Chapter 2 - Arrays

Tran Thanh Tung

Objective

- Basics of Array in Java
- Linear search – Binary search
- Storing objects
- Big O notation

Introduction

- How do we store list of integer entered by user?

```
int num1;
int num2;
int num3;
```

- If there are 100 of them !?

Introduction

- Array is the most commonly used data structure
- Built into most of the programming languages

123	412	19	20	

Definition

- An ARRAY is a collection of variables all of the same TYPE
  - Element
  - Index / positions
- In Java

	123	412	19	20	
index	*0*	*1*	*2*	*3*	*4*

Accessing array elements

- Element is access by index number via subscript
- In Java,

  `ArrayName[index]`

- **`A[3], A[0]`**
- **`A[6]`**

A	123	412	19	20	
index	*0*	*1*	*2*	*3*	*4*

Example in Java

```
int A[];
int A1[] = new int[100];
int A2[] = new int { 1, 7, 9, 20};

for (int i=0; i< 100; i++)
     A1[i] = i*2;

for (int j=0; j< A2.length; j++)
     A2[j] = j*2;
```

Operations on array

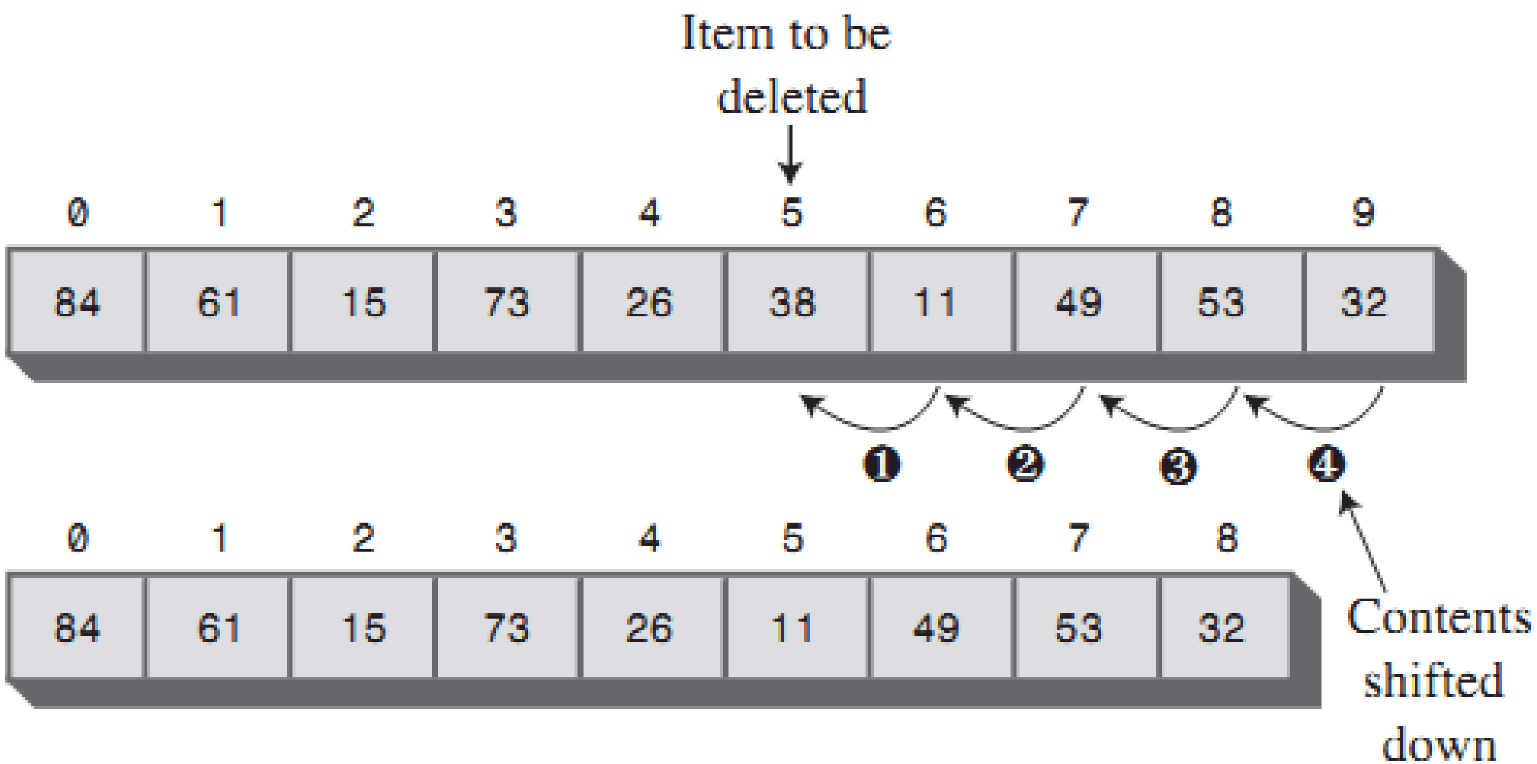
- Insertion
- Searching
- Deletion

(Demo on Array Workshop applet)

- Duplication issue

→ How does it work ?
→ Java code in p.

Delete an item



Multi-dimension array

- A matrix

1	0	3	4
5	1	32	12
6	7	1	10
19	5	4	1

Two-dimension array

- Declaration

```
int [][] matrix = new int
[ROWS][COLUMNS];
int [][] matrix2 =
{
    {1, 2, 3},
    {6, 1, 4},
    {9, 5, 1}
};
```

- Accessing
  - **matrix[0][10];**

Linear searching technique

- Look for ‘20’

A	123	412	19	20	25
index	*0*	*1*	*2*	*3*	*4*

- Step through the array
- Comparing searchKey with each element.
- Reach the end but don’t found matched element
  → Can’t find

Ordered arrays

- Data items are arranged in order of key value.

A	412	123	25	20	19
index	*0*	*1*	*2*	*3*	*4*

A2	19	20	25	123	412
Index	*0*	*1*	*2*	*3*	*4*

Linear searching in ordered array

Look for ‘20’

A	412	123	25	20	19
index	*0*	*1*	*2*	*3*	*4*

A2	19	20	25	123	412
Index	*0*	*1*	*2*	*3*	*4*

Binary searching technique

- Guess the number between 1 and 100

46

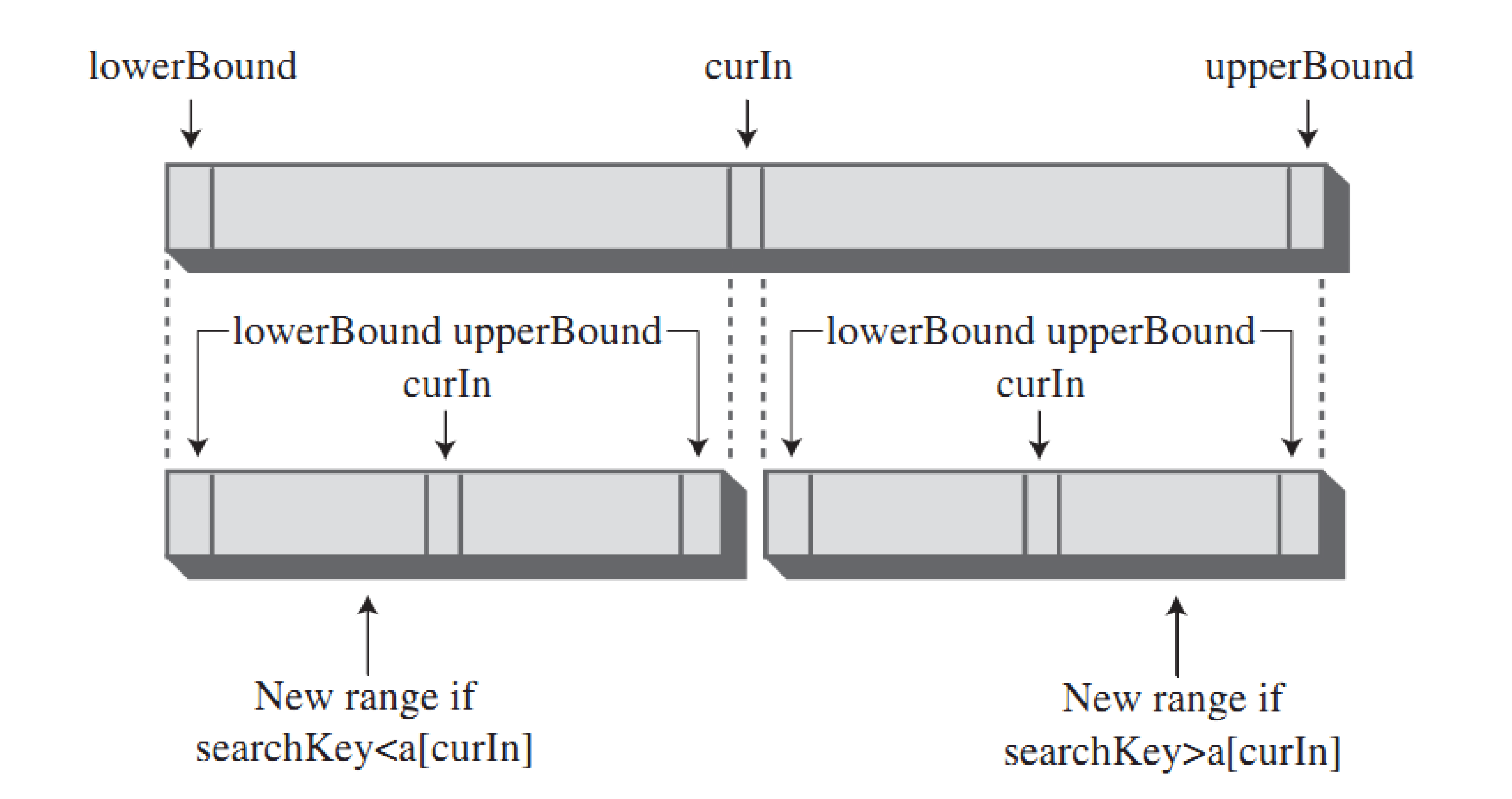
Smaller

Larger

Binary searching technique

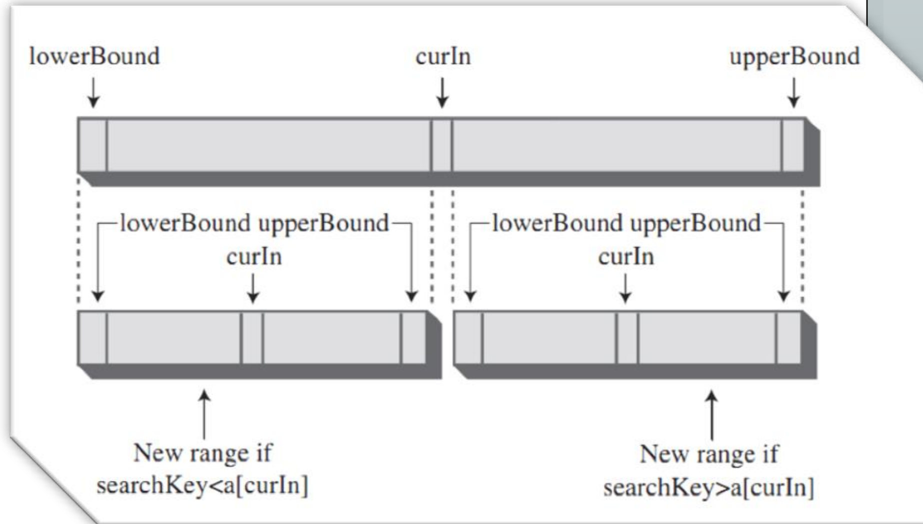
Step #	Number guessed	Result	Range of possible value
0			0-100
[redacted]			

Algorithm



Binary search

Data structures and algorithms in Java – p.57




```
public int find(long searchKey)
   {
   int lowerBound = 0;
   int upperBound = nElems-1;
   int curIn;

   while(true)
      {
      curIn = (lowerBound + upperBound ) / 2;
      if(a[curIn]==searchKey)
         return curIn;              // found it
      else if(lowerBound > upperBound)
         return nElems;             // can't find it
      else                          // divide range
         {
         if(a[curIn] < searchKey)
            lowerBound = curIn + 1; // it's in upper half
         else
            upperBound = curIn - 1; // it's in lower half
         }  // end else divide range
      }  // end while
   }  // end find()
```

In class work

- Modify the Binary search algorithm for a descending array

Advantage of ordered arrays

- Searching time : Good
- Inserting time : Not good
- Deleting time : Not good

- → Useful when
  - Searches are frequent
  - Insertions and deletions are not

Logarithm

- Binary search

→ $Log_2 n$

Range	Comparisons needed
10	4
100	7
1,000	10
10,000	14
100,000	17
1,000,000	20
10,000,000	24
100,000,000	27
1,000,000,000	30

Must known

2^i	n	$\log_2 n$
2^0	1	0
2^1	2	1
2^2	4	2
2^3	8	3
2^4	16	4
2^5	32	5

2^i	n	$\log_2 n$
2^6	64	6
2^7	128	7
2^8	256	8
2^9	512	9
2^{10}	1024	10
2^{11}	2048	11

Storing objects

We need to

- Store a collections of Students
- Search student by Student name
- Insert a new student, delete a student

In class work:

- Read the sample code in p.65-69

Big O notation

- To measure the EFFICIENCY of algorithms
- Some notions
  - Constant
  - Proportional to N
  - Proportional to log(N)
- Big O relationships between time and number of items

O(1) – constant

- The time needed by the algorithm is not depend in size of items
- Example
  - Insertion in an unordered array
  - Any others ?

O(N) – Proportional to N

- Linear search of items in an array of N items
  On average T = K * N /2

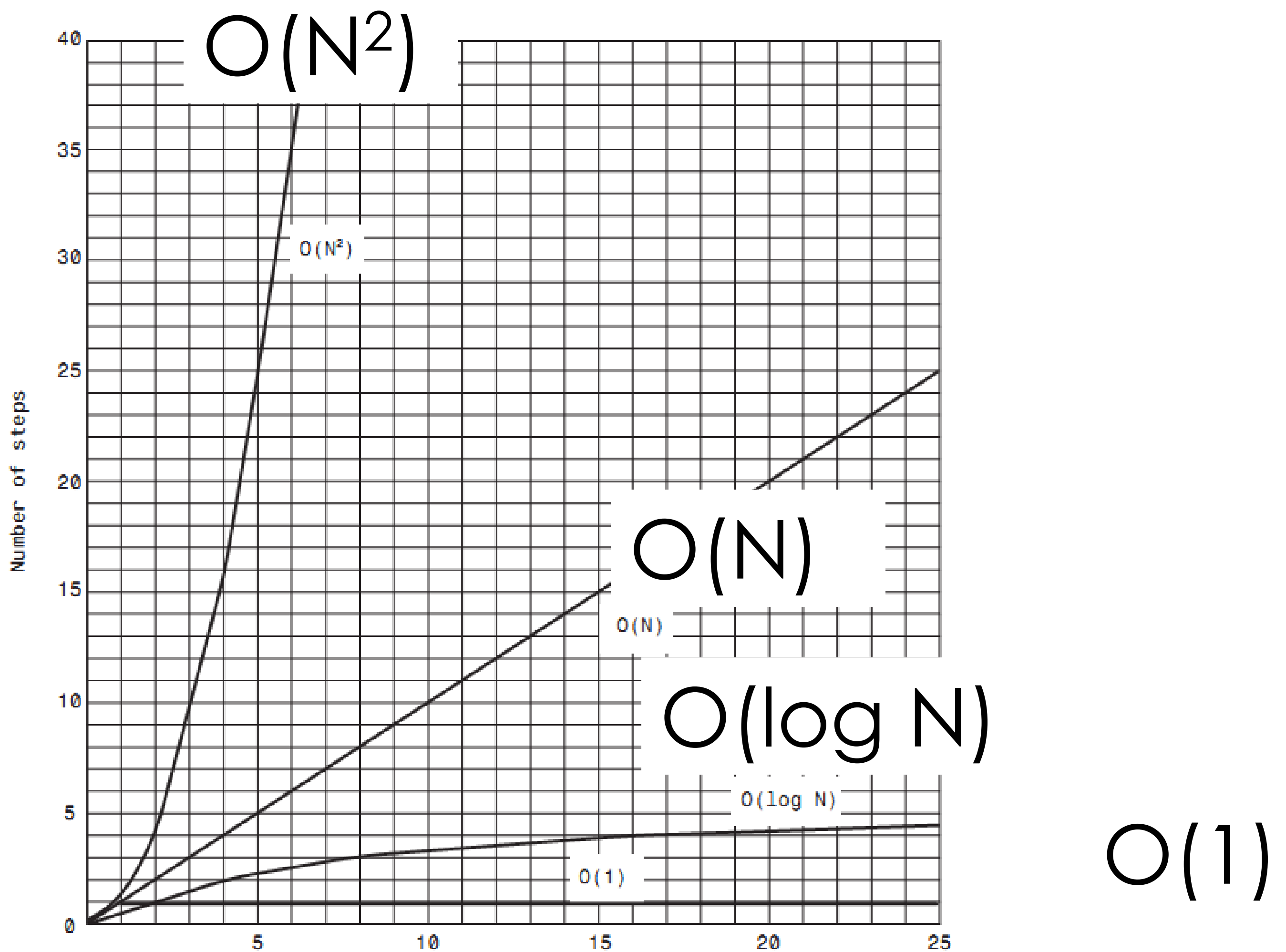
- Average linear search times are proportional to size of array.

- For an array of N' items
  If N' = 2 * N
  Then T' = 2 * T

O(log N) - Proportional to log(N)

- For binary search:
  $T = K * \log_2(N)$

- We can say : $T = K * \log(N)$
  (by lumping the difference between $\log_2$ and log into K)


O(N²)
O(N)
O(log N)
O(1)
Number of steps
O(N²)
O(N)
O(log N)
O(1)
40
35
30
25
20
15
10
5
0
5
10
15
20
25

Summary

- Arrays in Java are objects, created with *new* operator
- Unordered arrays offer
  - fast insertion,
  - but slow searching and deletion
- Binary search can be applied to an ordered array
- Big O notation provides a convenient way to compare the speed of algorithms
- An algorithm that runs in O(1) is the best, O(log N) is good, O(N) is fair and $O(N^2)$ is pretty bad