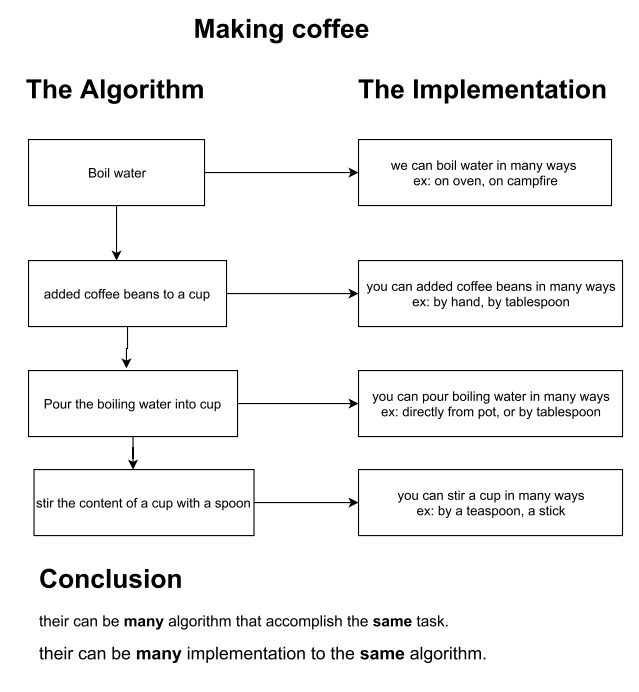
Algorithms

# Bubble sort

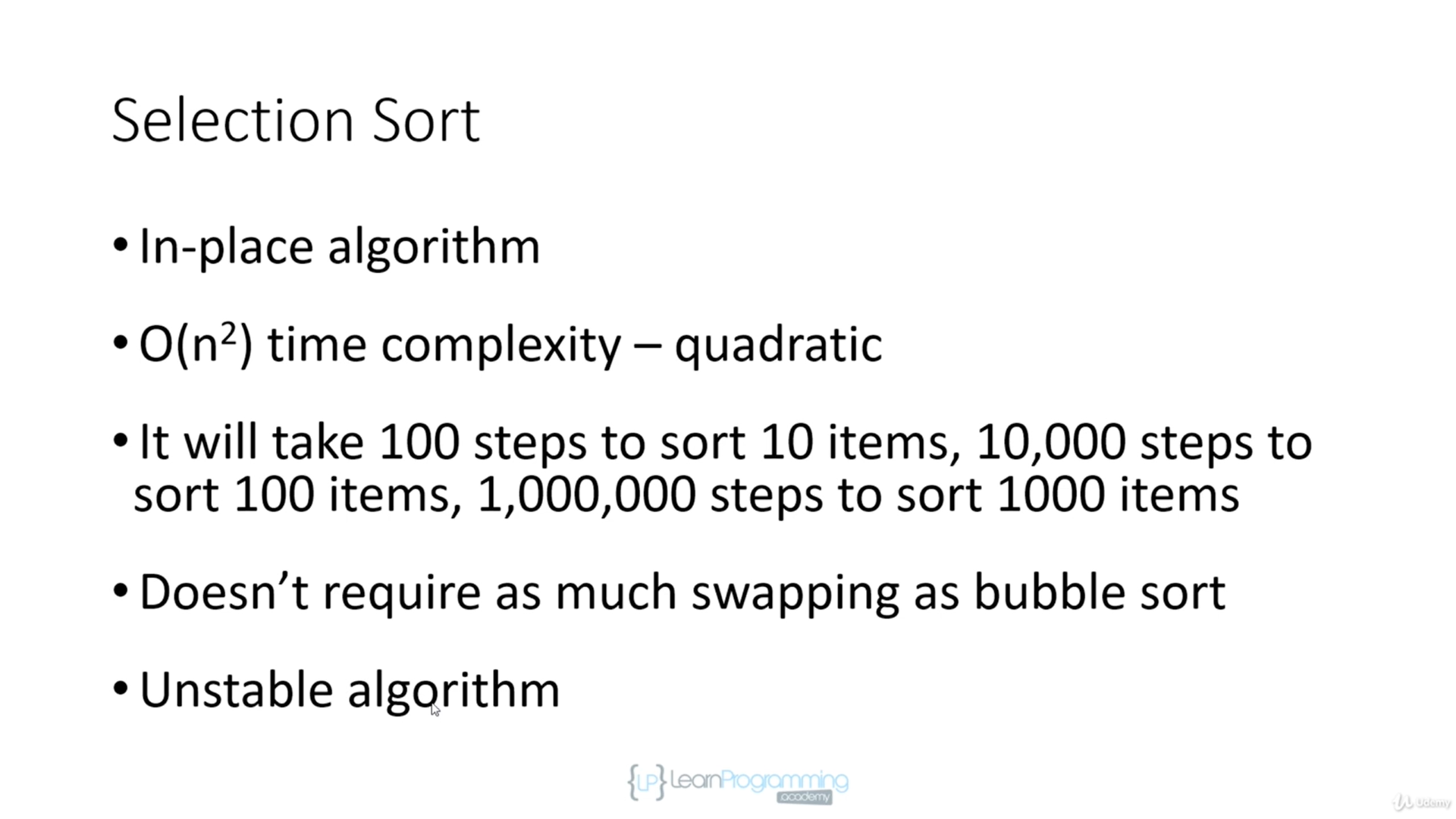
* In place algorithm
* – time complexity – quadratic.



package com.mughees;  
  
import java.util.Arrays;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
 int[] intArray = {20 , 35, -15, 7, 55, 1, -22};  
  
 for(int lastUnsortedIndex = intArray.length - 1; lastUnsortedIndex > 0;  
 lastUnsortedIndex--) {  
 for (int i = 0; i < lastUnsortedIndex; i++) {  
 if (intArray[i] > intArray[i+1]) {  
 *swap*(intArray, i, i+1);  
 }  
 }  
 System.*out*.println(Arrays.*toString*(intArray));  
 }  
 }  
  
 private static void swap(int[] array, int i, int j) {  
  
 if (i == j) {  
 return;  
 }  
 int temp = array[i];  
 array[i] = array[j];  
 array[j] = temp;  
 }  
}

# Selection sort

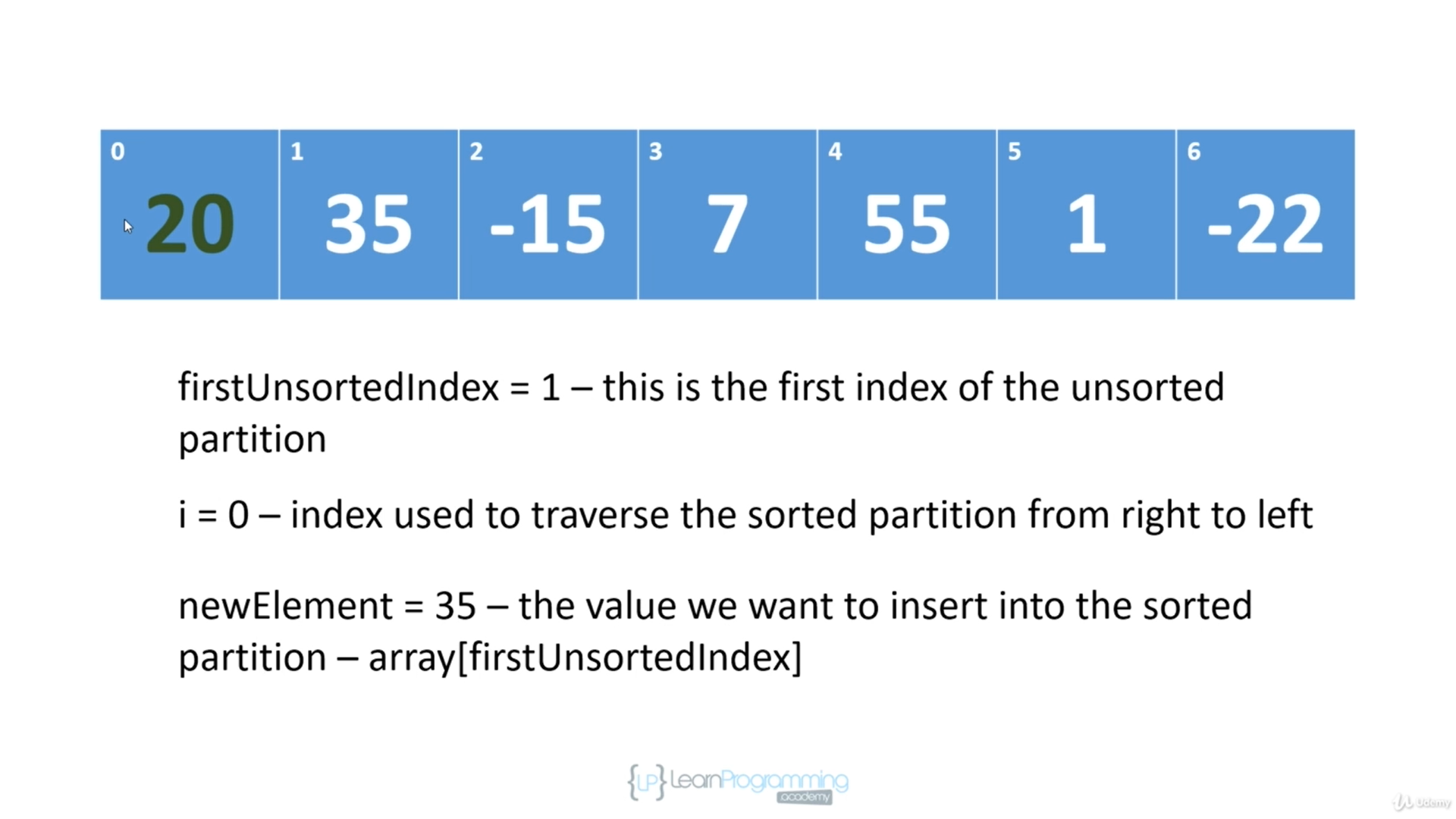
* Each traversal, selecting the largest number and putting it in place.

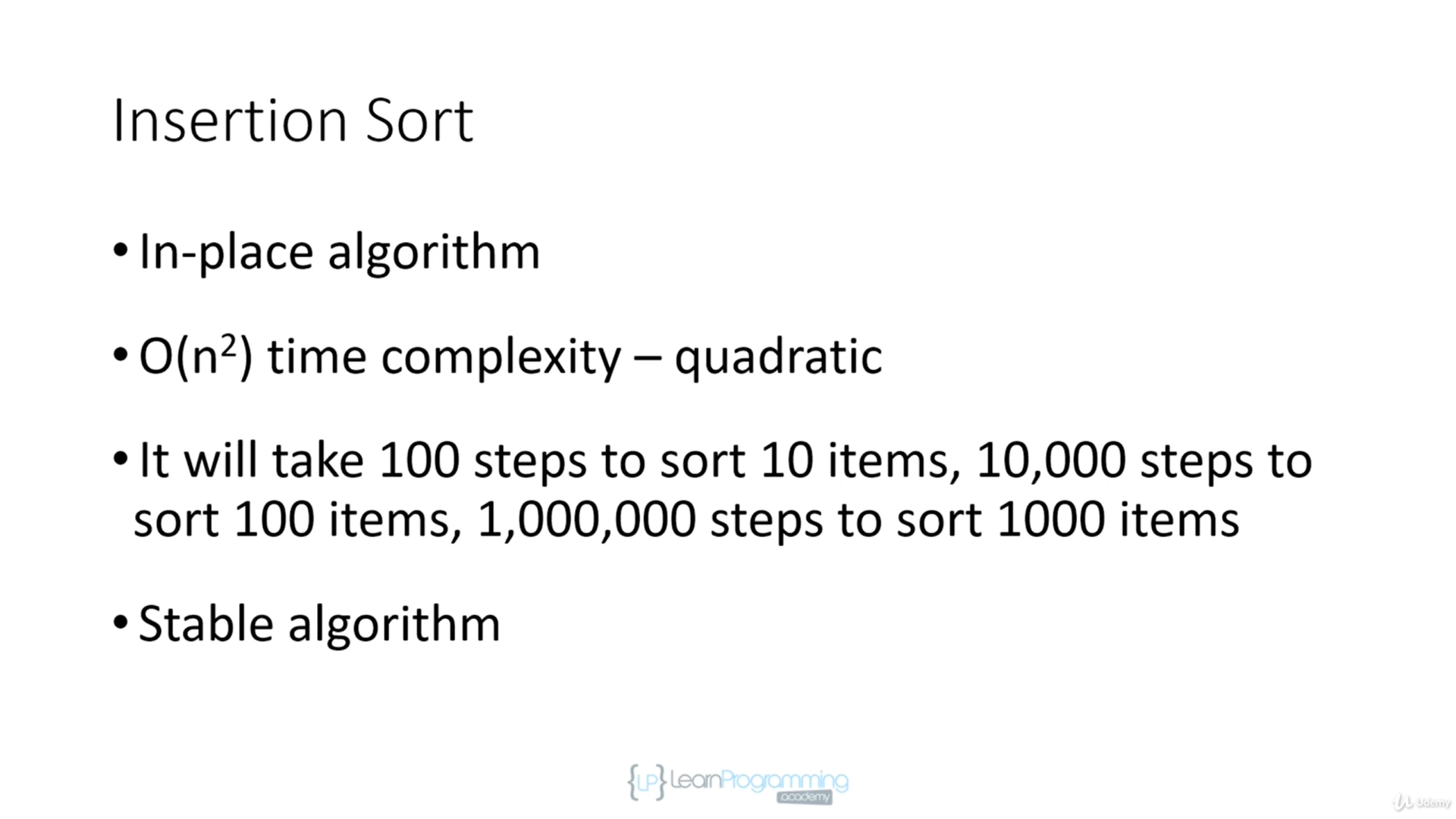


package com.mughees;  
  
import java.util.Arrays;  
  
public class Main {  
  
 public static void main(String[] args) {  
 int[] array = {20, 35, -15, 7, 55, 1, -22};  
  
 for (int lastUnsortedIndex = array.length - 1; lastUnsortedIndex > 0;  
 lastUnsortedIndex--) {  
  
 int largest = 0;  
  
 for (int i = 1; i <= lastUnsortedIndex; i++) {  
 if(array[i] > array[largest]) {  
 largest = i;  
 }  
 }  
 *swap*(array, largest, lastUnsortedIndex);  
 }  
 System.*out*.println(Arrays.*toString*(array));  
 }  
  
 private static void swap(int[] array, int i, int j) {  
 if (i == j) {  
 return;  
 }  
 int temp = array[i];  
 array[i] = array[j];  
 array[j] = temp;  
 }  
}



# Insertion Sort

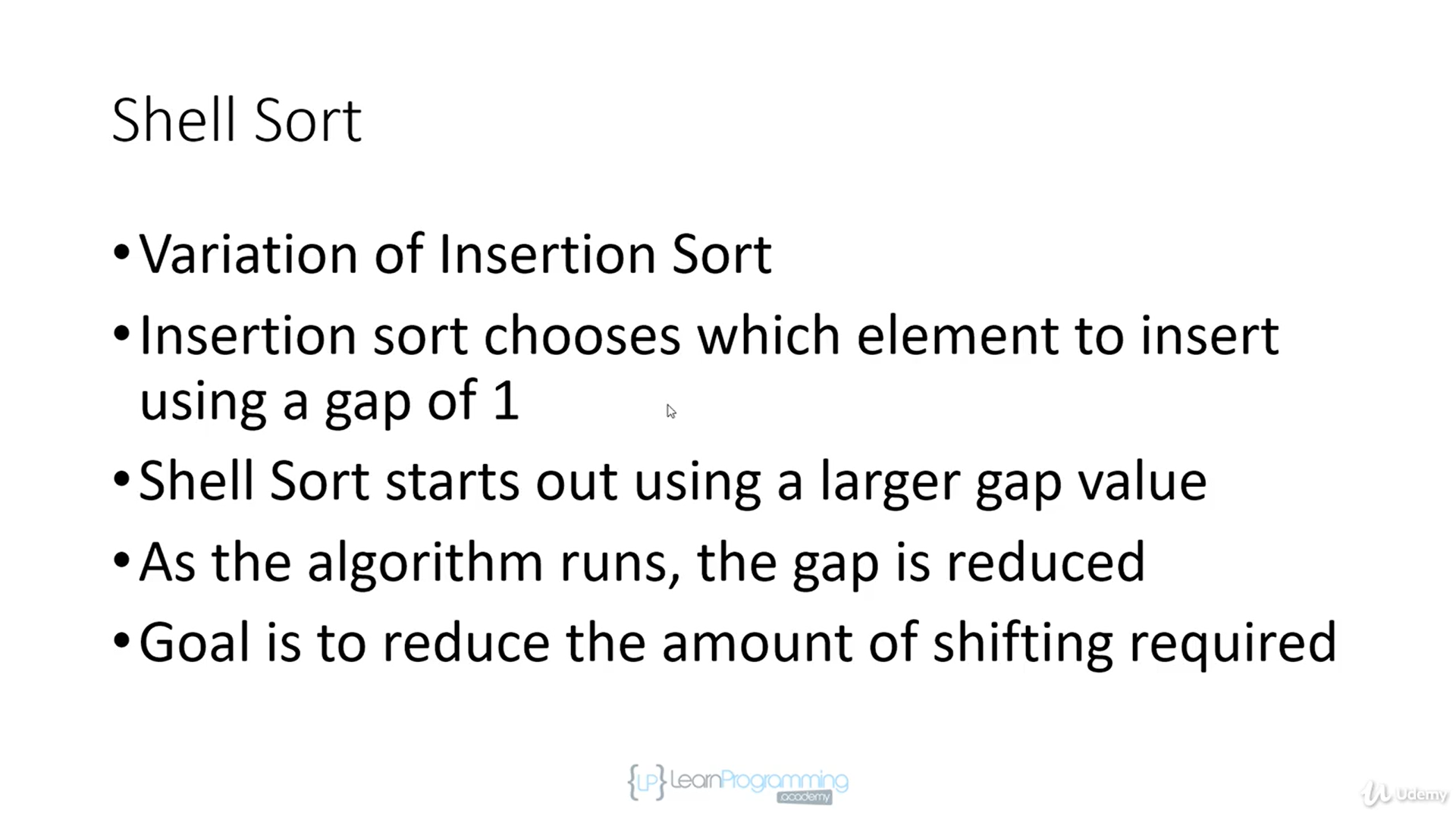


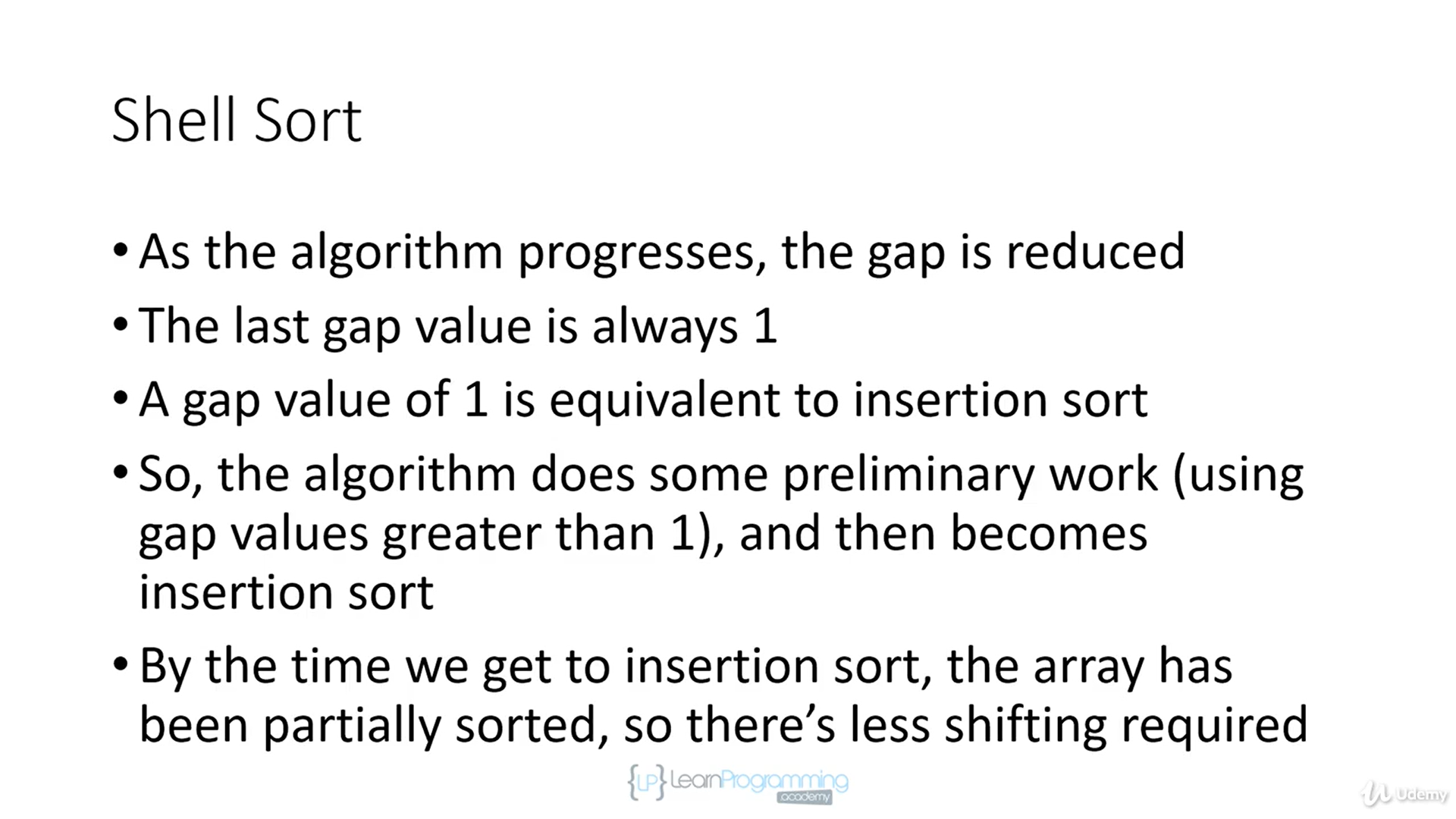


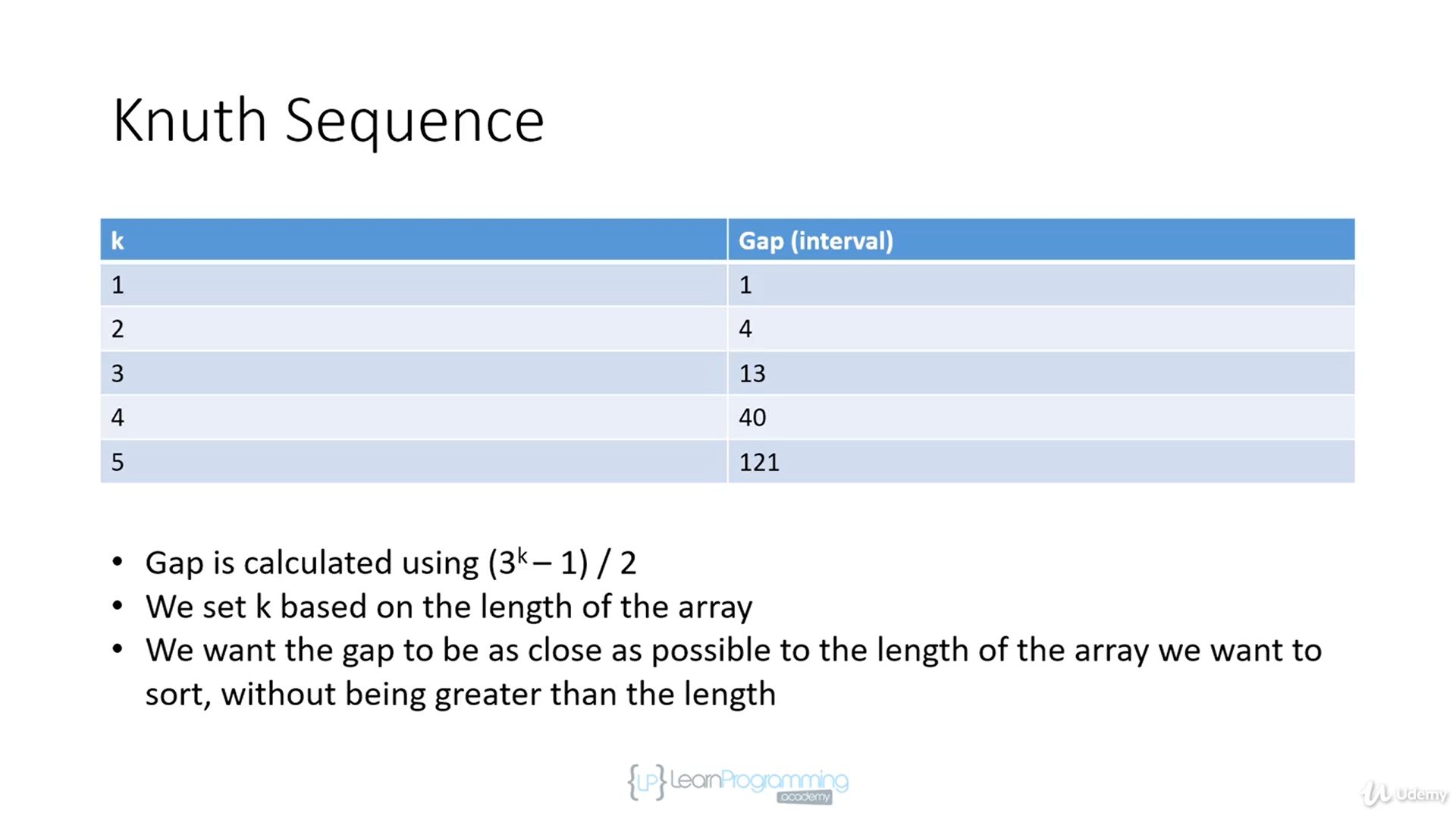
public class Main {  
  
 public static void main(String[] args) {  
 int[] array = {20, 35, -15, 7, 55, 1, -22};  
  
 for (int firstUnsortedIndex = 1; firstUnsortedIndex < array.length;  
 firstUnsortedIndex++) {  
 int newElement = array[firstUnsortedIndex];  
 int i;  
 for(i = firstUnsortedIndex; i > 0 && array[i -1] > newElement; i--){  
 array[i] = array[i - 1];  
 }  
 array[i] = newElement;  
 System.*out*.println(Arrays.*toString*(array));  
 }  
 }

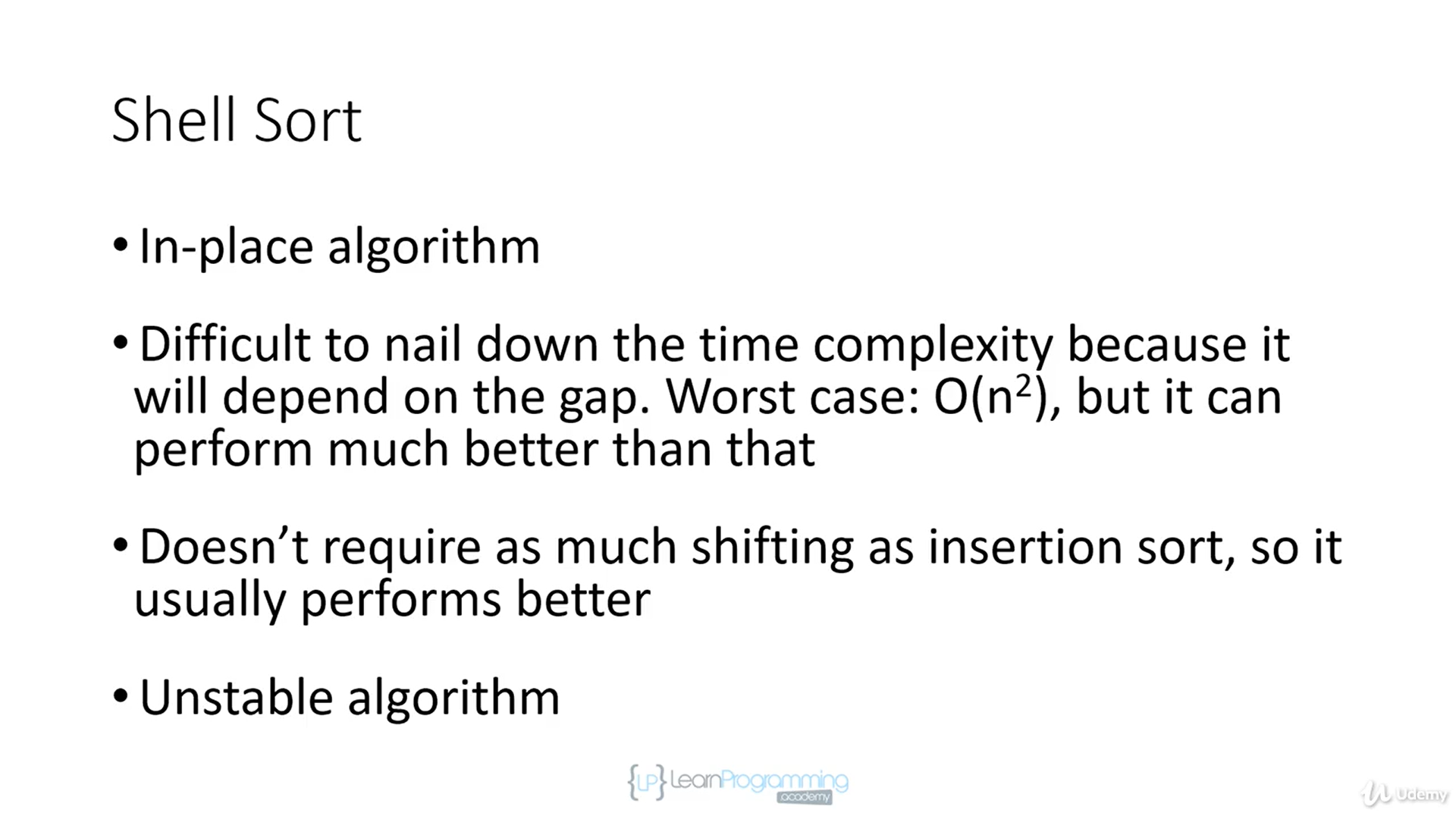
}

# Shell sort



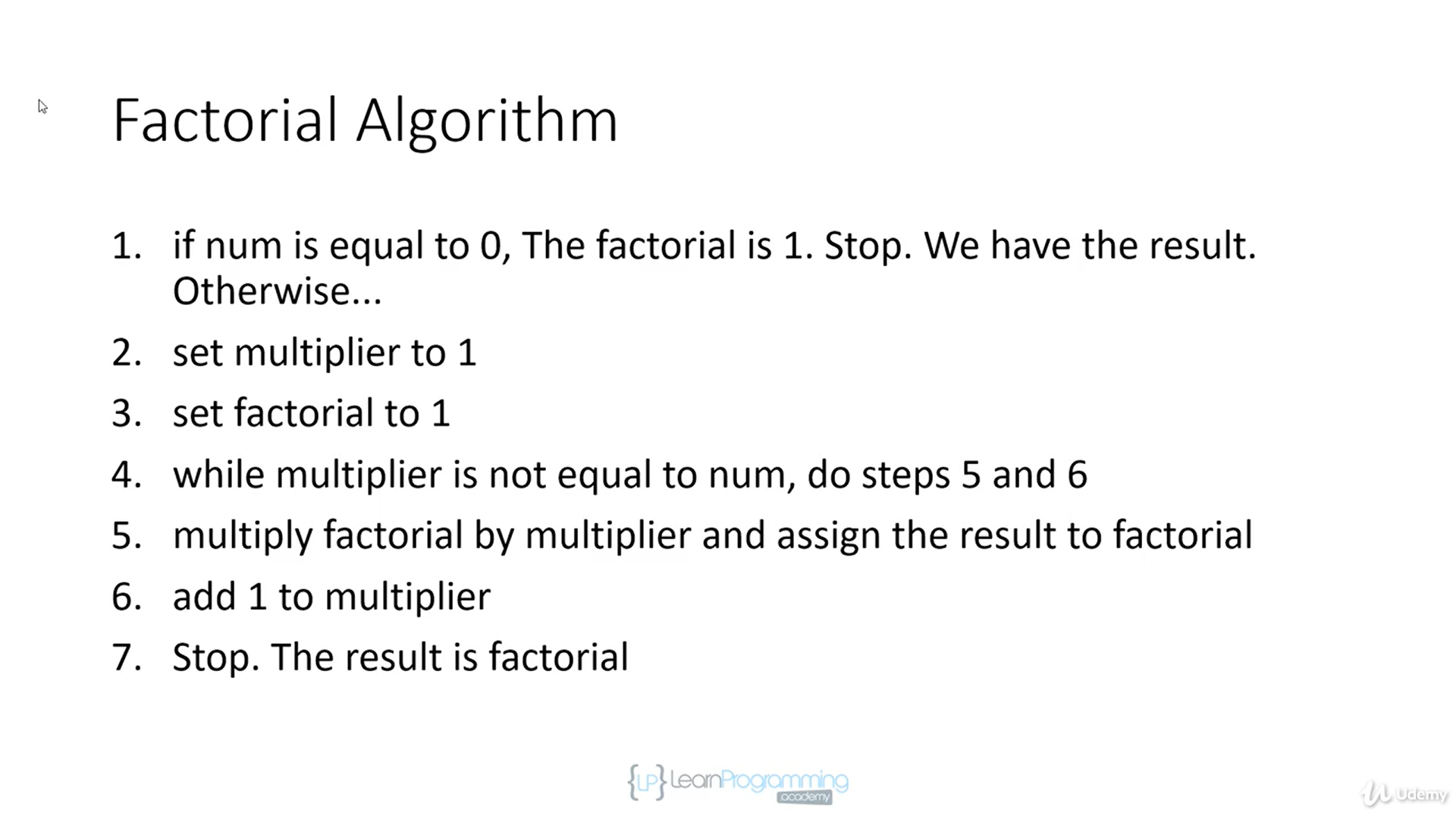






public static void main(String[] args) {  
  
 int[] array = {20, 35, -15, 7, 55, 1, -22};  
 for (int gap = array.length / 2; gap > 0; gap /= 2) {  
 for (int i = gap; i < array.length; i++) {  
 int newElement = array[i];  
 int j = i;  
 while (j >= gap && array[j - gap] > newElement) {  
 array[j] = array[j - gap];  
 j -= gap;  
 }  
 array[j] = newElement;  
 }  
 }  
 System.*out*.println(Arrays.*toString*(array));  
 }  
}

# Recursion



public class Main {  
  
 public static void main(String[] args) {  
 *iterativeFactorial*(10);  
 System.*out*.println(*recursiveFactorial*(4));  
 }  
  
 // n! = n \* (n-1)!  
  
 public static int recursiveFactorial (int num) {  
 if (num == 0) {  
 return 1;  
 }  
 return num \* *recursiveFactorial*(num - 1);  
 }  
   
 public static int iterativeFactorial (int num) {  
 if (num == 0) {  
 return 1;  
 }  
 int factorial = 1;  
 for (int i = 1; i <= num; i++) {  
 factorial \*= i;  
 System.*out*.println(factorial);  
 }  
 return factorial;  
 }  
}

* Iterative is faster and uses less memory because overheard involved in pushing method calls onto the call stack. Each call stack uses memory.
* But recursion is still used due to ease of understanding and elegance.

# Merge sort