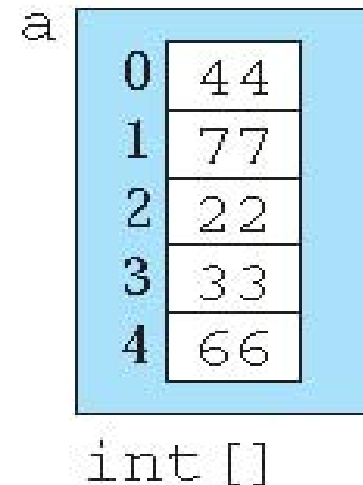


### 3. 배열

# 3.1 Java의 배열

- 배열(array)은 첨자 연산자를 이용해 접근할 수 있는 인접한 원소들의 시퀀스
- Java에서 배열은 객체임
- 배열의 타입은 `t[]` 형태
  - `t`는 배열의 원소 타입
  - 예: 원소 타입이 `int`이면 배열 타입은 `int[]`
- 유효한 배열 선언문
  - `int[] a;` // 원소는 프리미티브 타입 `int`임
  - `String[] args;` // 원소는 객체 타입 `String`임
  - `List[] lists;` // 원소는 인터페이스 타입 `List`임
  - `double[][] matrix;` // 원소는 배열 타입 `double[]`임
  - `int[][][] x3d;` // 원소는 배열 타입 `int[][]`임

- new 연산자를 이용해 메모리 할당 가능
  - `a = new int[8];` // 8개의 int 원소로 된 배열
- 배열의 길이는 length 필드를 이용해서 접근
  - `int n = a.length;` // 배열 a의 원소의 수
- 배열의 길이가 n일 때 인덱스의 범위는 0에서 n-1이 됨
- 초기화 리스트를 이용해 배열을 초기화할 수 있음
  - `Int [ ] a = {44, 77, 22, 33, 66};`
- 한 배열을 다른 배열에 할당해도 실제로 복사되는 것은 아니다. 단지 다른 이름(즉, 다른 참조)만 부여하게 된다.
  - `b = a;` // a[]와 b[]는 동일 배열이 됨



- 배열과 관련된 기초적인 사항들
  - 다른 곳에서도 마찬가지로 방식으로 메소드의 매개변수 리스트에 배열 매개변수를 선언할 수 있다.
    - `public void print(int[] a)`
  - 배열은 이름만 이용해서 메소드로 전달된다.
    - `print(a);`
  - 배열 타입은 메소드에 대한 리턴 타입이 될 수 있다.
    - `public int[] randomArray(int n)`
  - 배열을 복사하려면 System 클래스에 정의된 `arraycopy()` 메소드를 이용할 수 있다.
    - `System.arraycopy(a, m, b, mm, k);`  
a: 소스 배열, b: 목표 배열, m: a[ ]에서의 시작 인덱스  
mm: b[ ]에서의 시작 인덱스, k: 복사할 원소의 수

- 중복 배열을 생성하려면 Object 클래스에 정의된 clone( ) 메소드를 이용할 수 있다.
  - `b = (int[])a.clone();`
  - clone()에 대한 리턴 타입은 Object이므로 타입을 배열로 변환시켜야 한다.
- 배열은 대개 for 루프를 이용해서 처리된다.

```
for (int i = 0; i < a.length; i++)  
    a[i] = random.nextInt(1000);
```
- 배열이 final로 선언되면 그 참조는 재할당될 수 없다.
  - `final int[] a = {22, 44, 66, 88};`
  - `a[3] = 99; // OK`
  - `a = new int[8]; // 불법임`

리스트 3.1

```
public class Main {
    private static java.util.Random random = new java.util.Random();
    public static void main(String[] args) {
        int[] a = randomInts(5,1000);
        int[] aa = (int[])a.clone(); // creates a duplicate of a in aa
        print(a);    print(aa);
        a[0] = a[1] = a[2] = 888;
        print(a);    print(aa);
    }
    public static int[] randomInts(int n, int range) {
        int[] a = new int[n];
        for (int i = 0; i < n; i++)
            a[i] = random.nextInt(range);
        return a;
    }
    public static void print(int[] a) {
        System.out.print "{" + a[0];
        for (int i = 1; i < a.length; i++)
            System.out.print "," + a[i];
        System.out.println("}");
    }
}
```

## 3.2 Java에서 배열의 프린팅

- 배열의 이름은 실제로는 배열에 대한 참조 변수의 이름임
  - 이 변수는 메모리에서 배열의 시작 주소를 저장함
  - 이 변수를 프린트하면 메모리 주소를 16진수로 보여주게 됨
  - 출력 문자열: [I@73d6a5
    - [I : 타입 int[]의 배열임을 의미
    - @73d6a5 : 배열이 저장된 메모리의 주소

```
1 public class Print {  
2     public static void main(String[] args) {  
3         int[] a = {66, 33, 99, 88, 44, 55, 22};  
4         System.out.println(a);  
5     }  
6 }
```

## 3.3 간단한 배열 알고리즘

- 최대값 원소의 발견
  - 주어진 시퀀스(배열)에서 최대 원소를 찾는 일

**Input:** a sequence  $\{a_0, a_1, a_2, \dots, a_{n-1}\}$ .

**Output:** an index value  $m$ .

**Postcondition:**  $a_m \geq a_i$ , for all  $i$ .

1. Let  $m = 0$ .
2. Repeat step 3 for  $i = 1$  to  $n - 1$ .
3. If  $a_i > a_m$ , set  $m = i$ .
4. Return  $m$ .



- 자리교환(스왑)
  - 두 원소의 위치를 서로 바꾸는 것

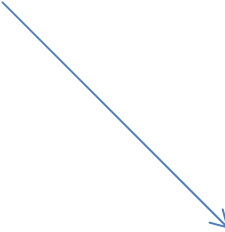
```
void swap(int[] a, int i, int j) {  
    int ai=a[i], aj=a[j];  
    if (ai == aj) return;  
    a[i] = aj;  
    a[j] = ai;  
}
```

## 3.4 객체의 배열

- 배열의 원소
  - 프리미티브 타입
    - 모든 원소는 동일한 타입
  - 참조 타입
    - 해당 배열에 대해 선언된 원소 타입의 확장에 속하기만 하면 다른 타입이 될 수도 있음
      - > 이질 배열(heterogeneous array)이 가능
    - 이질 배열의 예: 리스팅 3.4

- LISTING 3.4: A Heterogeneous Array of Objects

```
1 public class ObjectArray {  
2     public static void main(String[] args) {  
3         String s="Mercury";  
4         Float x = new Float(3.14159);  
5         java.util.Date d = new java.util.Date();  
6         int[] a = new int[] {11, 33, 55, 77, 99};  
7         Object[] objects = {s, x, d, a};  
8         print(objects);  
9     }  
10 }
```



다음 문장의 실행 결과는?  
for (int i=0; i<objects.length; i++)  
 System.out.print(", " + objects[i]);

## 3.5 순차 탐색

- 순차 탐색 (선형 탐색 또는 직렬 탐색)
  - 주어진 목표 값을 찾아 리스트를 앞에서부터 순차적으로 탐색
  - 목표가 발견된 첫 번째 위치를 리턴; 목표가 발견되지 않으면 음수를 리턴

**Input:** a sequence  $\{a_0, a_1, a_2, \dots, a_{n-1}\}$  and a target  $x$ .

**Output:** an index value  $i$ .

**Postcondition:** either  $a_i = x$ , or  $i < 0$ .

1. Repeat step 2 for  $i = 0$  to  $n - 1$ .
2. If  $a_i = x$ , return  $i$ .
3. Return  $-n$ .

복잡도는  $\Theta(n)$

선형시간(linear time)에 수행

## 3.6 복잡도 분석

- 점근적 비례 (asymptotically proportional)
  - 두 함수는 그들의 비율과 그 비율의 역에 대해 어떤 상수 값으로 상한값이 주어지면 점근적으로 비례
  - 예: 두 함수  $f(n) = 4n-3$ 과  $g(n)=n$ 은 점근적으로 비례
    - 두 함수의 비율인  $f(n)/g(n) = (4n-3)/(n) = 4-3/n$ 은 4에 의해 상한값이 주어짐
    - 그 역은  $g(n)/f(n) = (n)/(4n-3) = 1/(4-3/n)$ 은 1에 의해 상한값이 주어짐
    - $n$ 이 커지면  $3/n$ 은 아주 작아지므로  $f(n)/g(n)$ 은 거의 4와 같아지고, 이는  $f(n) \approx g(n)$ 임을 의미
  - $g(n)=n$ 에 점근적으로 비례적인 모든 함수의 집합을  $\Theta(n)$ 으로 표기
    - $f(n)=4n-3$ 은 복잡도 클래스  $\Theta(n)$ 의 한 원소가 됨
    - 이를  $4n-3 \in \Theta(n)$  또는  $4n-3 = \Theta(n)$ 으로 표기.

- 대부분의 알고리즘들의 실행 시간 행위는 아래 7개 주요 범주의 하나에 속함  
 $\Theta(1)$ ,  $\Theta(\lg n)$ ,  $\Theta(n)$ ,  $\Theta(n \lg n)$ ,  $\Theta(n^2)$ ,  $\Theta(n^3)$ ,  $\Theta(2^n)$ 
  - 이 범주들을 복잡도 클래스(complexity class) 또는 점근적 성장 클래스(asymptotic growth class)라고 함
  - $\Theta(1)$  : 상수 시간(constant time)
  - $\Theta(n^2)$  : 제곱 시간(quadratic time)
  - $\Theta(n^3)$  : 큐빅 시간(cubic time)
- 5개의 상이한 복잡도 클래스의 정의
  - $O(g(n)) = \{ f(n) \mid f(n)/g(n) \text{에 한계값이 주어짐} \}$   
 $g$ 보다 점근적으로 더 *느리거나* 동등한 모든 함수들의 집합
  - $\Omega(g(n)) = \{ f(n) \mid g(n)/f(n) \text{에 한계값이 주어짐} \}$   
 $g$ 보다 점근적으로 더 *빠르거나* 동등한 모든 함수들의 집합
  - $\Theta(g(n)) = \{ f(n) \mid f(n)/g(n) \text{과 } g(n)/f(n) \text{에 한계값이 주어짐} \}$   
 $g$ 와 점근적으로 동등한 모든 함수들의 집합
  - "big oh of  $g(n)$ ", "big omega of  $g(n)$ ", "big theta of  $g(n)$ "이라고 발음

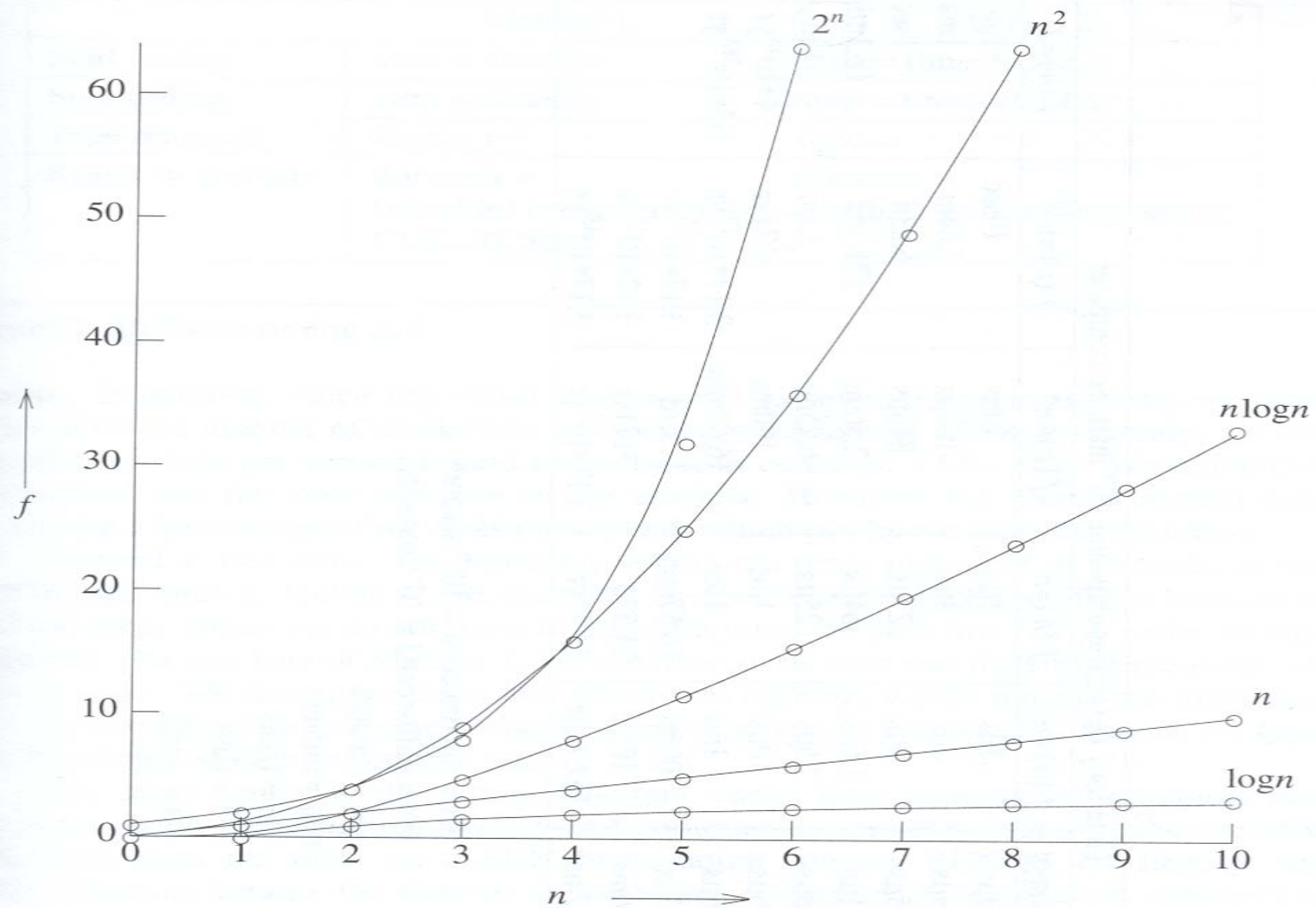


Figure 1.8 Plot of function values

## 3.7 이진 탐색

- 주어진 시퀀스가 정렬이 되어 있을 때
  - 주어진 시퀀스를 반복적으로 반으로 나누어가며,  
각 단계에서 그 범위에 목표를 포함하는 반쪽에 집중
- 알고리즘
  - 입력: 시퀀스와 목표 값  $x$
  - 출력: 인덱스 값  $i$
  - 선조건: 시퀀스는 정렬되어 있음
  - 후조건:  $a_i = x$ ; 또는 모든  $j < p$ 에 대해서  $a_j < x$ 이고 모든  $j \geq p$ 에 대해서  $a_j > x$ 일 때  $i = -p-1$
  - 1.  $p=0, q=n-1$ 로 놓음.
  - 2.  $p \leq q$ 이면 단계 2-5를 반복.
  - 3.  $i = (p+q)/2$ 로 놓음.
  - 4.  $a_i = x$ 이면,  $i$ 를 리턴.
  - 5.  $a_i < x$ 이면,  $p = i+1$ 로 놓음; 그렇지 않으면  $q = i-1$ 로 놓음.
  - 6.  $-p-1$ 을 리턴.



- LISTING 3.6: The Binary Search

```

1 public class TestBinarySearch {
2     public static void main(String[] args) {
3         int[] a = {22, 33, 44, 55, 66, 77, 88, 99};
4         System.out.println("search(a," + 55 + "): " + search(a, 55));
5         System.out.println("search(a," + 50 + "): " + search(a, 50));
6     }
7     static int search(int[] a, int x) {
8         int p = 0, q = a.length-1;
9         while (p <= q) { // search the segment a[p..q]
10             int i = (p+q)/2; // index of element in the middle
11             if (a[i] == x) return i;
12             if (a[i] < x) p = i+1; // search upper half
13             else q = i-1; // search lower half
14         }
15         return -p-1; // not found
16     }
17 }

```

- 출력 결과

search(a,55): 3

search(a,50): -4

- 시간 복잡도:  $\Theta(\lg n)$

## 3.8 java.util.Arrays 클래스

- 배열 조작을 위한 여러 유틸리티 메소드를 제공
- 메소드들이 모두 static 선언으로 되어 있음 (접두어 "Arrays"에 의해 호출됨)
- 메소드의 일부
  - `public static int binarySearch(double[] a, double x)`
  - `public static boolean equals(double[] a, double[] b)`
  - `public static void fill(double[] a, double x)`
  - `public static void fill(double[] a, int lo, int hi, double x)`
  - `public static void sort(double[] a)`

## LISTING 3.7 java.util.Arrays 클래스의 사용

```
1  public class TestJavaUtilArrays {
2
3      public static void main(String[] args) {
4          int[] a = {77, 44, 55, 22, 99, 66, 33, 88};
5          print(a);
6          java.util.Arrays.sort(a);
7          print(a);
8          test(a,55);
9          test(a,60);
10         test(a,88);
11         test(a,90);
12     }
13     static void test(int[] array, int target) {
14         int i = java.util.Arrays.binarySearch(array, target);
15         System.out.print("target="+target+", i="+i);
16         if (i >= 0) System.out.println("Wta[" + i + "]= " + array[i]);
17         else System.out.println("WtInsert " + target+" at a["+(-i-1)+""]");
18     }
```

```
20      static void print(int[] a) {  
21          // See lines 22-27 in Listing 3.1 on page 76  
22      }  
23  }
```

- 출력 결과

a = {77,44,55,22,99,66,33,88}

a = {22,33,44,55,66,77,88,99}

target=55, i=3 a[3]=55

target=60, i=-5 Insert 60 at a[4]

target=88, i=6 a[6]=88

target=90, i=-8 Insert 90 at a[7]

## 3.9 사용자 정의 IntArrays 클래스

- int 배열을 위한 “집에서 만든(homegrown)” IntArrays 클래스: 리스팅 3.8
- `public static boolean isSorted(int[] a);`
- `public static void print(int[] a)`
- `public static int[] randomInts(int n, int range)`
- `public static int[] resize(int[] a, int n)`
- `public static void swap(int[] a, int i, int j)`

## 3.10 java.util.Vector 클래스

- 버전 1.1에서 java.util 패키지에 Vector 클래스 도입
  - 그 이후로 java.util.ArrayList 클래스로 대체
- 크기조정가능 배열(resizable array)에 대해 유용
  - 한 배열을 동일 원소들을 포함하는 더 큰 배열로 대체하는 프로그래밍 기법
    - 새 배열은 기본 배열의 2배의 길이를 갖게 됨

## – 정수 배열의 크기 조정

- LISTING 3.9: Resizing an Integer Array

```
int[ ] resized(int[ ] a) {  
    int n=a.length;  
    int[ ] aa = new int[2*n];  
    System.arraycopy(a,0,aa,0,n);  
    return aa;  
}
```

# Vector 클래스

Vector
<pre>#CAPACITY:int=10 #object:object[] #size:int</pre>
<pre>+vector() +vector(capacity:int) +vector(a:object[]) +size():int +toString():String #resize()</pre>



# 배열을 적재하는 생성자

- LISTING 3.12: A Constructor That Loads an Array

```
1 public Vector(Object[] a) {  
2     int n = a.length;  
3     objects = new Object[2*n];  
4     System.arraycopy(a,0,objects,0,n);  
5     size = n;  
6 }
```

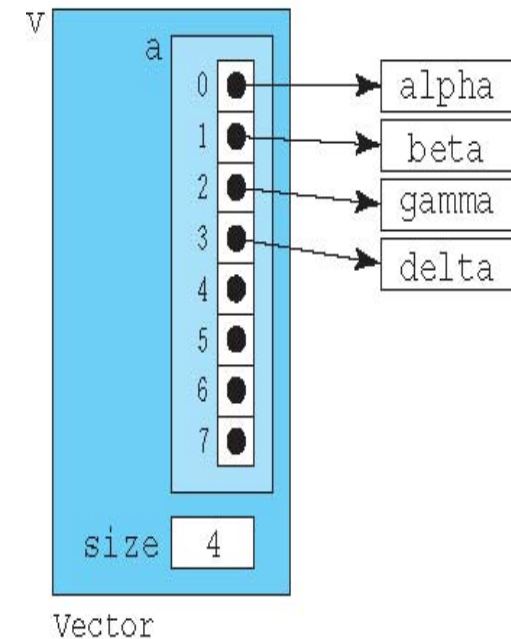
# Vector 클래스의 테스트

- LISTING 3.13: Testing the Vector Class

```
1 public static void main(String[] args){  
2     Vector v = new Vector(args);  
3     System.out.println(v);  
4 }
```

-출력 결과

A: \>java TestVector alpha beta gamma delta  
(alpha,beta,gamma,delta)



## 3.11 다차원 배열

### ■ 배열 b[ ][ ]

- 균일
- 컴포넌트 배열 3개 모두가 4-원소 서브-배열(행)

```
int[][] b = { { 22,44,66,88 },  
              { 0, 0, 0, 0 },  
              { 33,55,77, 0 } };
```

b

	0	1	2	3
0	22	44	66	88
1	0	0	0	0
2	33	55	77	0

int [ ] [ ]

```
System.out.println("b.length: " + b.length);  
IntArrays.print(b[0]);  
IntArrays.print(b[2]);  
System.out.println("b[2].length: " + b[2].length);
```

- 2차원 배열: 배열의 원소가 다시 배열
  - 배열 `a[ ][ ]`
    - 분리된 배열들의 배열
    - 각 컴포넌트 배열의 크기가 다름 -> 울퉁불퉁한 배열 (ragged array)

```
int[][] a = new int[3][];
```

```
a[0] = new int[] {22,44,66,88};
```

```
a[2] = new int[] {33,55};
```

