Java Programming

Zheng-Liang Lu

Department of Computer Science & Information Engineering
National Taiwan University

Online Course

```java
class Lecture5 {

                "Arrays and More Data Structures"

}
```

Arrays

An array stores a large collection of data of same type.

```
1 ...
2         // Assume that the size is known.
3         T[] A = new T[size];
4         // A is a reference point to T-type array.
5 ...
```

- Note that **T** could be any type.
- The variable *size* must be a nonnegative integer for the capacity of arrays.
- We now proceed to look into Line 3 in two stages.

Stage 1: Creation

- First we focus on the RHS of Line 3.
- By invoking the new operator followed by **T** and [ ] surrounding an integer as its size, one array is allocated in the heap.[1]
- Note that the size cannot be changed after allocation.[2]
- In the end, one memory address associated with that array is returned and should be cached.

[1] Recall about the simplified memory model.

[2] What if the array is full?! Stay tuned.

Stage 2: Reference

- We declare one reference of **T**[ ], say A, for the array.
- I strongly emphasize that A is not the array.
- To understand the type correctly, one should read the type from right to left.
- For example, A is a reference to an array (represented by [ ]) whose elements are of **T** type.

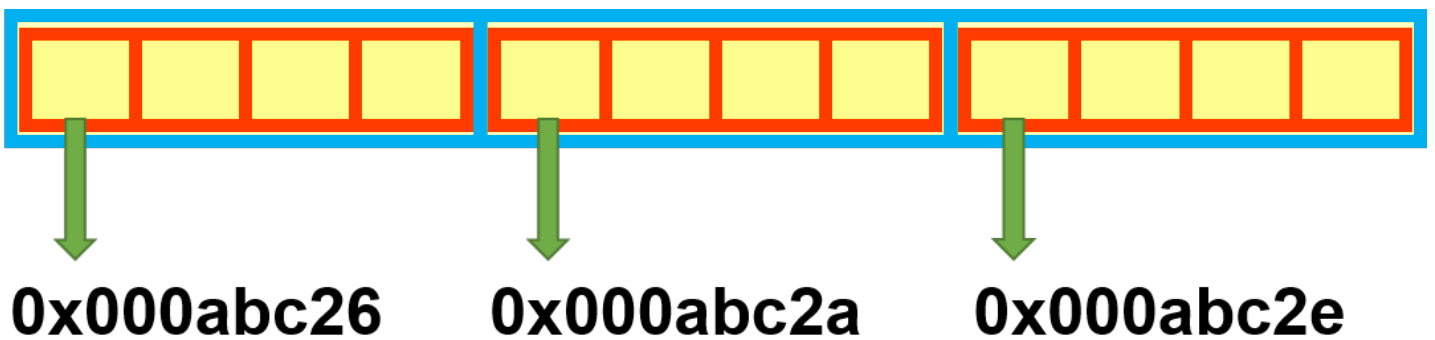
Zero-Based Array Indexing

- Every array starts from 0, but not 1.
- For example, the first element is A[0], the second A[1], the third A[2], and so on.
- Note that the last index of one array is *size* -1.
  - An **ArrayIndexOutOfBoundsException** is thrown out if the index exceeds *size* -1.
- This convention is common among the mainstream languages! (Why?)

Memory Allocation of Arrays

- An array is allocated contiguously in the memory.
- Indeed, we can treat the whole memory as an array.
- For example,

```
1 ...
2         int[] A = new int[3];
3 ...
```



Zero-Based Array Indexing (Concluded)

- To fetch the second element, jump to the address stored by A and shift by 1 unit size of int, denoted by A[1].
- Now you could explain why the first element is denoted by A[0].
- Array index clearly acts as the offset from the beginning of arrays!

Array Initialization

- Every array is implicitly initialized once the array is created.
- Default values are listed below:
  - 0 for all numeric types;
  - \u0000 for char type;
  - false for boolean type;
  - null for all reference types.[3]
- An array can also be created by enumerating all elements without using the new operator, for example,

```
1 ...
2         int[] A = {10, 20, 30}; // Syntax sugar.
3 ...
```

[3] We will visit the keyword null in the chapter of OOP.

Processing Arrays

We often use for loops to process array elements.

- Arrays have an attribute called *length*, which is the array capacity.
  - For example, A.*length*.
- So it is natural to use a for loop to manipulate arrays.

Examples

```
1  ...
2          // Create an integer array of size 5.
3          int[] A = new int[5];
4
5          // Generate 5 random integers ranging from 0 to 99.
6          for (int i = 0; i < A.length; ++i) {
7              A[i] = (int) (Math.random() * 100);
8          }
9
10         // Display all elements of A: O(n).
11         for (int i = 0; i < A.length; ++i) {
12             System.out.printf("%d ", A[i]);
13         }
14         System.out.println();
15 ...
```

```
1 ...
2         // Find maximum and minimum of A: O(n).
3         int max = A[0];
4         int min = A[0];
5         for (int i = 1; i < A.length; ++i) {
6             if (max < A[i]) max = A[i];
7             if (min > A[i]) min = A[i];
8         }
9 ...
```

- How about the locations of extreme values?
- Can you find the 2nd max of A?
- Can you keep the first k max of A?

```
1 ...
2         // Sum of A: O(n).
3         int sum = 0;
4         for (int i = 0; i < A.length; ++i) {
5             sum += A[i];
6         }
7 ...
```

- Calculate the mean of A.
- Calculate the variance of A.
- Calculate the standard deviation of A.

Shuffle Algorithm

```
1  ...
2          for (int i = 0; i < A.length; ++i) {
3
4              // Choose a randon integer j.
5              int j = (int) (Math.random() * A.length);
6
7              // Swap A[i] and A[j].
8              int tmp = A[i];
9              A[i] = A[j];
10             A[j] = tmp;
11
12         }
13 ...
```

- However, this naive algorithm is broken![4]
- How to swap by using XOR (that is, $\wedge$)?

[4] See https://blog.codinghorror.com/the-danger-of-naivete/.

Exercise

Write a program to deal the first 5 cards from a deck of 52 shuffled cards.

- As you can see, RNG produces only random numbers.
- How to shuffle nonnumerical objects?
- Simply label 52 cards by $0, 1, \ldots, 51$.
- Shuffle these numbers!

```
...
        String[] suits = {"Club", "Diamond", "Heart", "Spade"};
        String[] ranks = {"3", "4", "5", "6", "7", "8", "9",
                          "10", "J", "Q", "K", "A", "2"};

        int size = 52;
        int[] deck = new int[size];
        for (int i = 0; i < deck.length; i++)
            deck[i] = i;

        // Shuffle algorithm: correct version.
        for (int i = 0; i < size - 1; i++) {
            int j = (int) (Math.random() * (size - i)) + i;
            int z = deck[i];
            deck[i] = deck[j];
            deck[j] = z;
        }

        for (int i = 0; i < 5; i++) {
            String suit = suits[deck[i] / 13];
            String rank = ranks[deck[i] % 13];
            System.out.printf("%-3s%8s\n", rank, suit);
        }
...
```

Sorting Problem

- In computer science, a sorting algorithm is an algorithm that puts elements of a list in a certain order.[5]
- For example,

```
1 import java.util.Arrays;
2
3 ...
4         int[] A = {5, 2, 8};
5         Arrays.sort(A); // Becomes 2 5 8.
6
7         String[] B = {"www", "csie", "ntu", "edu", "tw"};
8         Arrays.sort(B); // Result?
9 ...
```

[5] What is the natural ordering of things?

Exercise: Bubble Sort

```
1  ...
2          // Bubble sort: O(n ^ 2).
3          boolean swapped;
4          do {
5              swapped = false;
6              for (int i = 0; i < A.length - 1; i++) {
7                  if (A[i] > A[i + 1]) {
8                      int tmp = A[i];
9                      A[i] = A[i + 1];
10                     A[i + 1] = tmp;
11                     swapped = true;
12                 }
13             }
14         } while (swapped);
15 ...
```

- Try to implement the Selection Sort and the Insertion Sort.[6]

[6] See `https://visualgo.net/en/sorting`.

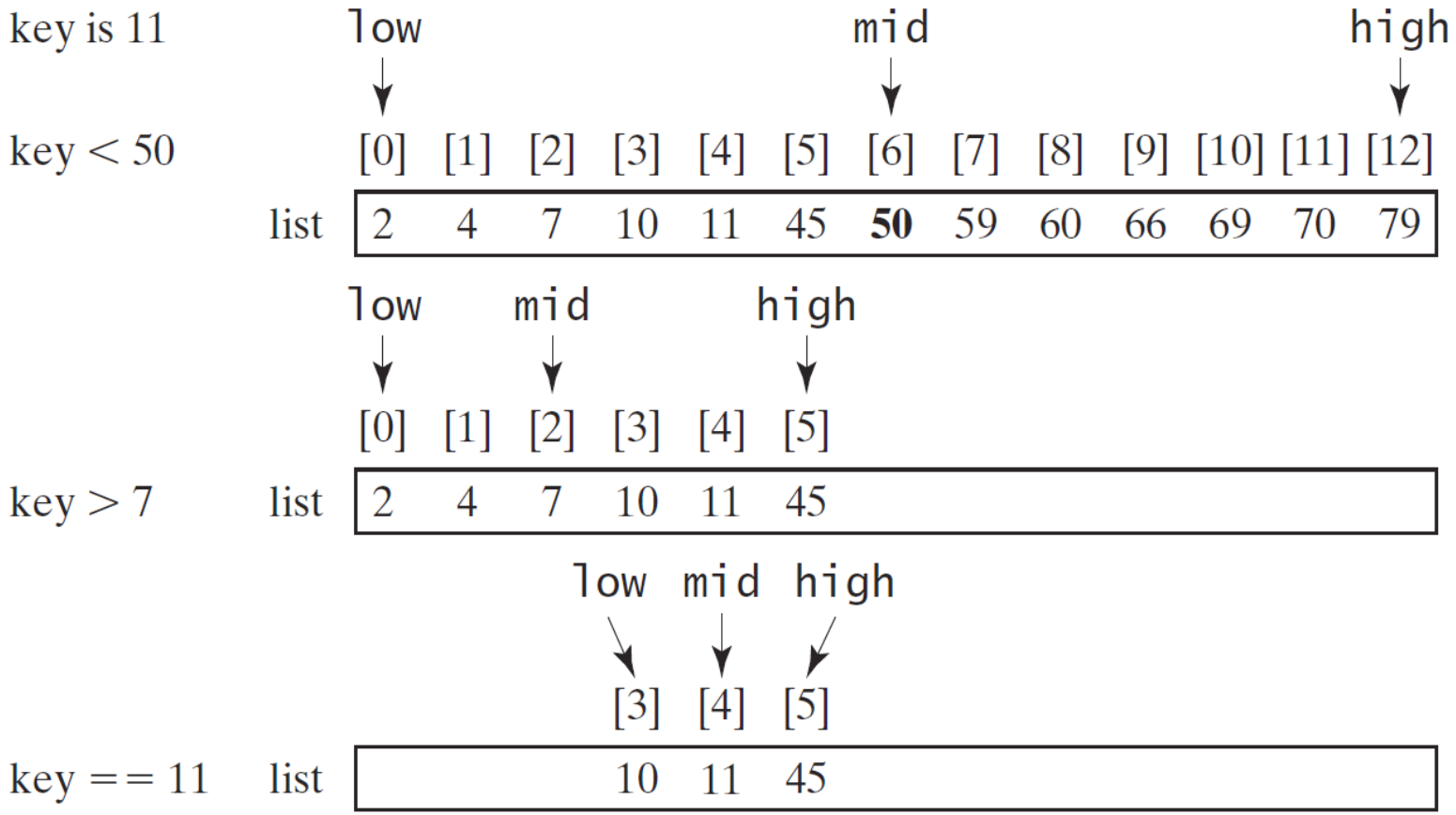
Searching Problem

- To find the location of a given key, the linear search compares the key with all elements sequentially.

```
...
        // Linear search: O(n).
        int[] A = {...};
        int founds = 0;
        for (int i = 0; i < A.length; i++) {
            if (A[i] == key) {
                System.out.printf("%d ", i);
                founds++;
            }
        }
        System.out.println("\nFounds: " + founds);
...
```

- Could we do better?

Alternative: Binary Search (Revisited)



```
1  ...
2          int idx = -1; // Why?
3          int high = A.length - 1, low = 0, mid;
4          while (high > low && idx < 0) {
5              mid = low + (high - low) / 2; // Why?
6              if (A[mid] < key)
7                  low = mid + 1;
8              else if (A[mid] > key)
9                  high = mid - 1;
10             else
11                 idx = mid;
12         }
13
14         if (idx > -1)
15             System.out.printf("%d: %d\n", key, idx);
16         else
17             System.out.printf("%d: not found\n", key);
18 ...
```

- However, the binary search works only when the data is sorted!

Discussions

- If the data is immutable, sort the data once for all and then do binary search.
- What if the data may be changed all the time?

Scenario / Operation	Insert	Search
Immutable unsorted array	N/A	$O(n)$
Immutable sorted array	N/A	$O(\log n)$
Mutable unsorted array	$O(1)^*$	$O(n)$
Mutable sorted array	$O(n)$	$O(\log n)$

*: insert by attaching behind the array.

- Note that big-O is additive by simply keeping the most dominant term.
- For example, $O(n) + O(\log n) = O(n)$.

Short Introduction to Data Structures

- A data structure is a particular way of organizing data in a program so that it can perform efficiently.[7]
- The choice among data structures depends on applications.
- As an alternative to arrays, linked lists[8] are used to store data in the way different from arrays.
- You may see plenty of data structures in the future.[9]
  - For example, priority queues, trees, graphs, tables.
- You could also find many questions about data structures on LeetCode.[10]

[7] See `http://bigocheatsheet.com/`.
[8] See `https://en.wikipedia.org/wiki/Linked_list`.
[9] See `https://en.wikipedia.org/wiki/Java_collections_framework`.
[10] See `https://leetcode.com/`.

Special Issue: for-each Loops

- A for-each loop is designed to iterate over a collection of objects, such as arrays and other data structures, in strictly sequential fashion, from start to finish.

```
1 ...
2         T[] A = { ... };
3         for (T element: A) {
4             // Loop body.
5         }
6 ...
```

Example

```
1 ...
2         int s = 0;
3         for (int i = 0; i < A.length; ++i) {
4             s += A[i];
5         }
6 ...
```

```
1 ...
2         int s = 0;
3         for (int item: A) {
4             s += item;
5         }
6 ...
```

- Short and sweet!
- You may consider using the for-each loop when you iterate over all elements and the order of iteration is irrelevant.

Exercise

```
1 ...
2         String[] letters = {"A", "B", "C", "D", "E"};
3
4         for (String letter: letters) {
5             System.out.printf("%s ", letter);
6         }
7         System.out.println();
8 ...
```

Special Issue: Cloning Arrays

- In practice, one might duplicate an array for some purpose.
- For example,

```
1  ...
2          int x = 1;
3          int y = x; // You can say that y copies the value of x.
4          x = 2;
5          System.out.println(y); // Output 1.
6
7          int[] A = {10, ...}; // Ignore the rest of elements.
8          int[] B = A;
9          A[0] = 100;
10         System.out.println(B[0]); // Output?
11 ...
```

- This is called the shallow copy.
- As you can see, the result differs from our expectation. (Why?)

- To clone an array, you should create a new array and use loops to copy every element, one by one.

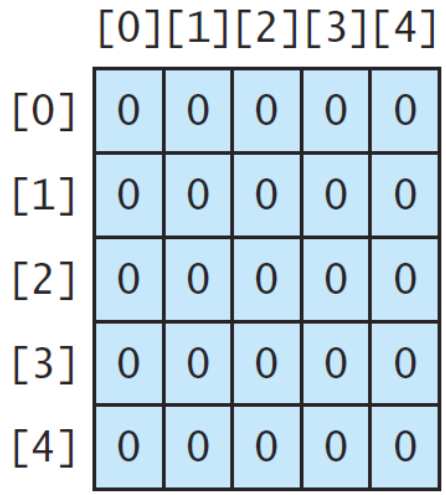
```
1 ...
2         // Let A be an array to be copied.
3         int[] B = new int[A.length];
4         for (int i = 0; i < A.length; ++i) {
5             B[i] = A[i];
6         }
7 ...
```

- This is called the deep copy.

Beyond 1-Dimensional Arrays

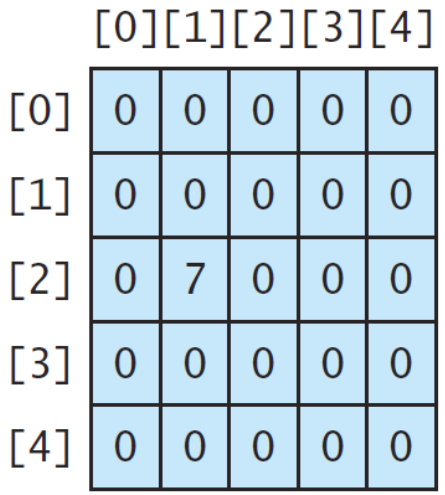
- 2D or higher dimensional arrays are widely used in various applications.
  - For example, RGB images are stored as 3D arrays.
- We can create 2D **T**-type arrays simply by adding one more [ ] with its size.
- For example,

```
1 ...
2         int rows = 4; // Row size.
3         int cols = 3; // Column size.
4         T[][] M = new T[rows][cols];
5 ...
```



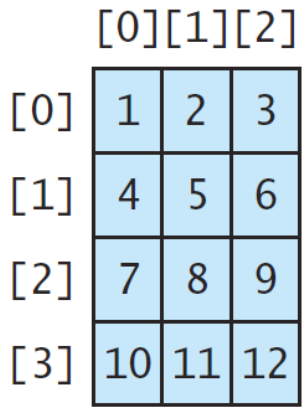

```
matrix = new int[5][5];
```

(a)




```
matrix[2][1] = 7;
```

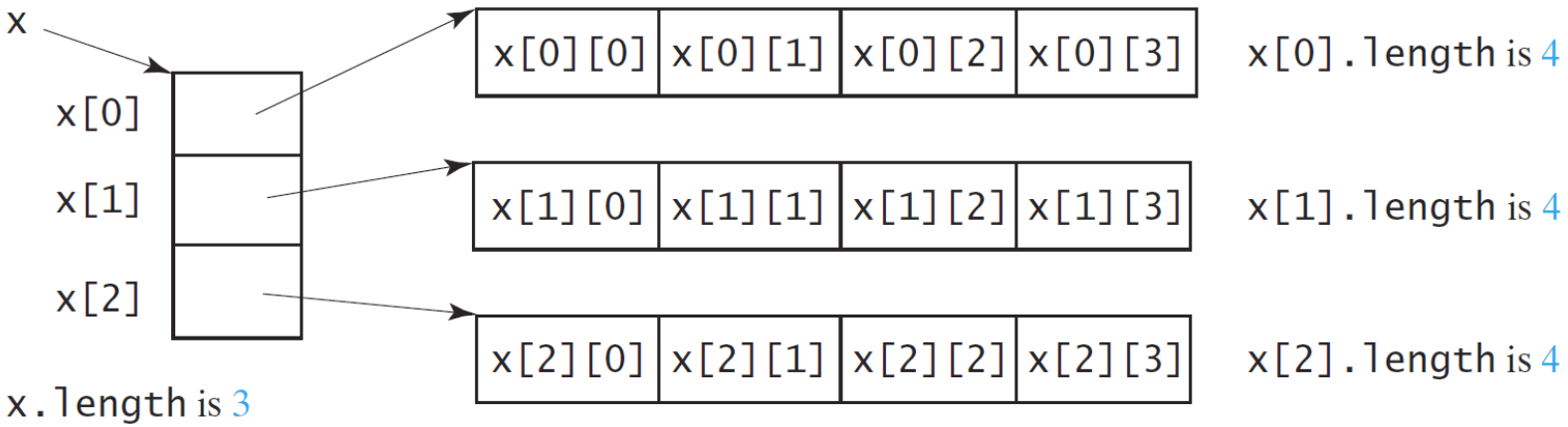
(b)



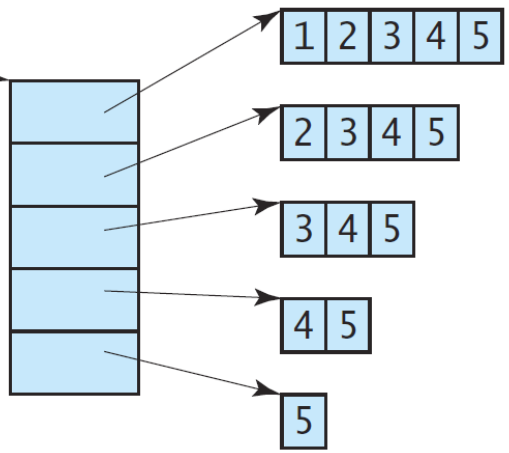

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

(c)

Reality: Memory Allocation for 2D Arrays




```
int[][] triangleArray = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```



Example: 2D Arrays & Loops[11]

```
1  ...
2          int[][] A = {{10, 20, 30}, {40, 50}, {60}};
3
4          // Conventional for loop.
5          for (int i = 0; i < A.length; i++) {
6              for (int j = 0; j < A[i].length; j++)
7                  System.out.printf("%3d", A[i][j]);
8              System.out.println();
9          }
10
11         // For-each loop.
12         for (int[] row: A) {
13             for (int item: row)
14                 System.out.printf("%3d", item);
15             System.out.println();
16         }
17 ...
```

[11] Thanks to a lively discussion on January 31, 2016.

Exercise: Matrix Multiplication

Let $A_{m\times n}$ and $B_{n\times q}$ be two matrices for $m, n, q \in \mathbb{N}$. Write a program to calculate $C = A \times B$.

- Let a_{ik} and b_{kj} be elements of A and B, respectively.
- For $k = 1, 2, \ldots, n$, use the formula

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

  for $i = 1, 2, \ldots, m$ and for $j = 1, 2, \ldots, q$.
- It takes $O(n^3)$ time. (Why?)

(Native) Array v.s. **ArrayList**

```
1  ...
2          int[] A = new int[3]; // The size should be preset.
3          A[0] = 100; A[1] = 200; A[2] = 300;
4          for (int item: A) System.out.printf("%d ", item);
5          System.out.println();
6
7          ArrayList<Integer> B = new ArrayList<>(); // Size?
8          B.add(100); B.add(200); B.add(300);
9          System.out.println(B); // Short and sweet!
10 ...
```

- Native array is the most fundamental data structure, but not convenient.
  - How to resize the array which is full?
- In practice, you should use **ArrayList**<E>, where E is the type given by users.[12]

[12] This is called generics. Stay tuned in Java Programming 2.

Case Study: Reversing Array

- Write a program to arrange the array in reverse order.
- Let A be the original array.
- The first try is to create another array with same size and copy each element from A to B, which is a reference to the new array.

```
1 ...
2         int[] A = {1, 2, 3, 4, 5};
3         int[] B = new int[A.length];
4         for (int i = 0; i < A.length; i++) {
5             B[A.length - 1 - i] = A[i];
6         }
7         A = B; // Why?
8 ...
```

Another Try

```
1 ...
2         int[] A = {1, 2, 3, 4, 5};
3         for (int i = 0; i < A.length / 2; i++) {
4             int j = A.length - 1 - i;
5             int tmp = A[i];
6             A[i] = A[j];
7             A[j] = tmp;
8         }
9 ...
```

Approach	Time Complexity	Space Complexity
First try	$O(n)$	$O(n)$
Second try	$O(n)$	$O(1)$

- The second try is better, both in time[13] and space.
- It is called an in-place algorithm.

[13] The second try runs in only half time of the first one, in practice.