CS 91 USACO

Bronze Division

Unit 5: Number Algorithms

LECTURE 24: PRIME NUMBER AND NEXT PALINDROME NUMBERS

DR. ERIC CHOU IEEE SENIOR MEMBER

Objectives

- Knowing number related algorithms such as
  - Prime Number checker
  - Palindrome String checker
  - Next Palindrome Number advancer
  - Next Permutation String advancer

Prime Numbers

SECTION 1

Review of Prime Number Algorithms

- Prime Number checker from 2 to n-1
- Prime Number checker from 2 to n/2
- Prime Number checker from 2 to sqrt(n)
- Prime Number checker for the previous prime number up to sqrt(n)

Palindrome Strings

SECTION 1

isPalindrome Checker

- A string that can be read from left to right and from right to level is called a palindrome string.
- To check a string is a palindrome string or not, you may just check half of the string’s characters and its corresponding character at its symmetric location. Then, you will know whether it is a palindrome string or not.

isPalindrome Checker

```java
public static boolean isPalindrome(int x){
    String xx = ""+x;
    int n = xx.length();
    for (int i=0; i<n/2; i++){
        if (xx.charAt(i) != xx.charAt(n-1-i)) return false;
     }
    return true;
 }
```

Next Palindrome Number Advancer

SECTION 1

Palindrome Number

- A palindrome number is a number that can be converted to a palindrome string which can be read from right to left or from left to right in the same way.
- All Single-digit number are palindrome numbers.
- All palindrome number can have a next palindrome number which is bigger than it and no other palindrome number between them.
- All 9 numbers will advance to a palindrome number with one more digit. The next palindrome number will be the all-9 number + 2.

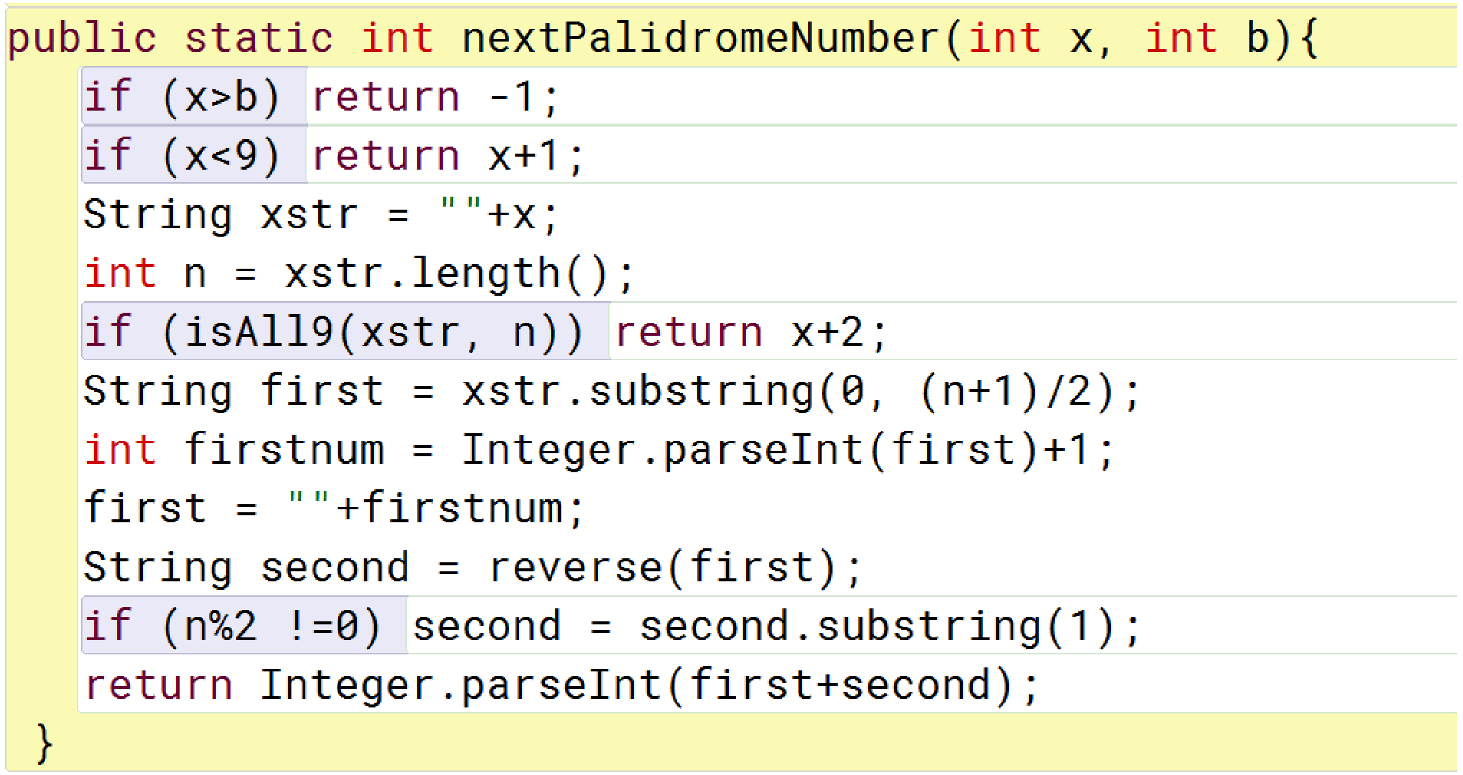
Palindrome Number

- 999’s next palindrome number will be 1001. 9999’s next palindrome number will be 10001.
- For all palindrome numbers of odd number of digits. The next palindrome number will be its first half and the reverse of the first half with the leading digit removed.
  - E.g. 121’s next palindrome number will be 131.
  - The first half is 12, the next number will be 13. Then, the next palindrome number will be 131

Palindrome Number

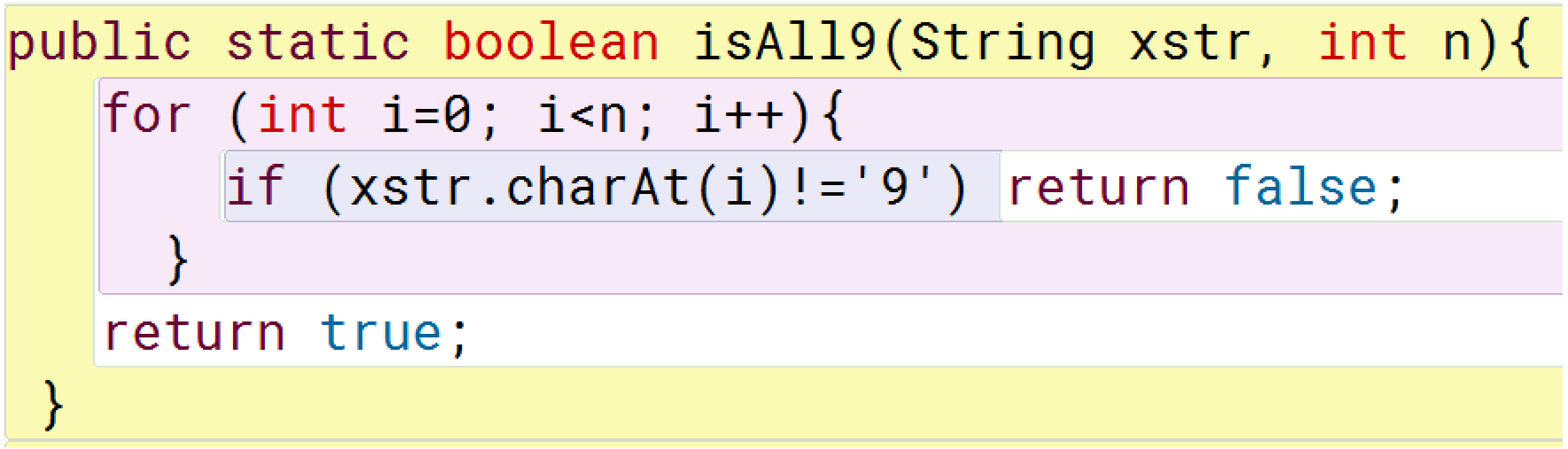
- For all palindrome numbers of even number of digits. The next palindrome number will be its first half and the reverse of the first half.
  - E.g.   1221’s next palindrome number will be 1331.
  - The first half is 12, the next number will be 13. Then, the next palindrome number will be 1331

Next Palindrome Number Generator

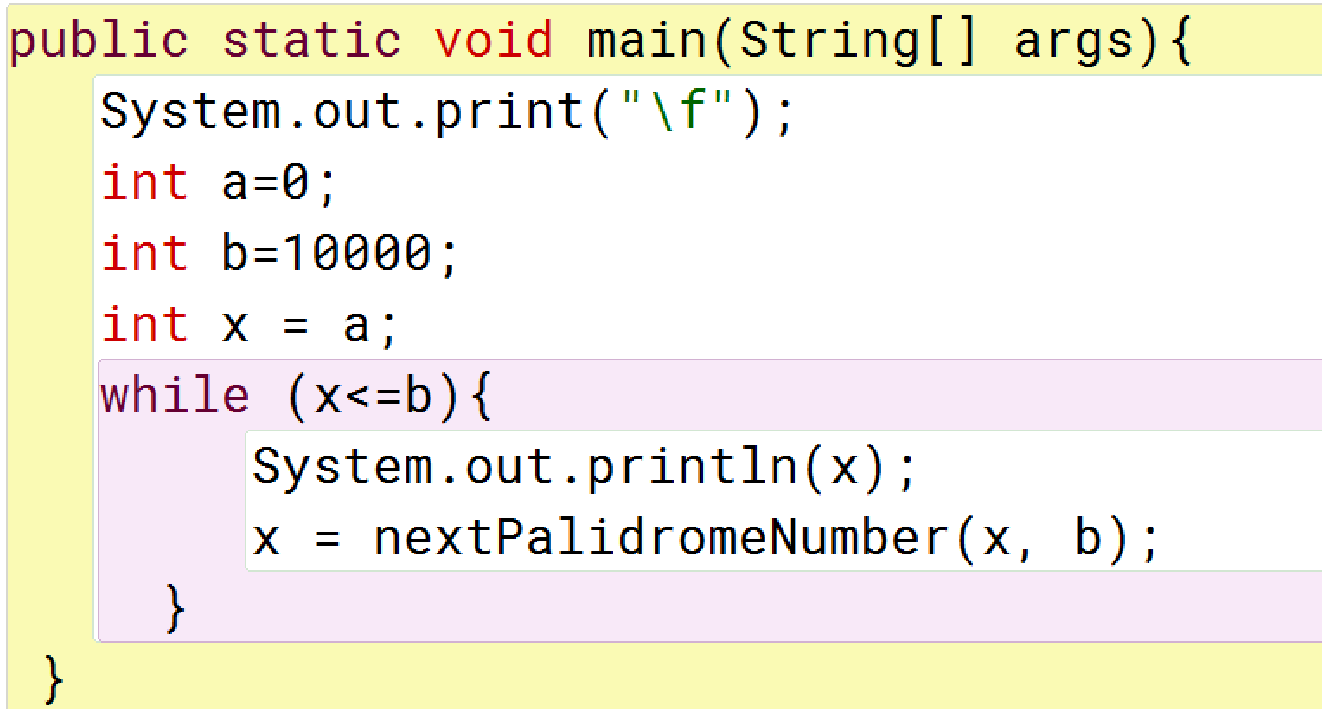


```
public static int nextPalidromeNumber(int x, int b){
    if (x>b) return -1;
    if (x<9) return x+1;
    String xstr = ""+x;
    int n = xstr.length();
    if (isAll9(xstr, n)) return x+2;
    String first = xstr.substring(0, (n+1)/2);
    int firstnum = Integer.parseInt(first)+1;
    first = ""+firstnum;
    String second = reverse(first);
    if (n%2 !=0) second = second.substring(1);
    return Integer.parseInt(first+second);
 }
```

isAll9: Check if a number with all 9-digits



```java
public static boolean isAll9(String xstr, int n){
    for (int i=0; i<n; i++){
        if (xstr.charAt(i)!='9') return false;
    }
    return true;
}
```



```
public static void main(String[] args){
    System.out.print("\f");
    int a=0;
    int b=10000;
    int x = a;
    while (x<=b){
         System.out.println(x);
         x = nextPalidromeNumber(x, b);
      }
 }
```

Creating Palindrome Number from 0 to 10000

Practice: Prime Palindromes (pprime)

SECTION 1

Problem Statement

- The number 151 is a prime palindrome because it is both a prime number and a palindrome (it is the same number when read forward as backward). Write a program that finds all prime palindromes in the range of two supplied numbers a and b (5 <= a < b <= 100,000,000); both a and b are considered to be within the range .

INPUT FORMAT (pprime.in):

- Line 1: Two integers, a and b

SAMPLE INPUT:

```
5 500
```

OUTPUT FORMAT (pprime.out):

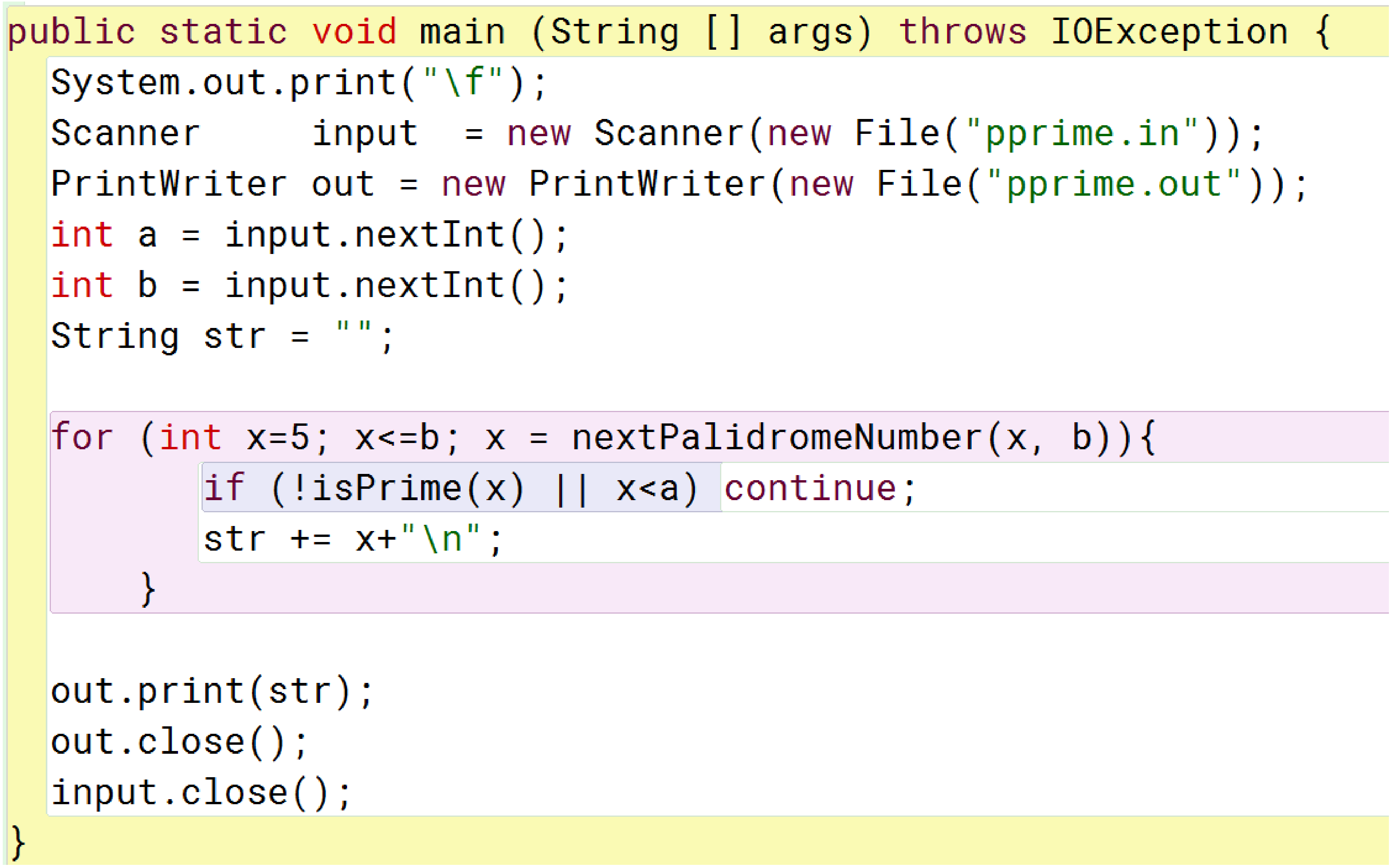
- The list of palindromic primes in numerical order, one per line.

SAMPLE OUTPUT:

```
5
7
11
101
131
151
181
191
313
353
373
383
```

Nature of the Problem

- Using regular isPrime() and isPalindrome() algorithms will be too slow. It can solve the problem but can never meet the time limits.
- Must use the next palindrome Advancer algorithm.
- A seed value 5 (or 2) for the initial palindrome number must be used



```java
public static void main (String [] args) throws IOException {
   System.out.print("\f");
   Scanner     input  = new Scanner(new File("pprime.in"));
   PrintWriter out = new PrintWriter(new File("pprime.out"));
   int a = input.nextInt();
   int b = input.nextInt();
   String str = "";

   for (int x=5; x<=b; x = nextPalidromeNumber(x, b)){
          if (!isPrime(x) || x<a) continue;
          str += x+"\n";
      }

   out.print(str);
   out.close();
   input.close();
}
```

Next Permutation Advancer

SECTION 1

Next Permutation Advancer

- Given an array or string, the task is to find the next lexicographically greater permutation of it in Java.

Brute-force Method

- **Naïve Approach:** Generate all the possible permutations of the given number/string, store them in a list and then traverse it to find the just greater permutation of the given number.
- **Time Complexity= O(n!*n) : n!** to generate all the permutations and an extra **n** to traverse and find the just greater permutation.

Brute-force Method

Space Complexity = O(n!) : to store all the permutations.

This approach is very naive and complex to implement. suppose we have array size as 100, which is not very big, but, it will generate 100! permutations. which is a huge number. Moreover, we will have to need 100! space to store all of them and then traverse it to find the next greater permutation.

Find the longest non-increasing suffix

- Suppose we have 13542 as our question and we have to find its next permutation, on observing it is clear that when we traverse from the last we see that the numbers are increasing up till and 5 and 3 is the first index which breaks the increasing order, hence, the first step:

Pivot and Reverse Method

1. Find the longest non-increasing suffix and find the pivot (3 i.e., index 1 is the pivot).
2. If the suffix is the whole array, then there is no higher order permutation for the data (In this case do as the question asks, either return -1 or the sorted array).

Pivot and Reverse Method

3. **Find the rightmost successor to the pivot :** to find the rightmost successor again start traversing from the back, the moment we encounter an element greater than the pivot we stop as it is the required element (here it is 4 index=3). This works because as we are traversing from the back, the elements are linearly increasing up till 3(this is the first time array starts decreasing) so, the moment we encounter an element greater than 3 it is indeed the just greater element or successor of 3 and all the elements to the left of 4 (till 3) are greater than 3 and all the elements to the right of 4 are smaller than 3.
4. Swap the successor and the pivot.

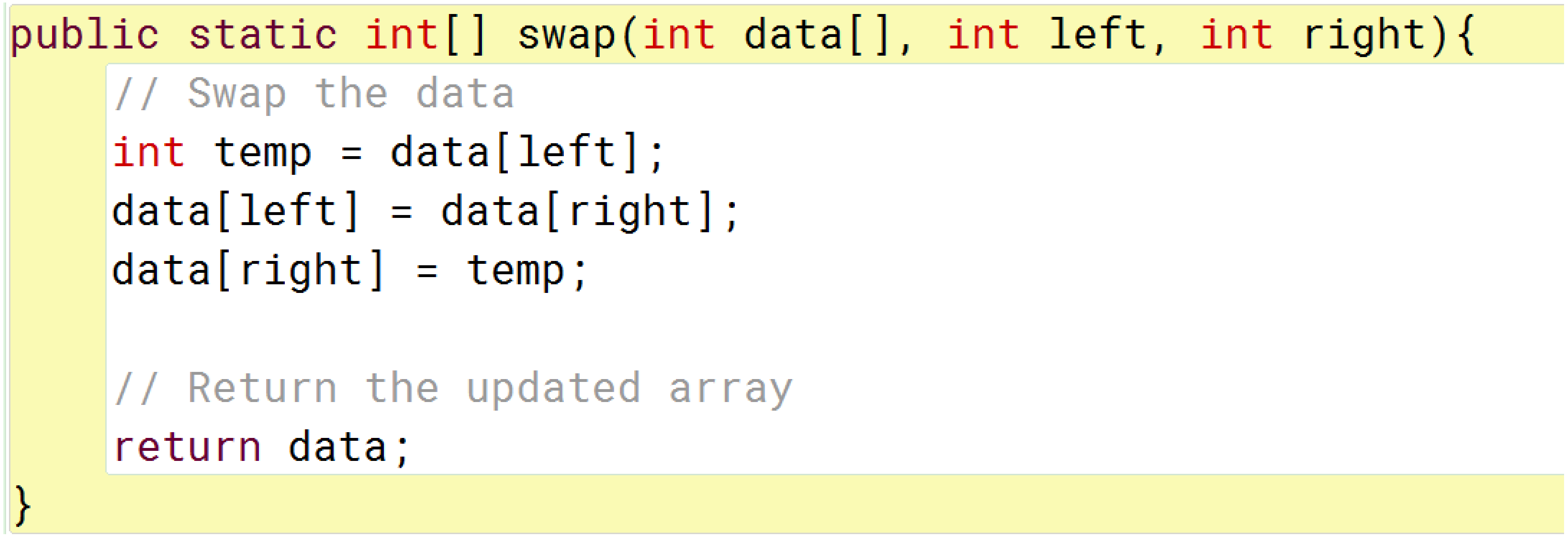
Pivot and Reverse Method

5. **Reverse the suffix:** once we swap the successor and pivot, a higher place value is modified and updated with a greater value, so it must be clear that we will obtain the next greater permutation only if the elements after the pivot are arranged in increasing order .

Time Complexity: O(n)

Auxiliary Space: O(1)

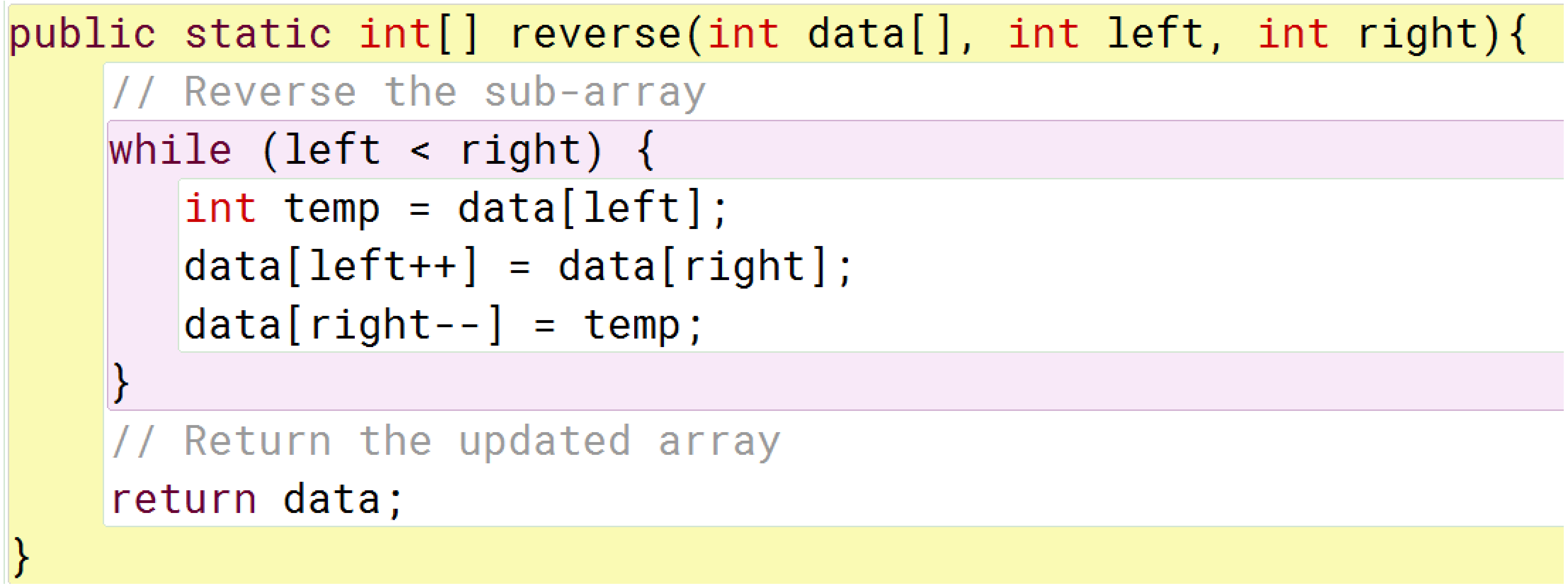
Swap left and right



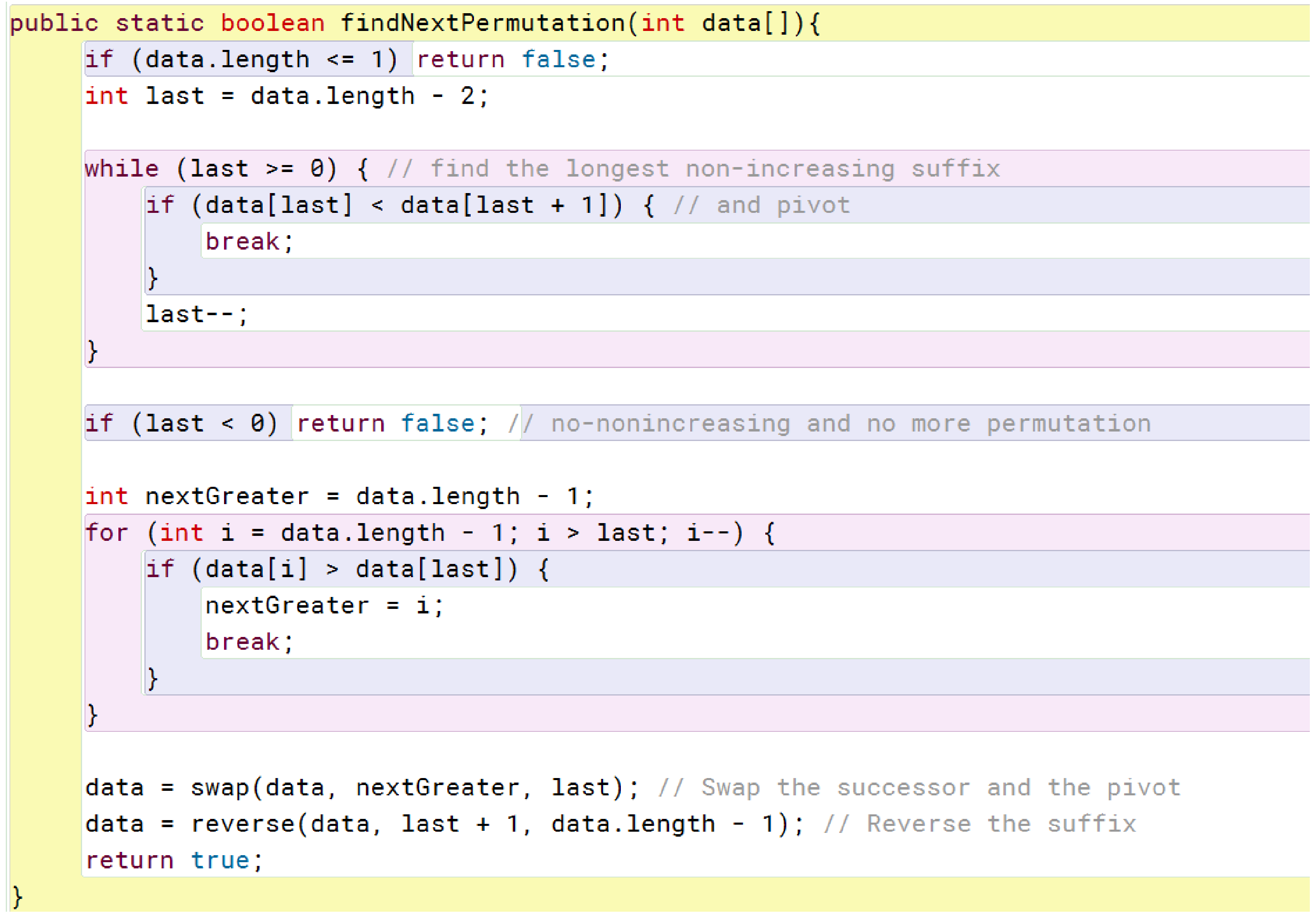
```java
public static int[] swap(int data[], int left, int right){
    // Swap the data
    int temp = data[left];
    data[left] = data[right];
    data[right] = temp;

    // Return the updated array
    return data;
}
```

Reverse of a sub-Array



```java
public static int[] reverse(int data[], int left, int right){
    // Reverse the sub-array
    while (left < right) {
        int temp = data[left];
        data[left++] = data[right];
        data[right--] = temp;
    }
    // Return the updated array
    return data;
}
```



```java
public static boolean findNextPermutation(int data[]){
    if (data.length <= 1) return false;
    int last = data.length - 2;

    while (last >= 0) { // find the longest non-increasing suffix
        if (data[last] < data[last + 1]) { // and pivot
            break;
        }
        last--;
    }

    if (last < 0) return false; // no-nonincreasing and no more permutation

    int nextGreater = data.length - 1;
    for (int i = data.length - 1; i > last; i--) {
        if (data[i] > data[last]) {
            nextGreater = i;
            break;
        }
    }

    data = swap(data, nextGreater, last); // Swap the successor and the pivot
    data = reverse(data, last + 1, data.length - 1); // Reverse the suffix
    return true;
}
```

Example

13542	**Last → 3** **To be promoted → 4** (nextGreater)
14532	**After swapping last and nextGreater**
14532	**Green sub-Array to be reversed**
14235	**After reversal**