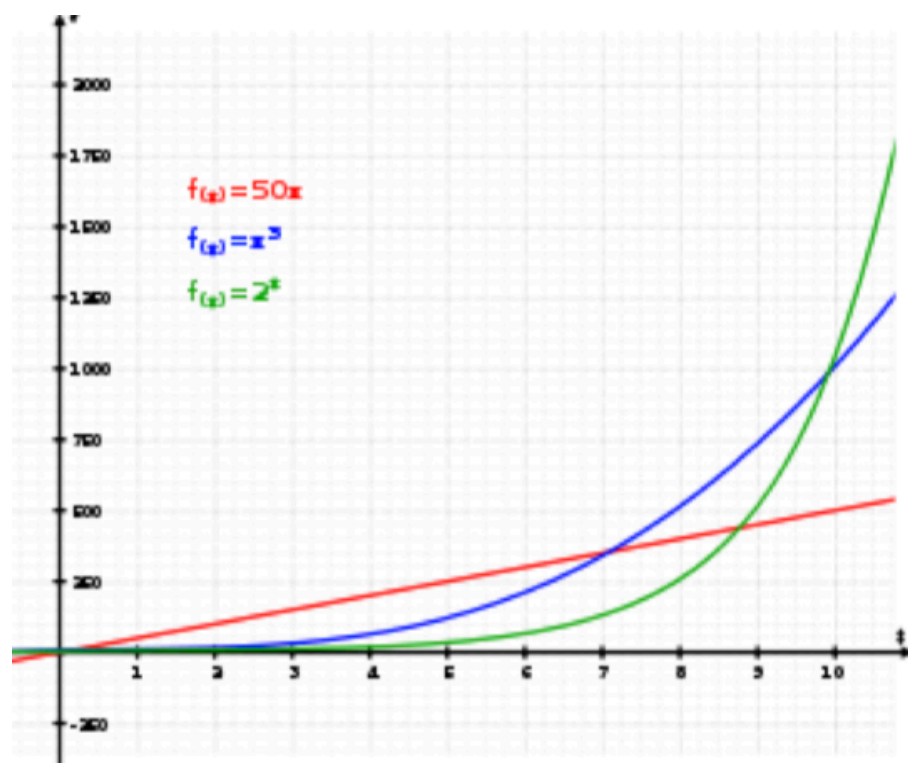
계산 복잡도 (Complexity)

- 프로그램 실행 비용의 증가 정도 (order of growth)
 - 계산비용: 시간과 메모리
 - 증가정도: 입력 크기에 대한 함수로. 단
관심: 입력이 커지면 결국 어떻게 될지 (asymptotic complexity)
- 계산복잡도가 $\Theta(f(n))$ 이다 (n : 입력 크기). 만일 그 복잡도가 $f(n)$ 으로 샌드위치 될 때:

$$k_1 \times f(n) \leq \bullet \leq k_2 \times f(n) \quad \text{단, } k_1, k_2 \text{는 } n \text{과 무관한 양수}$$

- n^2 , $1000 \times n^2$, $3 \times n^2 + 1000 \times n$ 모두 $\Theta(n^2)$

계산 복잡도 (Complexity)



- 삽입 정렬의 최악 계산 복잡도는 $\Theta(n^2)$

계산 복잡도 (Complexity)

- 삽입 정렬의 최선 계산 복잡도는 $\Theta(n)$
- 삽입 정렬의 최악 계산 복잡도는 $\Theta(n^2)$
- 순차 탐색의 최선 계산 복잡도는 $\Theta(1)$
- 순차 탐색의 최악 계산 복잡도는 $\Theta(n)$
- 이진 탐색의 최선 계산 복잡도는 $\Theta(1)$
- 이진 탐색의 최악 계산 복잡도는 $\Theta(\log_2 n)$

삽입정렬의 교훈

- 나쁜 알고리즘은 슈퍼컴퓨터도 구제 못함

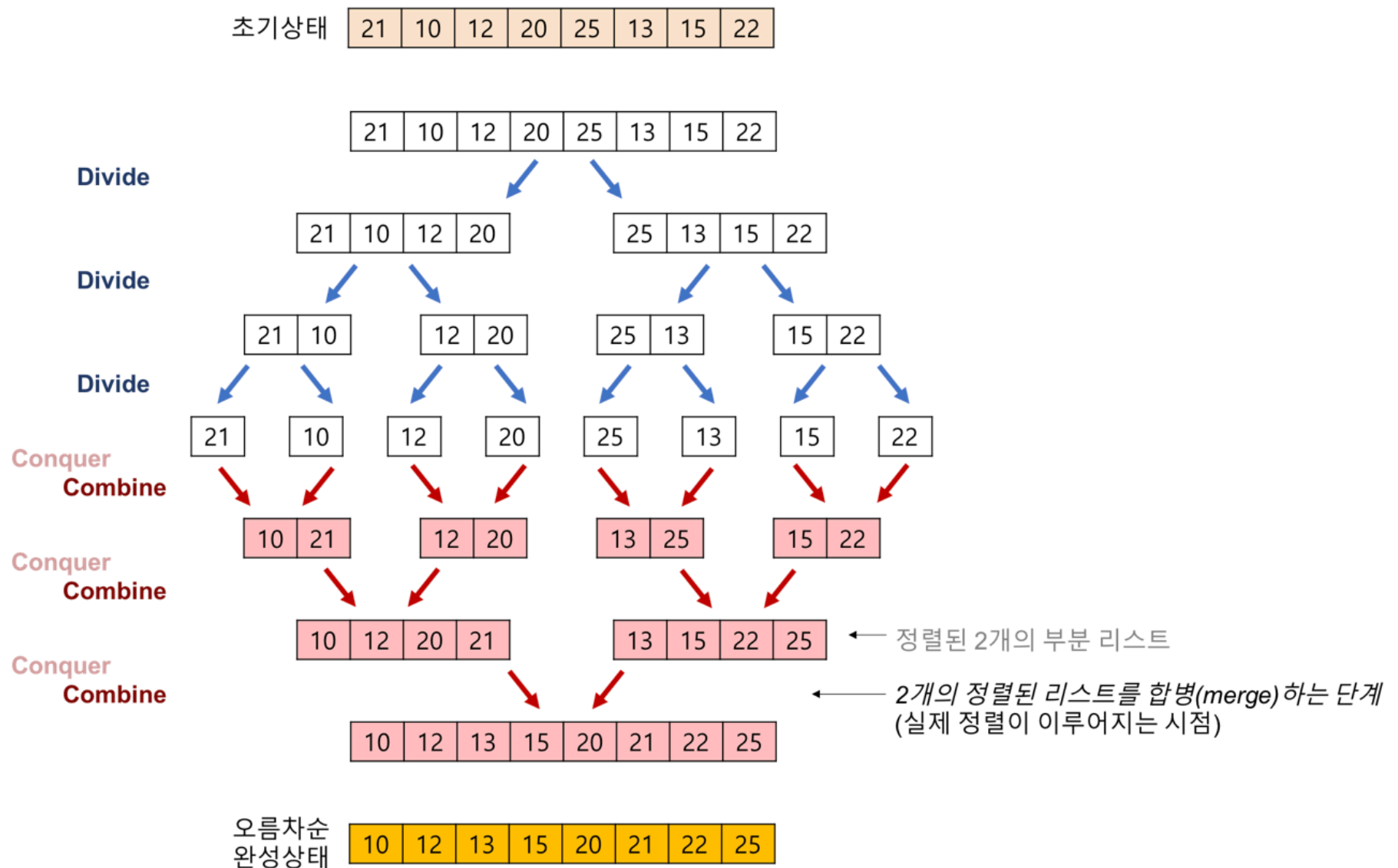
컴퓨터	초당 비교 연산 횟수	몇천	몇백만	몇십억
랩탑	10^7	순식간	1일	3세기
슈퍼컴퓨터	10^{12}	순식간	1초	2주

출처: <http://www.cs.princeton.edu/courses/archive/fall09/cos126/>

병합 정렬 (Merge Sort)

- 리스트를 둘로 나눔
- 각 절반을 재귀적으로 병합정렬을 이용하여 정렬
- 두 절반을 합쳐서 정렬된 전체를 구성

병합 정렬의 예

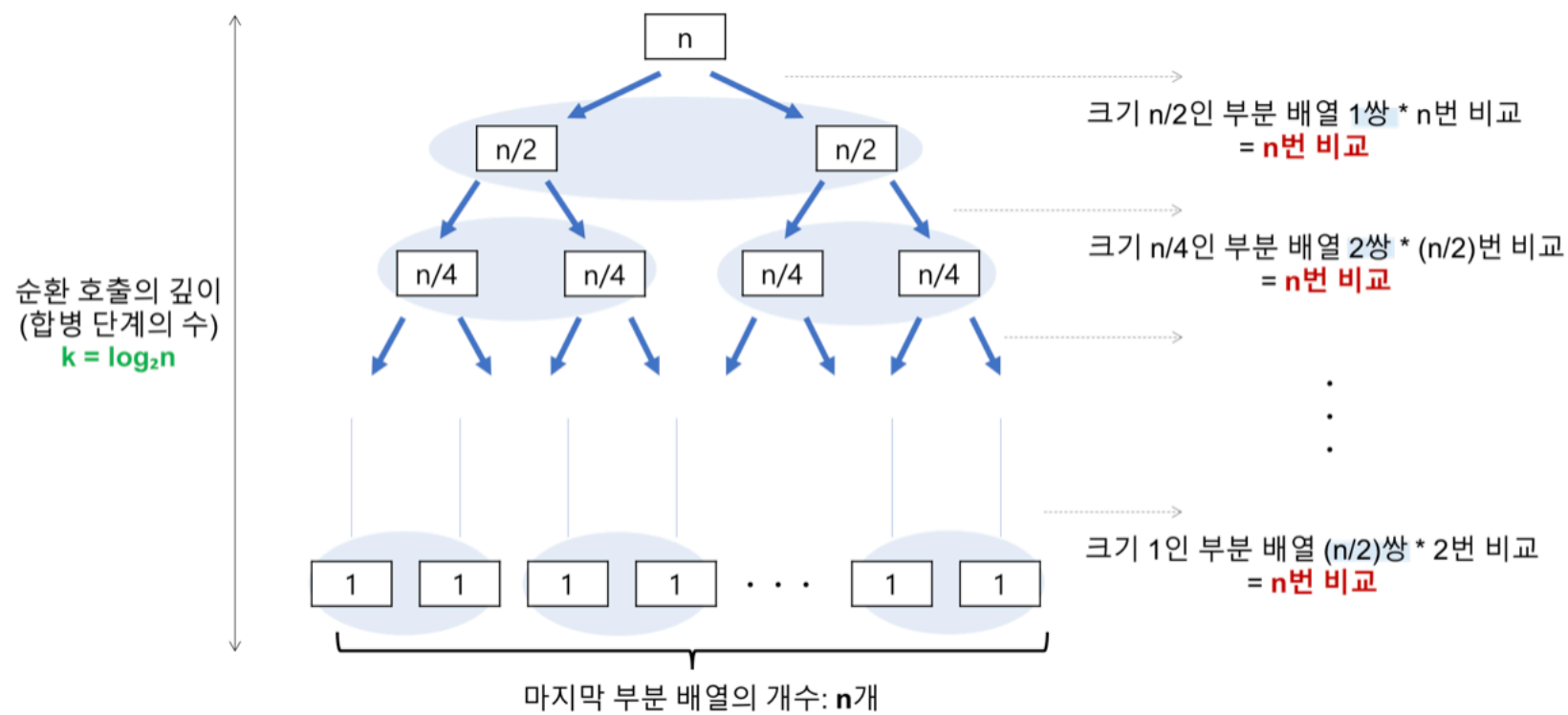


출처: <https://gmlwjd9405.github.io/2018/05/06/algorithm-merge-sort.html>

```
public class Merge {
    public static void sort(String[] a) {sort(a, 0, a.length);}
    // Sort a[lo, hi).
    public static void sort(String[] a, int lo, int hi) {
        int N = hi - lo;
        if (N <= 1) return;
        // Recursively sort left and right halves.
        int mid = lo + N/2;
        sort(a, lo, mid);
        sort(a, mid, hi);
        // Merge sorted halves
        String[] aux = new String[N];
        // Merge into auxiliary array.
        int i = lo, j = mid;
        for (int k = 0; k < N; k++) {
            if (i == mid) aux[k] = a[j++];
            else if (j == hi) aux[k] = a[i++];
            else if (a[j].compareTo(a[i]) < 0) aux[k] = a[j++];
            else aux[k] = a[i++];
        }
        // Copy back.
        for (int k = 0; k < N; k++) a[lo + k] = aux[k];
    }
}
```

계산 복잡도

○ 최선, 최악 모두 $\Theta(n \log_2 n)$



병합 정렬의 교훈

- 좋은 알고리즘은 슈퍼컴퓨터보다 뛰어남
(10억개 요소 정렬시)

컴퓨터	초당 비교연산 횟수	삽입정렬	병합정렬
랩탑	10^7	3세기	3시간
슈퍼컴퓨터	10^{12}	2주	순식간

출처: <http://www.cs.princeton.edu/courses/archive/fall09/cos126/>

요약

- 데이터를 저장 및 관리하기 위한 다양한 자료구조
 - 문서, 파일, 통칭(Generic)을 이용한 값 모음(Collection)
 - 예외 처리
- 자료구조에 자주 쓰이는 연산: 탐색과 정렬
 - 효율적인 탐색: 이진 탐색
 - 간단하지만 비효율적인 정렬: 삽입 정렬
 - 좀 더 복잡하지만 효율적인 정렬: 병합 정렬
- 효율적이란? 프로그램 계산 복잡도