C# Programming

Zheng-Liang Lu

Department of Computer Science & Information Engineering
National Taiwan University

Online Course

```
1 class Lecture5
2 {
3                     "Arrays and More Data Structures"
4 }
```

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

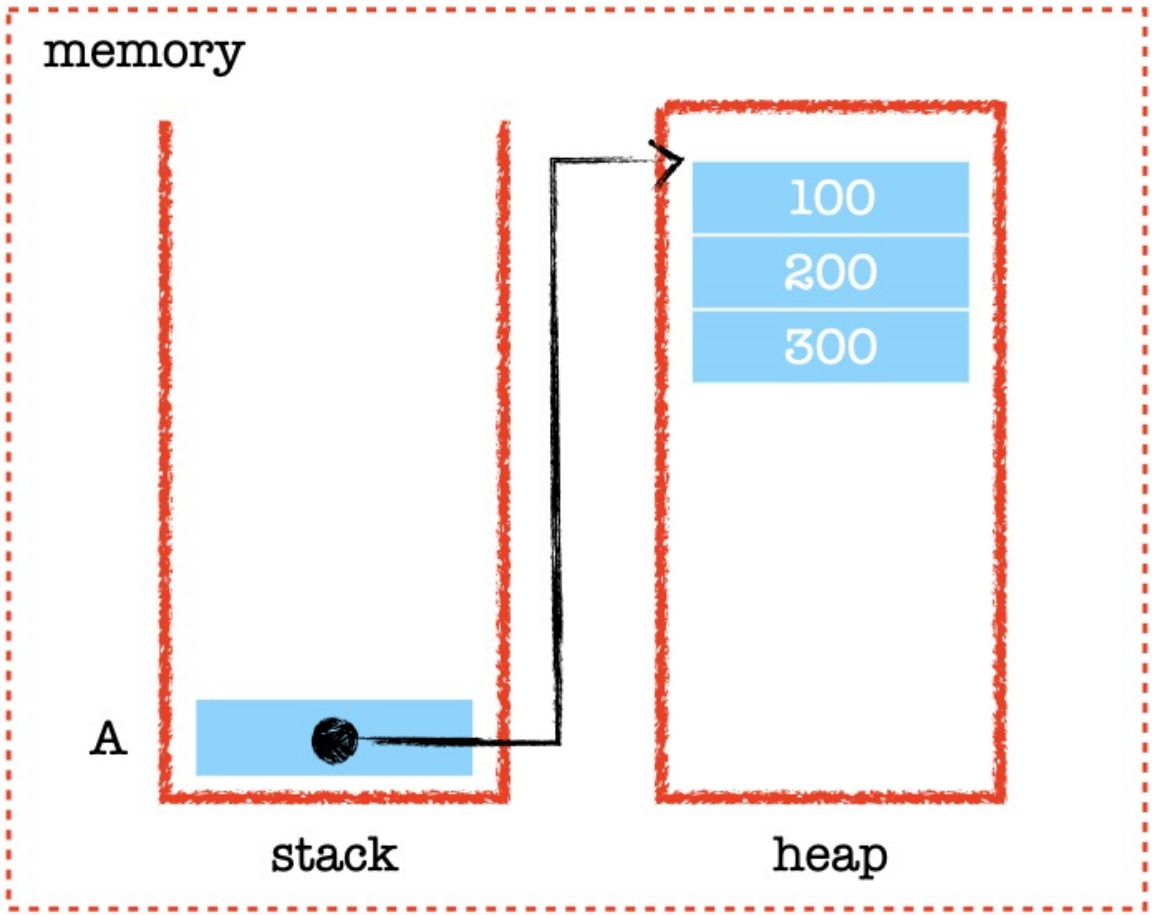
- First we focus on the RHS of Line 3.
- By invoking the new operator followed by **T** and [ ] surrounding an integer as its size, one array is allocated in the heap.
- Note that the size cannot be changed after allocation.[1]
- In the end, one memory address associated with that array is returned and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

[1] 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.

Simplified Memory Model



Zero-Based Array Indexing

- An array is allocated contiguously in the memory.
  - Indeed, we can treat the whole memory as an array.
- 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 **IndexOutOfRangeException** is thrown out if the index exceeds $size - 1$.

- 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!
- This convention is common among the mainstream languages! (Why?)

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 bool type,
  - null for all reference types.[2]
- 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 ...
```

[2] 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          Random rng = new Random();
7          for (int i = 0; i < A.Length; ++i)
8              A[i] = rng.Next(100);
9
10         // Display all elements of A: O(n).
11         for (int i = 0; i < A.Length; ++i)
12             Console.Write("{0, 3}", A[i]);
13         Console.WriteLine();
14 ...
```

```
...
        // Find maximum and minimum of A: O(n).
        int max = A[0];
        int min = A[0];
        for (int i = 1; i < A.Length; ++i)
        {
            if (max < A[i]) max = A[i];
            if (min > A[i]) min = A[i];
        }
...
```

- 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 ...
```

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

Special Issue: foreach Loops

- A foreach 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.

```
...
        T[] A = { ... };
        foreach (T element in A)
        {
            // Loop body.
        }
...
```

Example

```
1 ...
2         int s = 0;
3         for (int i = 0; i < A.Length; ++i)
4         {
5             s += A[i];
6         }
7 ...
```

```
1 ...
2         int s = 0;
3         foreach (int item in A)
4         {
5             s += item;
6         }
7 ...
```

- 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         foreach (string letter in letters)
5         {
6             Console.Write("{0} ", letter);
7         }
8         Console.WriteLine();
9 ...
```

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          Console.WriteLine(y); // Output 1.
6
7          int[] A = {10, ...}; // Ignore the rest of elements.
8          int[] B = A;
9          A[0] = 100;
10         Console.WriteLine(B[0]); // Output?
11 ...
```

- This is called 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         {
6             B[i] = A[i];
7         }
8 ...
```

- This is called deep copy.

Shuffling Algorithm

```
1  ...
2          // Shuffle algorithm.
3          for (int i = 0; i < A.Length; ++i)
4          {
5              // Choose a randon integer j.
6              int j = rng.Next(A.Length);
7              // Swap A[i] and A[j].
8              int tmp = A[i];
9              A[i] = A[j];
10             A[j] = tmp;
11         }
12 ...
```

- However, this naive algorithm is broken![3]
- How to swap by using XOR (that is, $\wedge$)?

[3] 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[] deck = new int[52];
        for (int i = 0; i < deck.Length; i++)
        {
            deck[i] = i;
        }
        // Shuffle algorithm: correct version.
        Random rng = new Random();
        for (int i = 0; i < deck.Length - 1; i++)
        {
            int j = rng.Next(deck.Length - i) + i;
            int tmp = deck[i]; deck[i] = deck[j]; deck[j] = tmp;
        }
        for (int i = 0; i < 5; i++)
        {
            Console.WriteLine("{0, 2} {1, 8}", ranks[deck[i] % 13]
                                             , suits[deck[i] / 13]);
        }
...
```

Sorting Problem

- In computer science, a sorting algorithm is an algorithm that puts elements of a list in a certain order.[4]
- For example,

```
1 ...
2         int[] A = {5, 2, 8};
3         Array.Sort(A); // Becomes 2 5 8.
4
5         string[] B = {"www", "csie", "ntu", "edu", "tw"};
6         Array.Sort(B); // Result?
7 ...
```

[4] What is the natural ordering of things?

Exercise: Bubble Sort

```
...
        // Bubble sort: O(n ^ 2).
        bool swapped;
        do
        {
            swapped = false;
            for (int i = 0; i < A.Length - 1; ++i)
            {
                if (A[i] > A[i + 1])
                {
                    int tmp = A[i]; A[i] = A[i + 1]; A[i + 1] = tmp;
                    swapped = true;
                }
            }
        }
        while (swapped);
...
```

- Try to implement the Selection Sort and the Insertion Sort.[5]

[5] See https://visualgo.net/en/sorting.

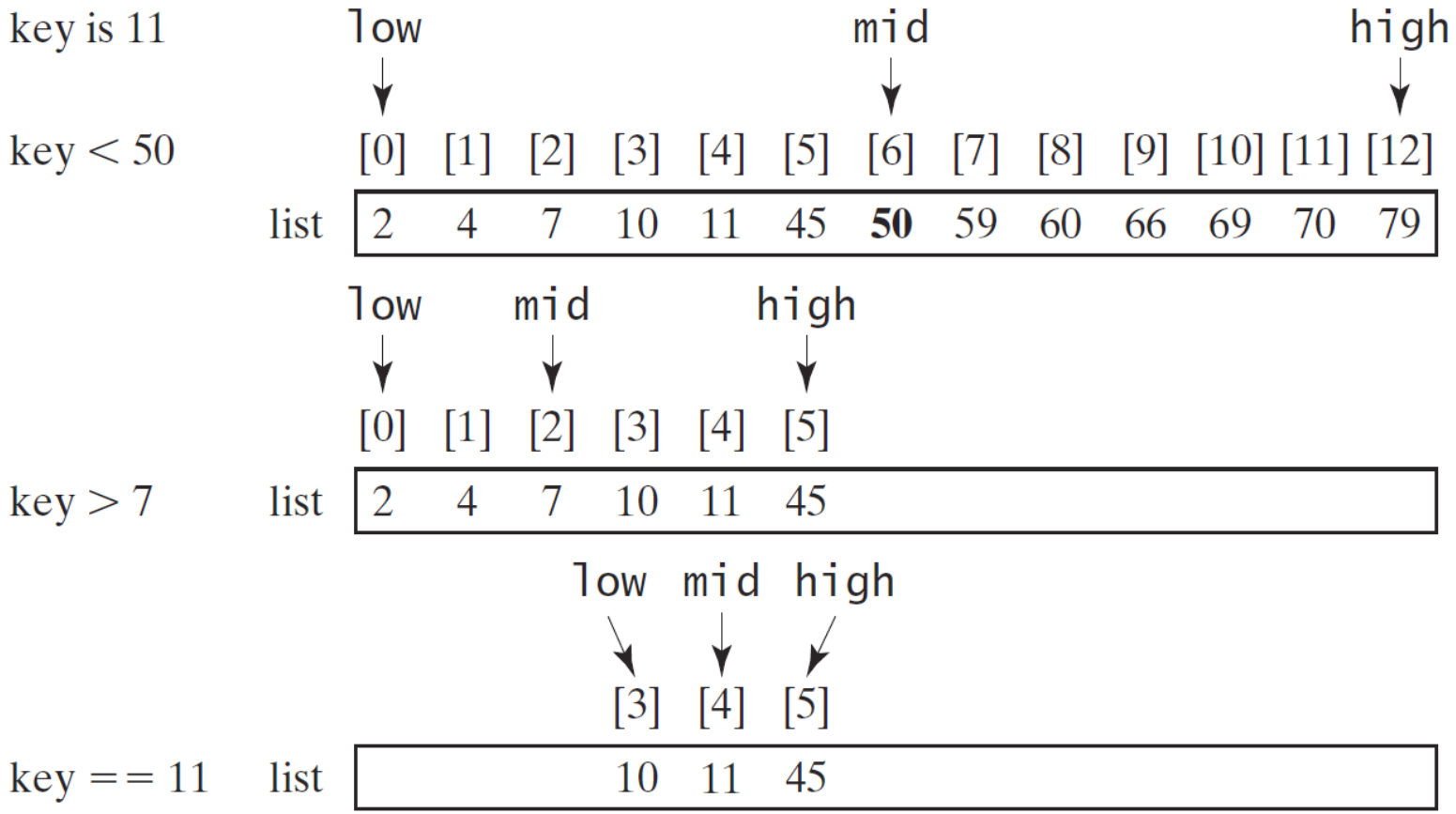
Searching Problem

- 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)
            {
                Console.Write("{0} ", i);
                founds++;
            }
        }
        Console.WriteLine("\nFounds: {0}", founds);
...
```

- Could we do better?

Alternative: Binary Search (Revisited)



```
...
        int idx = -1; // Why?
        int high = A.Length - 1, low = 0, mid;
        while (high > low && idx < 0)
        {
            mid = low + (high - low) / 2; // Why?
            if (A[mid] < key)
                low = mid + 1;
            else if (A[mid] > key)
                high = mid - 1;
            else
                idx = mid;
        }

        if (idx > -1)
            Console.WriteLine("{0}: {1}", key, idx);
        else
            Console.WriteLine("{0}: not found", key);
...
```

- 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, $O(n) + O(\log n) = O(n)$.

About Data Structures

- A data structure is a particular way of organizing data in a program so that it can perform efficiently.[6]
- The choice among data structures depends on applications.
- As an alternative to arrays, linked lists[7] are used to store data in the way different from arrays.
- You may see plenty of data structures in the future.[8]
  - For example, priority queues, trees, graphs, tables.
- You could also find many questions about data structures on LeetCode.[9]

---

[6] See `http://bigocheatsheet.com/`.

[7] See `https://en.wikipedia.org/wiki/Linked_list`.

[8] See `https://docs.microsoft.com/en-us/dotnet/standard/collections/`.

[9] See `https://leetcode.com/`.

Multidimensional Arrays

- 2D or higher dimensional arrays are widely used in various applications.
  - For example, RGB images are stored as 3D arrays.
- We now proceed to create 2D arrays in C#.
- For example,

```
1 ...
2         int[,] M1 = new int[2, 3];
3         M1[0, 0] = 10; M1[0, 1] = 20; M1[0, 2] = 30;
4         M1[1, 0] = 40; M1[1, 1] = 50; M1[1, 2] = 60;
5
6         // The above procedure can be simplified by below.
7         int[,] M2 = new int[,] {{10, 20, 30},
8                                 {40, 50, 60}};
9 ...
```

How to Iterate Through Multidimensional Arrays

```
1  ...
2          // Dimension 0: the number of rows.
3          // Dimension 1: the number of cols.
4
5          for (int i = 0; i < M1.GetLength(0); i++)
6          {
7              for (int j = 0; j < M1.GetLength(1); j++)
8              {
9                  Console.Write("{0, 3}", M1[i, j]);
10             }
11             Console.WriteLine();
12         }
13 ...
```

- As you can see, use the method GetLength() with the specified dimension index to get the size of the dimension.

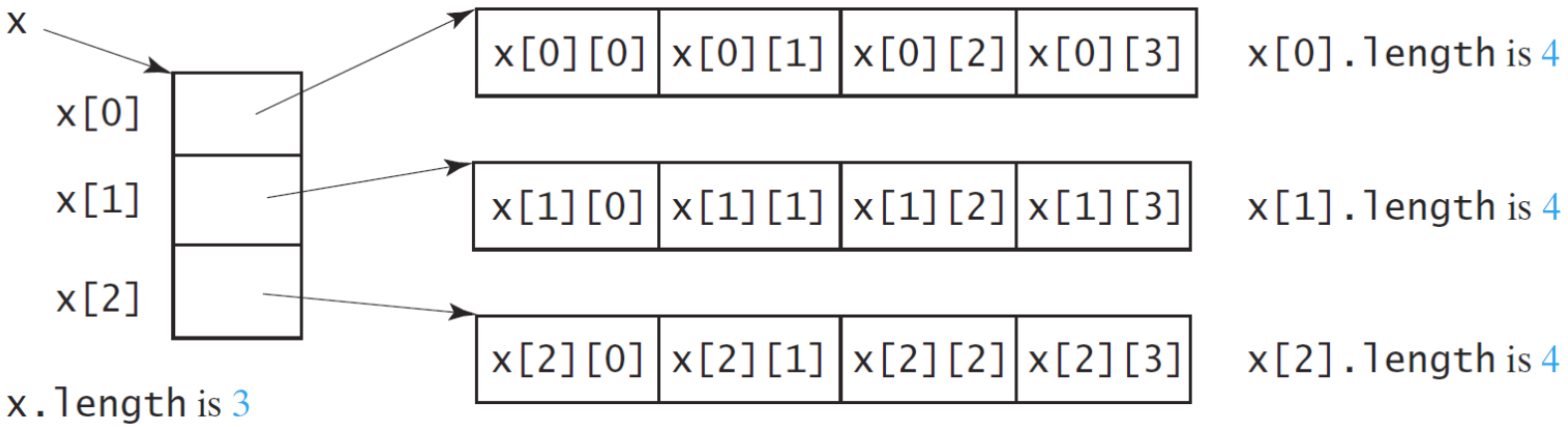
Jagged Arrays: Arrays of Arrays

- For example, we can create a 4×3 integer matrix by

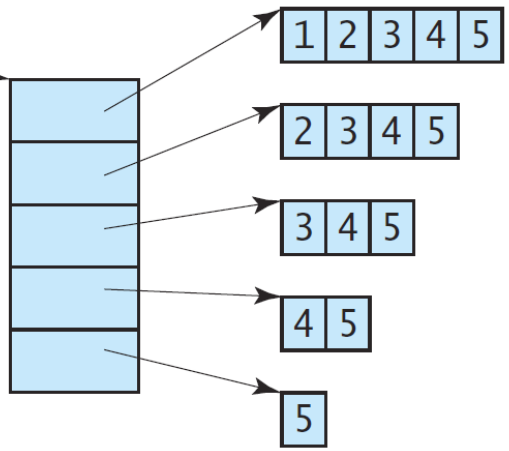
```
1  ...
2          int rows = 4; // Row size.
3          int cols = 3; // Column size.
4          T[][] M = new T[rows][];
5
6          for (int i = 0; i < M.Length; ++i)
7          {
8              M[i] = new int[cols];
9          }
10 ...
```

- Why brother?
- The elements of a jagged array can be of different dimensions and sizes!

Memory Allocation for Jagged Arrays




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



How to Iterate Through Jagged Arrays

```
1  ...
2          int[][] A = new int[][] { new int[] {10, 20, 30},
3                                    new int[] {40, 50},
4                                    new int[] {60}           };
5          // Conventional for loop.
6          for (int i = 0; i < A.Length; i++)
7          {
8              for (int j = 0; j < A[i].Length; j++)
9                  Console.Write("{0} ", A[i][j]);
10             Console.WriteLine();
11         }
12         // Foreach loop.
13         foreach (int[] row in A)
14         {
15             foreach (int item in row)
16                 Console.Write("{0} ", item);
17             Console.WriteLine();
18         }
19 ...
```

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?)

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         A = B; // Why?
7 ...
```

Another Try

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

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[10] and space.
- It is called the in-place algorithm.

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