Data structures and algorithms Spring 2025

Lecture 1: Introduction to Data Structures and Algorithms

Lecturer: Do Thuy Duong

Outline

- Basic course information
- Introduction to Data structures and algorithms

Basic course information

- Faculty: Information Technology
- Unit code: 61FIT2DSA
- Unit name: Data Structure and Algorithm
- Level: Undergraduate
- Units of credit: 5
- Prerequisite: PR1
- Suggested study: 5 hours per week
- Year: Spring 2025

Basic course information

- Lecturer: Ms. Do Thuy Duong
- Tutor: Ms. Do Thuy Duong (duongdt@hanu.edu.vn)

  Ms. Sai Thi Phuong Ly (lystp@hanu.edu.vn)

Course homepage

- https://lms.fit.hanu.vn/course/view.php?id=365
- Check frequently for homework, assignments and announcements

Basic course information

- Textbook:
  - Mark Allen Weiss, “Data Structures and algorithm analysis in Java”, 3rd edition, Pearson, 2012
- Reference:
  - Thomas H. Cormen, “Introduction to Algorithms”, 3rd edition, The MIT Press, 2009

Assessments

- Attendance: 0% (students must attend at least 80% of classes in order to be allowed to take the final exam)
- Discussion: 10%
  - Answer questions in lecture & tutorial section
- Midterm Exam: 30%
- Final Exam: 60%

Basic course information

- Lecture classes focus on theory
- Tutorial classes focus on programming
- Homework: solved by yourself to develop necessary skills
- Programming language: Java

Course info: Resources

- Check the website frequently for resources
- Java IDE:
  - BlueJ
  - Netbeans
  - Eclipse (used in this course)

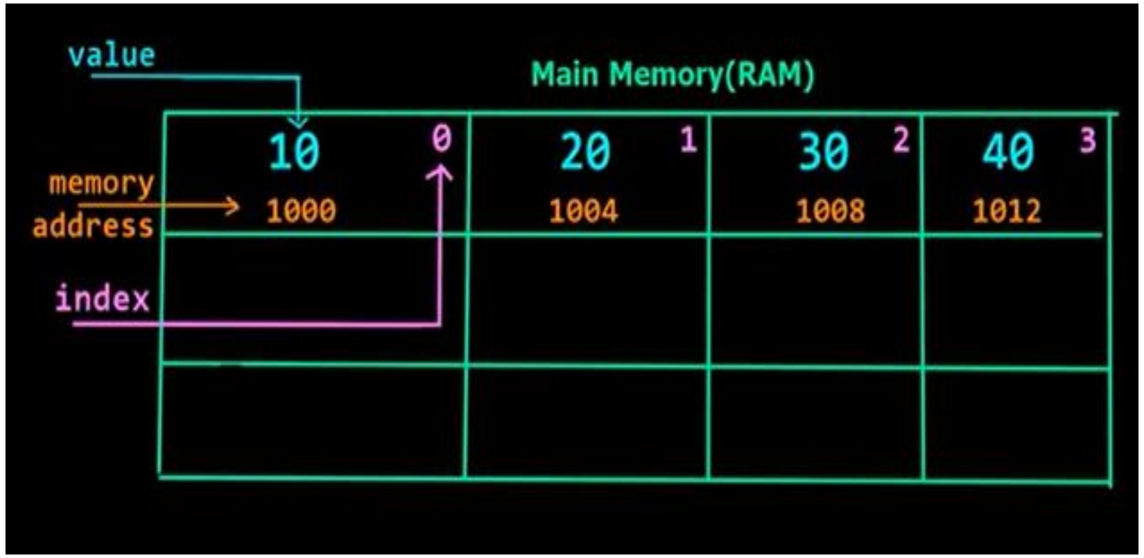
Introduction to DSA

- a Computer Program = Data Structures + Algorithms
- What is Data Structure?
- What is Algorithm?

What is Data structures?

- Definition: A data structure is a particular **way of organizing data** in a computer so that we can perform operations on these data in an effective way.
- You can arrange your data in many ways, each of which organizes and stores it in a unique format within your computer's memory. Your choice will affect how fast and easily you can perform tasks with the data.
- E.g: we can store a list of items having the same data type using the *array* data structure.

int numbers[4] = {10, 20, 30, 40};



Types of data structures?

1. Linear data structures

Linear data structures organize data in sequence, with each element arranged on a single level and connected to elements on either side. In these structures, you perform operations such as insertion or deletion in a linear sequence.

- Linked lists: A linked list is a sequence of elements where each one contains reference information for the element next to it. With linked lists, you can efficiently insert and delete elements, easily adjusting the size of your list.
- Arrays: An array is a group of data elements arranged in adjacent memory locations. The index gives the direct address of each element, making arrays a highly efficient data structure for accessing different data pieces. Arrays are common across computer science functions as a convenient way to store accessible data.

Types of data structures

- Stacks: Stack structures follow the last in, first out (LIFO) principle. The addition (push) and removal (pop) of elements happen only at one end, referred to as the top of the stack.
- Queues: A queue structure uses the first in, first out (FIFO) principle, meaning that you add elements from the back (enqueue) and remove them from the front (dequeue).

Types of data structures

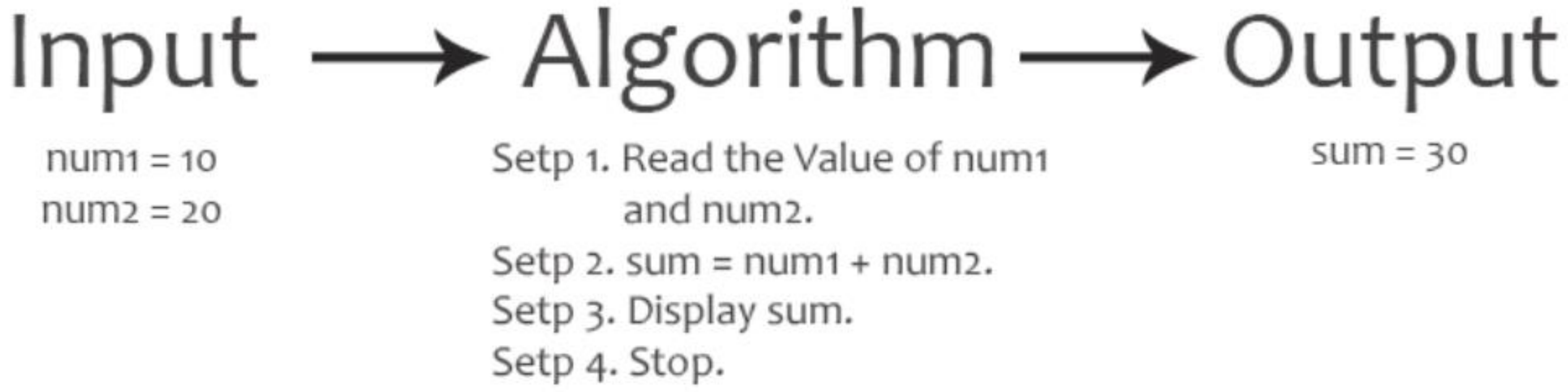
2. Nonlinear data structures

Nonlinear data structures do not organize data sequentially. Instead, they form a hierarchical arrangement where one element connects to one or more elements, leading to a branching structure.

- Graphs: Graphs are fundamental types of nonlinear data structures. A graph is a collection of nodes connected by edges. It can represent networks, such as social networks or transportation systems. However, graphs are often chosen to represent networks of information, such as social media relationships or geographical maps.
- Trees: A tree is a graph structure with a hierarchy of nodes. The top node is the "root," and the descendants of this node are "children." Nodes with no children are "leaves."You can use trees in various applications, including hierarchical data representation, databases, and searching algorithms.

What is an algorithm?

- Algorithm is a step-by-step procedure for solving a problem in a finite amount of time. An algorithm **manipulates data**.
- Algorithm is not the complete code or program, it is just the core logic of a problem.



Setp 1. Read the Value of num1 and num2.
Setp 2. sum = num1 + num2.
Setp 3. Display sum.
Setp 4. Stop.

Algorithum for Sum of Two Numbers

What is an algorithm?

- An algorithm possesses the following properties:
  - It must be **correct**.
  - It must be composed of **precisely defined** steps.
  - The order of the steps must be **precisely defined**.
  - It must be composed of a **finite** number of steps.
  - It must **terminate** for all inputs.
- A *computer program* is an instance or concrete representation, for an algorithm in some programming language.

What is algorithm?

- Weiss' Definition [Text book p.29]:

A clearly specified set of simple instructions to be followed to solve a problem

- **Shampoo Algorithm**

Step 1: Wet hair
Step 2: Lather
Step 3: Rinse
Step 4: Repeat

- Is this correct algorithm?

Which step will be repeated?

What is an algorithm?

- **Shampoo Algorithm (Revision #1)**

Step 1: Wet hair
Step 2: Lather
Step 3: Rinse
Step 4: Repeat Steps 1-4

How many times do we repeat step 1? Infinitely? Or at most 2 times?

What is an algorithm?

- Keep a count of the number of times to repeat the steps
- Repeat the algorithm at most 2 times

- **Shampoo Algorithm (Revision #2)**

```
Step 1: Set count = 0
Step 2:Wet hair
Step 3: Lather
Step 4: Rinse
Step 5: Increment count (increase its value by 1)
Step 6: If count < 2 repeat steps 2-6
Step 7: EXIT
```

- Will execute steps 2-6 **two** times

Why need DSA?

- Requirements for a good software
  - ✓ Clean Design
  - ✓ Easy maintenance
  - ✓ Reliable (no core dumps)
  - ✓ Easy to use
  - ✓ Fast

→ **Efficient data structures**

→ **Efficient algorithms**

Why need DSA?

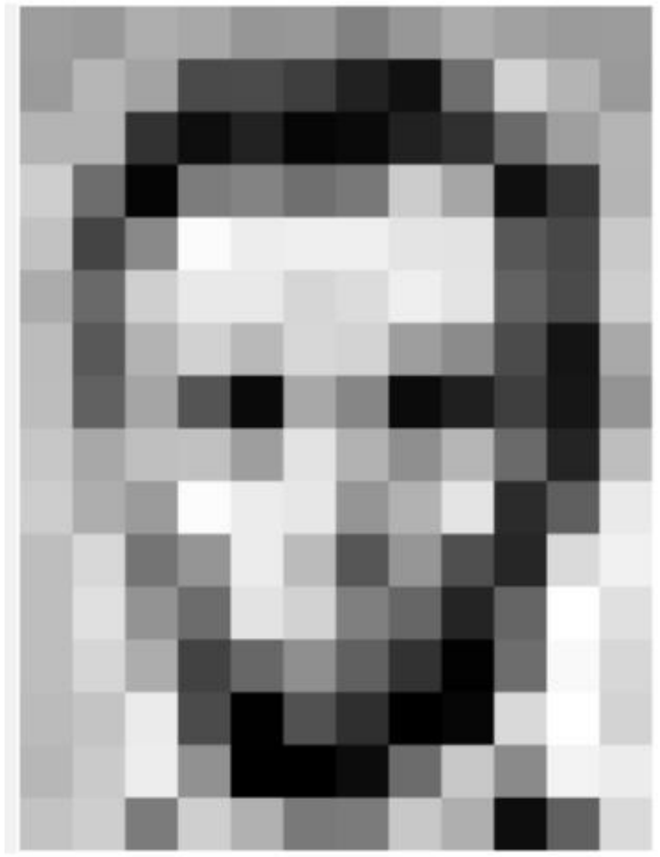
- DSA in our daily life:
  - ✓ Facebook. Your friends on Facebook, friends of friends, mutual friends they all can be represented easily by **Graph**.
  - ✓ If you need to keep a deck of cards and arrange it properly, You will throw it randomly or you will arrange the cards one over another and from a proper deck. You can use **Stack** here to make a proper arrangement of cards one over another.
  - ✓ If you need to search a word in the dictionary? Do you go page by page or you open some page and if the word is not found you open a page prior/later to one opened depending upon the order of word to the current page (**Binary Search**).

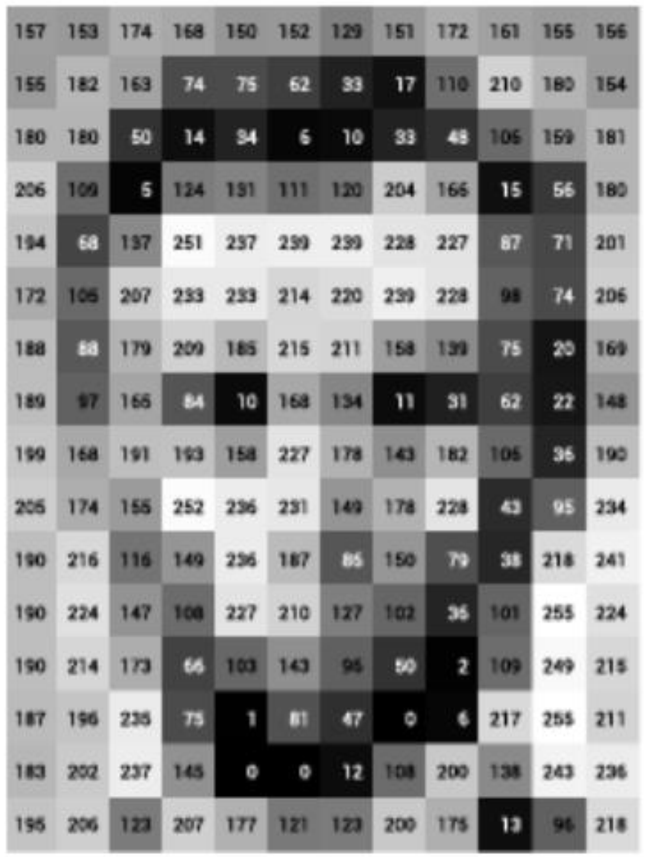
Why need DSA?

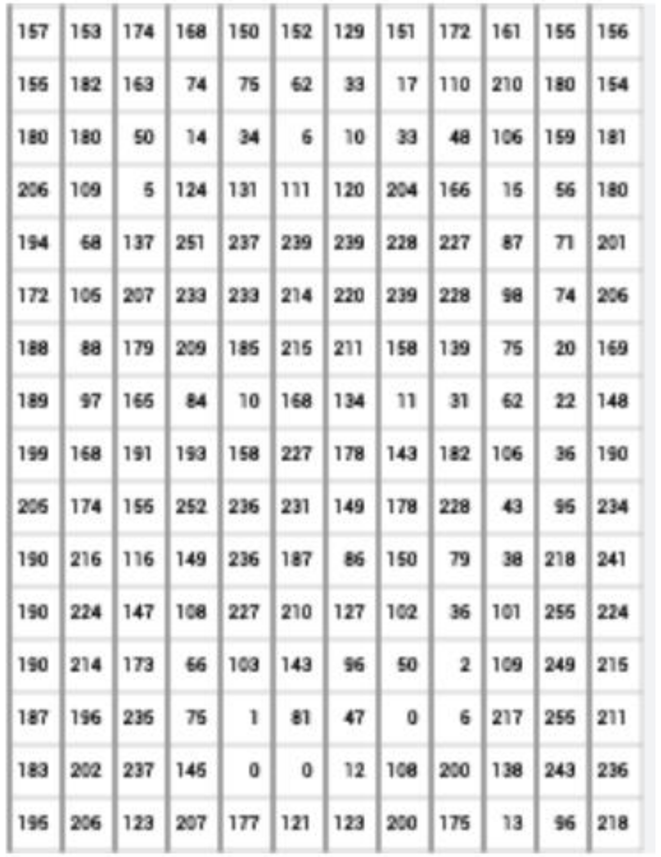
- A Good programmer needs the right tools (algorithm and data structure) to make the software run properly and efficiently (The main idea is to reduce the space and time complexities of different tasks.).
- If you know the characteristics of one data structure in contrast to another you will be able to make the right decision in choosing the right data structure to solve a problem.

Applications of Data structures

- Arrays: Image Processing
  - ❖ Images are represented as 2D arrays of pixels, where each pixel has a specific value

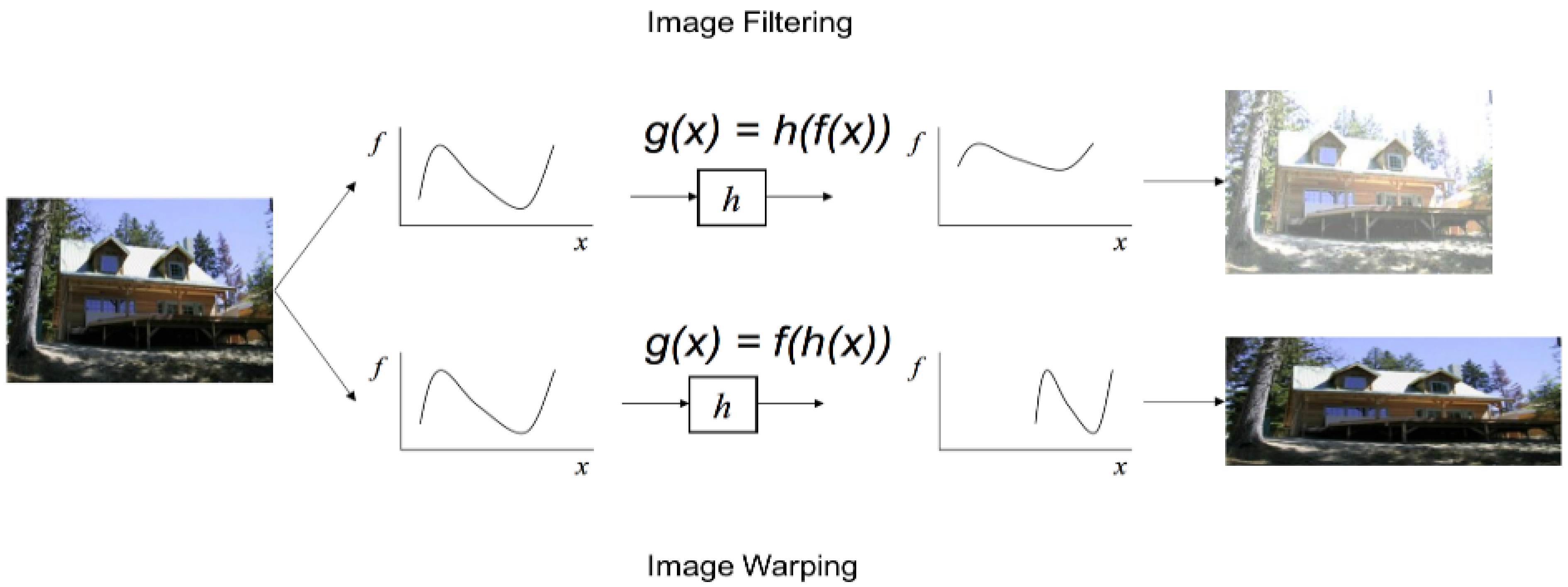






157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

- Two main types of image processing:
  - Image filtering: changes the range (i.e. the pixel values) of an image, so the colors of the image are altered without changing the pixel positions
  - Image warping: changes the domain (i.e. the pixel positions) of an image, where points are mapped to other points without changing the colors.



- NLP

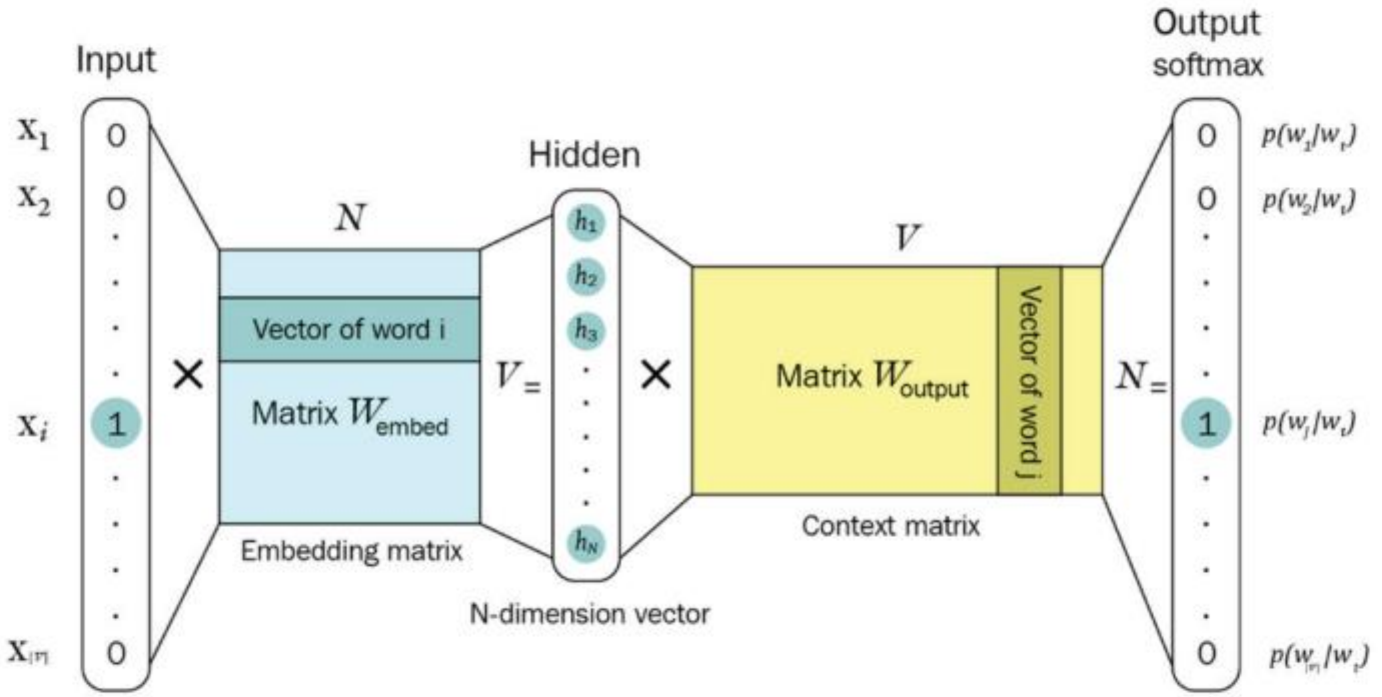
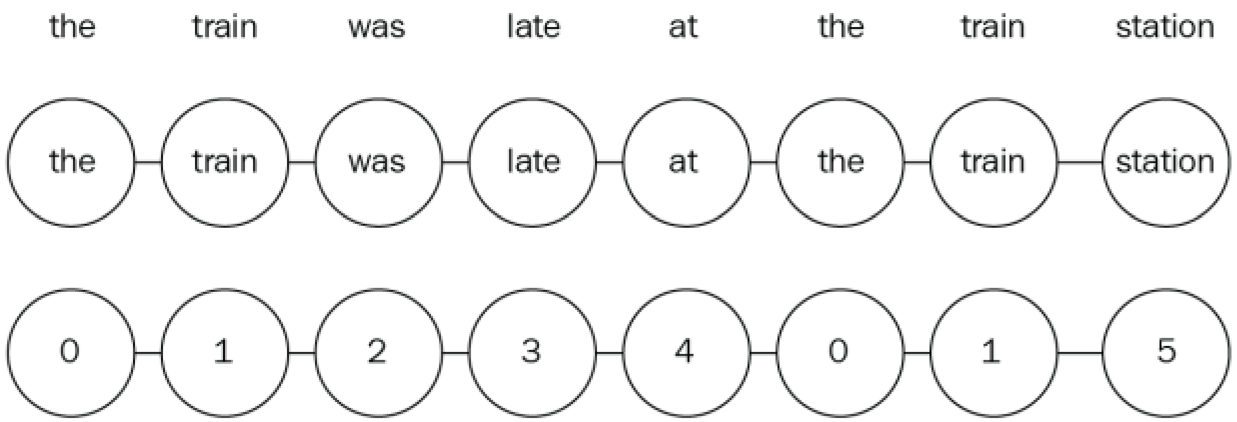


Figure 3.3 – The Word2Vec architecture

- Graph


the
train
was
late
at
the
train
station
the
train
was
late
at
the
train
station
0
1
2
3
4
0
1
5

Course contents

- **Fundamentals of complexity analysis:**
  - The running time of a program
  - Worst-case and expected time complexity.
- **Data Structures:**
  - ADT
  - List, Stack, Queue, Trees, heap.
  - Graph.
- **Algorithms:**
  - Searching
  - Sorting
  - And some graph algorithms: DFS, BFS, Prim’s, Dijkstra’s.
  - Divide and conquer, Dynamic programming technique.

Algorithm example

Maximum subsequence sum problem

- **Maximum subsequence sum problem:**

  The problem is presented in textbook p.33

  Given (possibly negative) integers $A_1, A_2, A_3, \ldots A_N$.

  The subsequence sum is:

$$S_{ij} = \sum_{k=i}^{j} A_k \ (1 \le i \le j \le N)$$

(For convenience, the maximum subsequence sum is 0 if all the integers are negative.)

Find the maximum value of S_{ij}?

- **Example:**

  A={-2, 11, -4, 13, -5, -2}

  S_{ij} can be: {-2+11}, {-2+11-4}, {-4+13-5} …

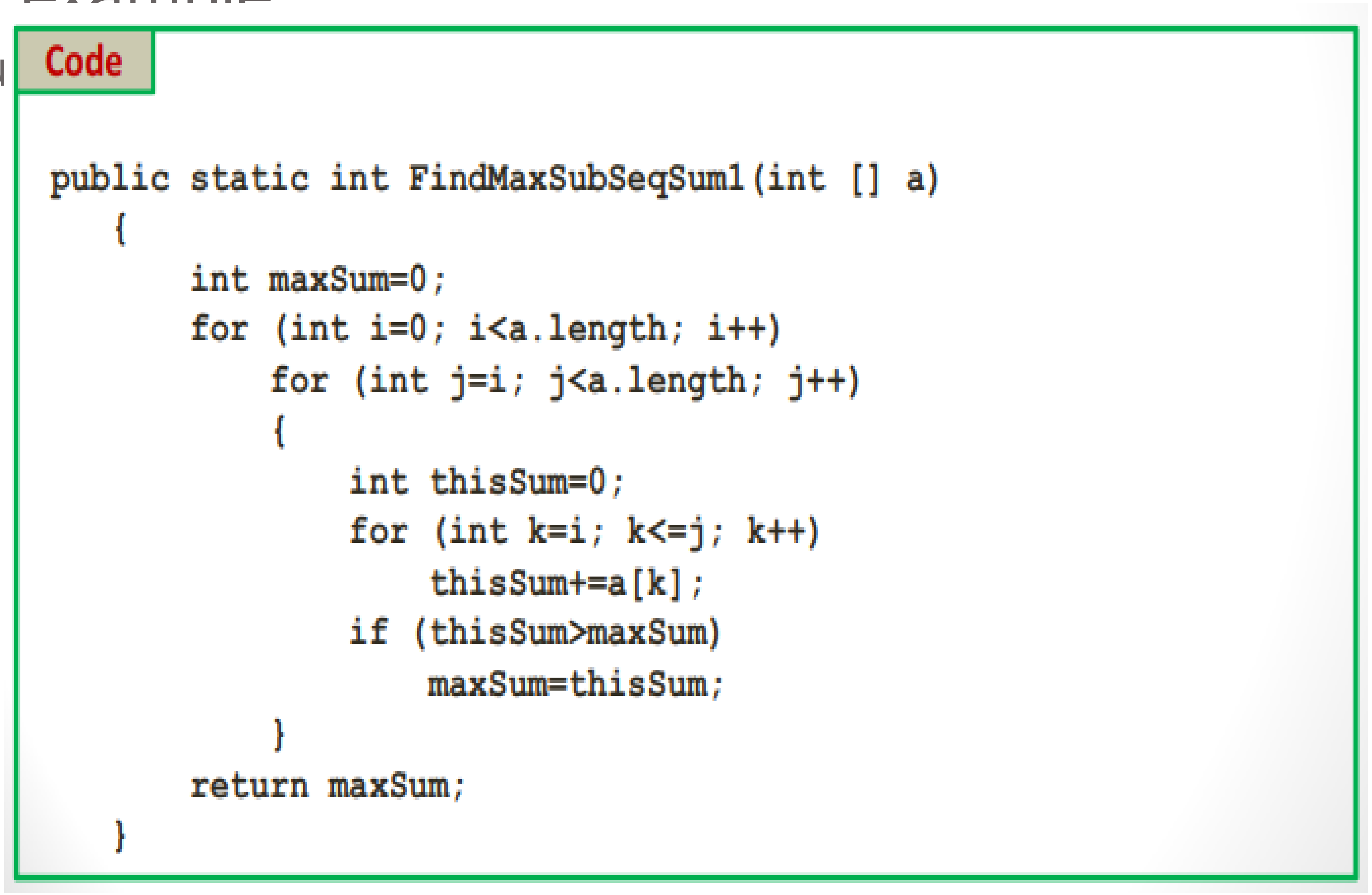
  Maximum value of S_{ij} is: {11 – 4 + 13}=20

Algorithm example

Maximum su

There are 4 solutions in textbook (p.39)

- 1st solution:



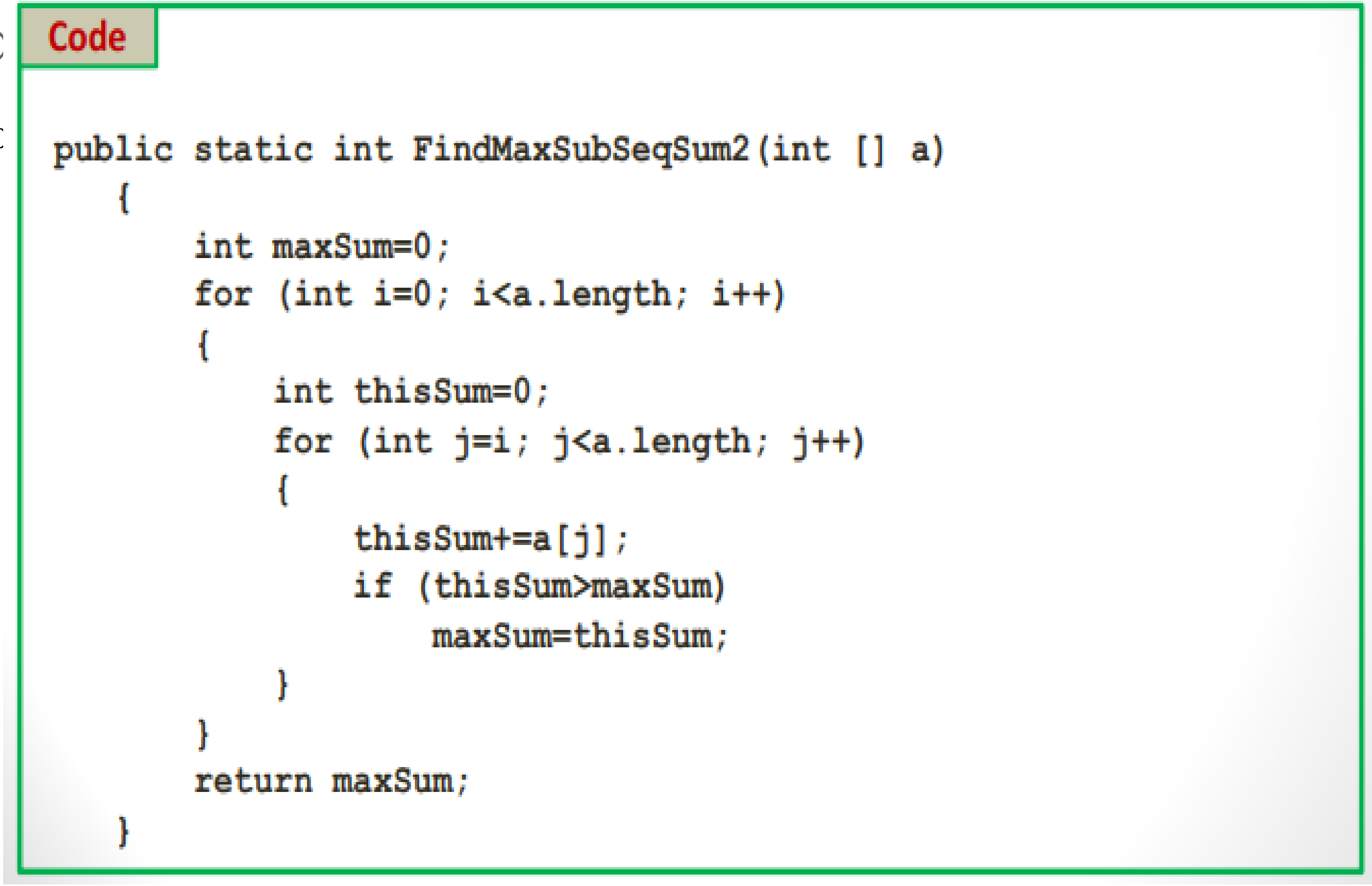
Code

```
public static int FindMaxSubSeqSum1(int [] a)
    {
         int maxSum=0;
         for (int i=0; i<a.length; i++)
              for (int j=i; j<a.length; j++)
              {
                   int thisSum=0;
                   for (int k=i; k<=j; k++)
                        thisSum+=a[k];
                   if (thisSum>maxSum)
                        maxSum=thisSum;
              }
         return maxSum;
    }
```

Algorithm example

Maximum subsec

- 2nd solution: Imprc



Code

```
public static int FindMaxSubSeqSum2(int [] a)
    {
        int maxSum=0;
        for (int i=0; i<a.length; i++)
        {
            int thisSum=0;
            for (int j=i; j<a.length; j++)
            {
                thisSum+=a[j];
                if (thisSum>maxSum)
                    maxSum=thisSum;
            }
        }
        return maxSum;
    }
```

Algorithm example

Maximum subsequence sum problem

- 3rd solution: Applied divide and conquer strategy

```
private static int maxSumRec( int [ ] a, int left, int right )
{
    if( left == right )  // Base case
        if( a[ left ] > 0 )
            return a[ left ];
        else
            return 0;

    int center = ( left + right ) / 2;
    int maxLeftSum  = maxSumRec( a, left, center );
    int maxRightSum = maxSumRec( a, center + 1, right );

    int maxLeftBorderSum = 0, leftBorderSum = 0;
    for( int i = center; i >= left; i-- )
    {
        leftBorderSum += a[ i ];
        if( leftBorderSum > maxLeftBorderSum )
            maxLeftBorderSum = leftBorderSum;
    }
```

Algorithm example

Maximum subsequence sum problem

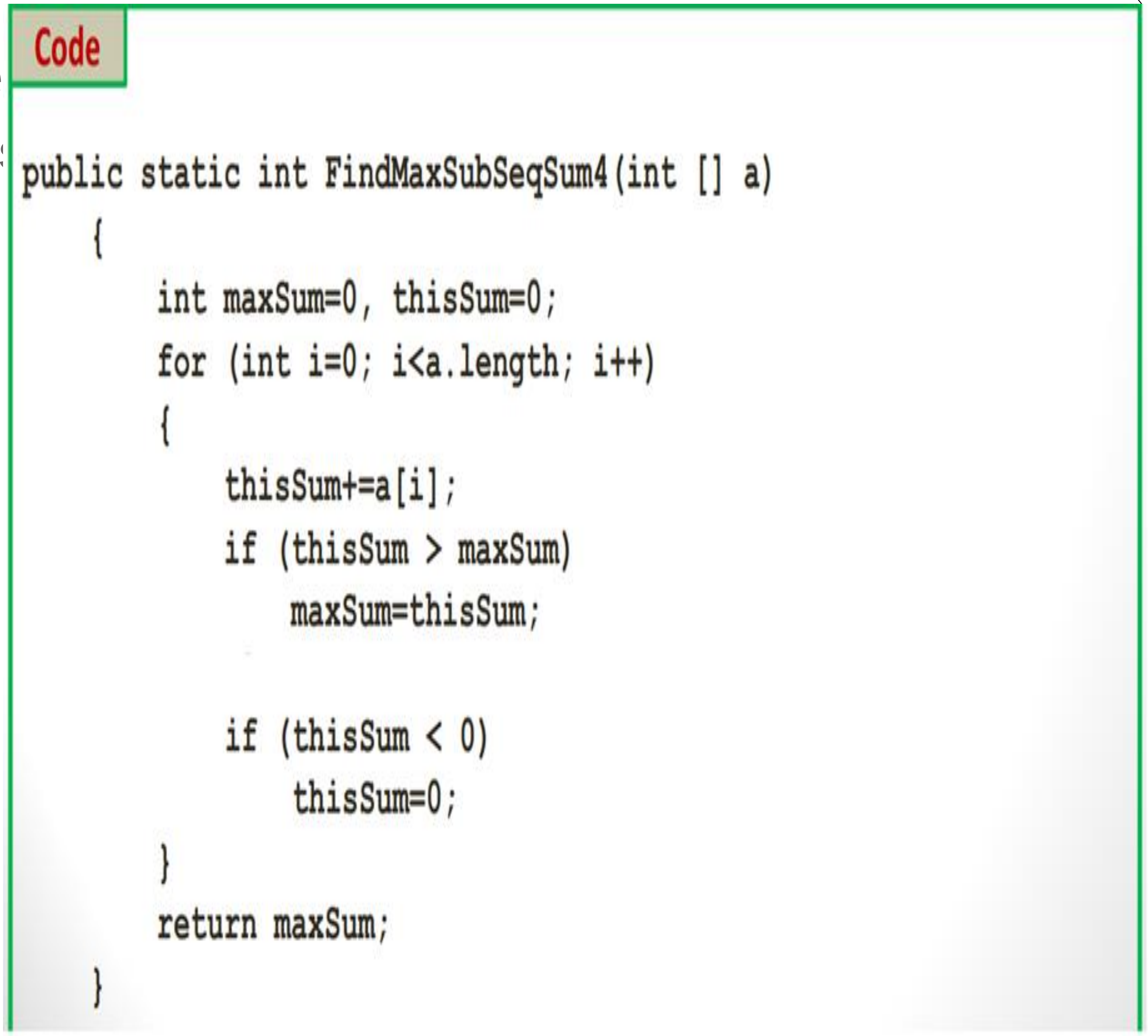
- 3rd solution:

```
    int maxRightBorderSum = 0, rightBorderSum = 0;
    for( int i = center + 1; i <= right; i++ )
    {
        rightBorderSum += a[ i ];
        if( rightBorderSum > maxRightBorderSum )
            maxRightBorderSum = rightBorderSum;
    }

    return max3( maxLeftSum, maxRightSum,
                 maxLeftBorderSum + maxRightBorderSum );
}

/**
 * Driver for divide-and-conquer maximum contiguous
 * subsequence sum algorithm.
 */
public static int maxSubSum3( int [ ] a )
{
    return maxSumRec( a, 0, a.length - 1 );
}
```

Algorithm example

Maximum subsequence

- 4th solution: Linear – time version
- Idea: The sub-array with negative sum is discarded (by assigning *thisSum* = 0)


Code

```
public static int FindMaxSubSeqSum4(int [] a)
    {
        int maxSum=0, thisSum=0;
        for (int i=0; i<a.length; i++)
        {
            thisSum+=a[i];
            if (thisSum > maxSum)
                maxSum=thisSum;

            if (thisSum < 0)
                thisSum=0;
        }
        return maxSum;
    }
```

- int maxSum = 0 → int maxSum = INT_MIN

Tutorial and next topic

- Prepare for the tutorial:
  - Download and install Eclipse
  - Practice Examples and Exercises in Week 1_Tutorial_Instruction
  - Read textbook section 2.4.3 (solution for the *Maximum subsequence sum* problem)
- Prepare for the next topic:
  - Read textbook chapter 2