Roman Hrabovskyi Student id 19385

Recursion

Recursion is an important concept in computer science that involves a function calling itself from within. Its ability to visualize a complex problem into basic steps has made it an essential tool in algorithm design.

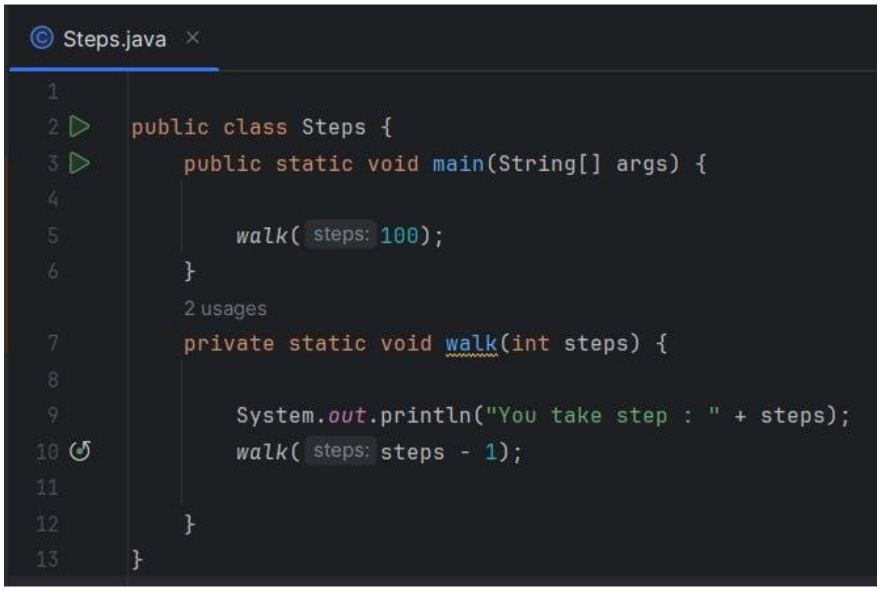
Recursive functions and algorithms have several advantages. They can reduce unnecessary calling of functions, make the code easier to write, reduce the length of code, and are very useful in solving data structure problems. Recursion is also useful in stacks evolutions and infix, prefix, postfix evaluations, among others.

However, there are also disadvantages to using recursion. Recursive functions are generally slower than non-recursive functions, and they may require a lot of memory space to hold intermediate results on the system stacks. Additionally, the code can be hard to analyze or understand, and it is not always more efficient in terms of space and time complexity.

Recursive functions and algorithms:

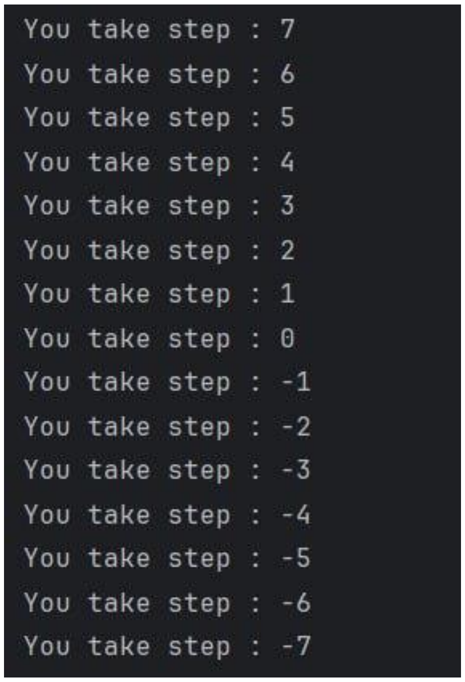
A common algorithm design tactic is to divide a problem into sub-problems of the same type as the original, solve those sub-problems, and combine the results. This is often referred to as the divide-and-conquer method; when combined with a lookup table that stores the results of previously solved sub-problems (to avoid solving them repeatedly and incurring extra computation time), it can be referred to as dynamic programming or memoization. To examples of lookup table we will back soon.

Ex (Procedural Paradigm) :

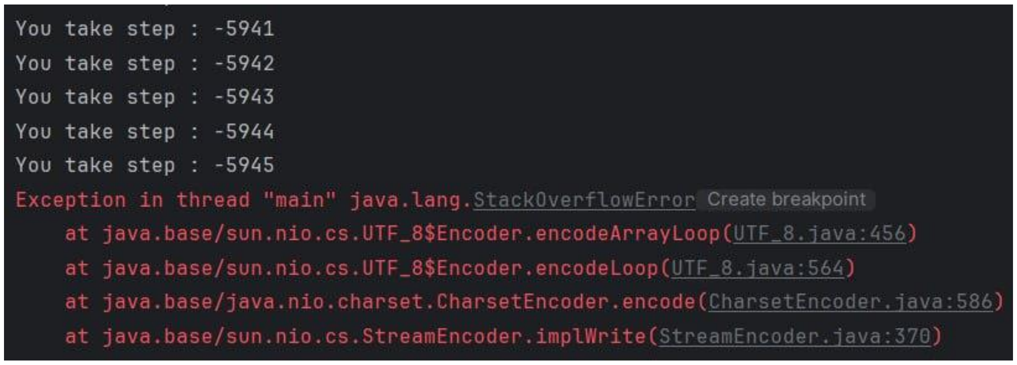


```java

public class Steps {
    public static void main(String[] args) {

        walk( steps: 100);
    }
    2 usages
    private static void walk(int steps) {

        System.out.println("You take step : " + steps);
        walk( steps: steps - 1);

    }
}
```

Here we see maybe the most common example of using recursive methods, but I also specially do not write the main thing. Let’s check the result :

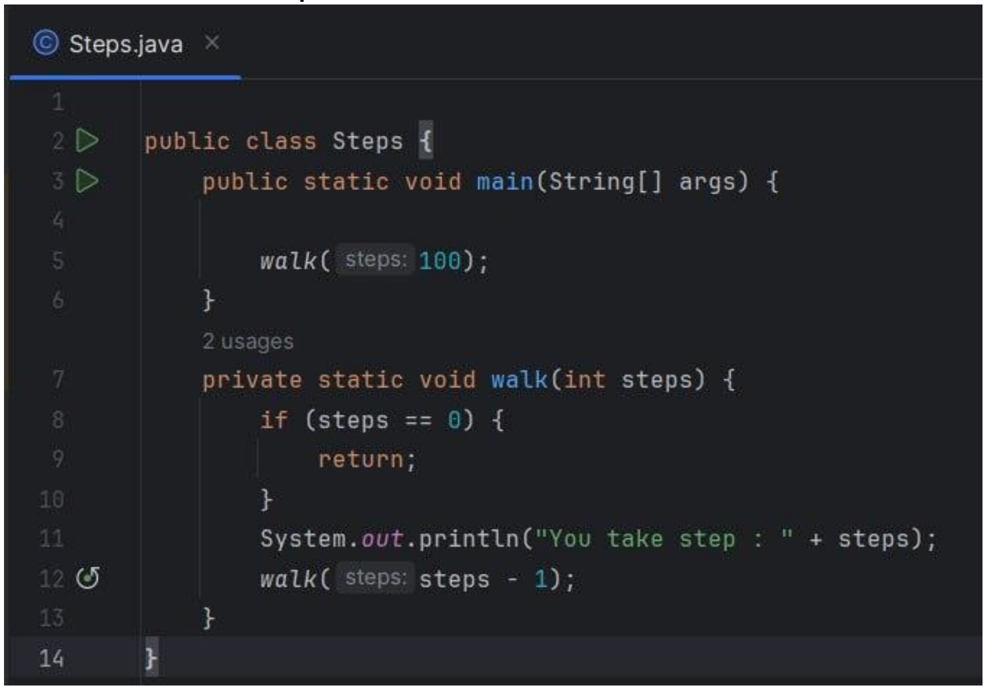


```
You take step : 7
You take step : 6
You take step : 5
You take step : 4
You take step : 3
You take step : 2
You take step : 1
You take step : 0
You take step : -1
You take step : -2
You take step : -3
You take step : -4
You take step : -5
You take step : -6
You take step : -7
```



```
You take step : -5941
You take step : -5942
You take step : -5943
You take step : -5944
You take step : -5945
Exception in thread "main" java.lang.StackOverflowError
    at java.base/sun.nio.cs.UTF_8$Encoder.encodeArrayLoop(UTF_8.java:456)
    at java.base/sun.nio.cs.UTF_8$Encoder.encodeLoop(UTF_8.java:564)
    at java.base/java.nio.charset.CharsetEncoder.encode(CharsetEncoder.java:586)
    at java.base/sun.nio.cs.StreamEncoder.implWrite(StreamEncoder.java:370)
```

The problem now named “Stack over flow Error “ it is because we do not have the Base case. For our ex. It is a condition that steps can’t be less than 0



```java

public class Steps {
    public static void main(String[] args) {

        walk( steps: 100);
    }
    private static void walk(int steps) {
        if (steps == 0) {
            return;
        }
        System.out.println("You take step : " + steps);
        walk( steps: steps - 1);
    }
}
```

Well now our code work correctly !

Let’s do something harder :

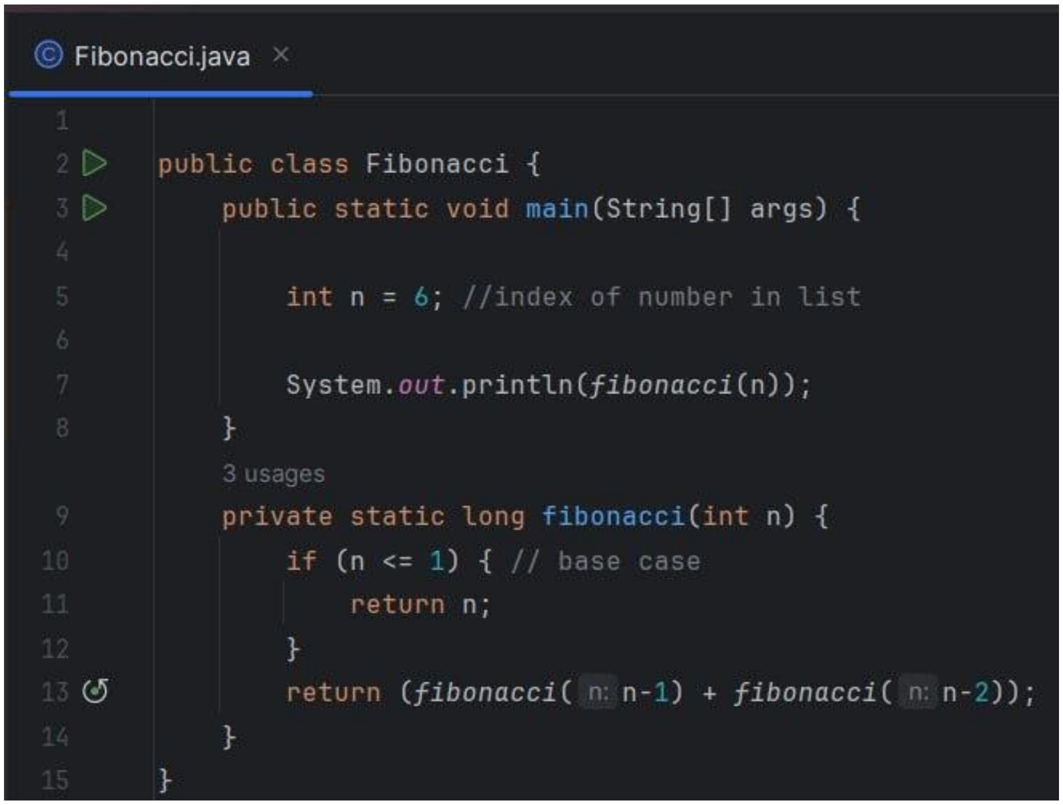
To illustrate the concept of recursion, let's consider two examples. The first example involves a simple recursive function that calculates the factorial of a number. The second example uses the Fibonacci sequence to demonstrate how recursion can become extremely slow when dealing with large numbers.

Ex 2 Fibonacci Series (OOP) :

0 1 1 2 3 5 8 13 21 34 55 . . .

f(n) = f(n – 1) + f(n – 2)

The code must find the number at its index in the list:



Fibonacci.java

```java
public class Fibonacci {
    public static void main(String[] args) {

        int n = 6; //index of number in list

        System.out.println(fibonacci(n));
    }
    3 usages
    private static long fibonacci(int n) {
        if (n <= 1) { // base case
            return n;
        }
        return (fibonacci( n: n-1) + fibonacci( n: n-2));
    }
}
```

Here is the code and it looks perfect and easy to read, But what happens if I decide to find 50 index in the list – spoiler it will take approximately 2 min and it’s the main disadvantage in the recursion method (much easier to read but extremely slow)

The Fibonacci sequence involves adding the previous two numbers in the sequence to obtain the next number. While recursion can be used to compute the Fibonacci sequence, it can become extremely slow for large values. However, using a lookup table to store intermediate results can significantly reduce the computation time.

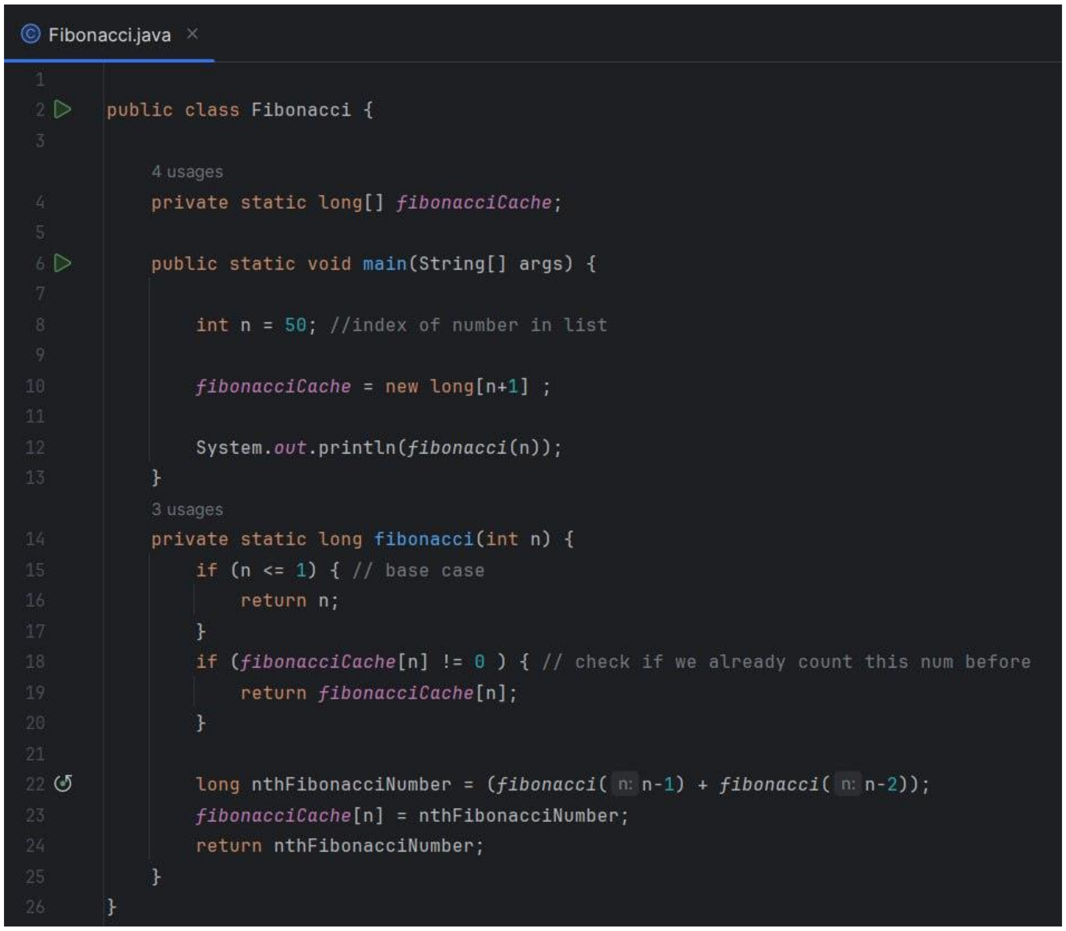
And now we should back to the lookup table. What is mean and how it use ?

Well let’s check when we repeat the same work ?

As was written here our formula $f(n) = f(n-1) + f(n-2)$:

$$\begin{tikzcd}
f(6) & = & f(5) \arrow[dll] & + & f(4) \\
f(5) & = & f(4) \arrow[dll] & + & f(3) \\
f(4) & = & f(3) & + & f(2)
\end{tikzcd}$$

As we see. We repeat the process of finding the same f(n). The idea to solve this problem is same in cache intermediate result:



```java
public class Fibonacci {

    4 usages
    private static long[] fibonacciCache;

    public static void main(String[] args) {

        int n = 50; //index of number in list

        fibonacciCache = new long[n+1] ;

        System.out.println(fibonacci(n));
    }
    3 usages
    private static long fibonacci(int n) {
        if (n <= 1) { // base case
            return n;
        }
        if (fibonacciCache[n] != 0 ) { // check if we already count this num before
            return fibonacciCache[n];
        }

        long nthFibonacciNumber = (fibonacci( n: n-1) + fibonacci( n: n-2));
        fibonacciCache[n] = nthFibonacciNumber;
        return nthFibonacciNumber;
    }
}
```

Now finding the 50 index take less than 1 second !!!

Ex 3 MergeSort (Functional Paradigm) :

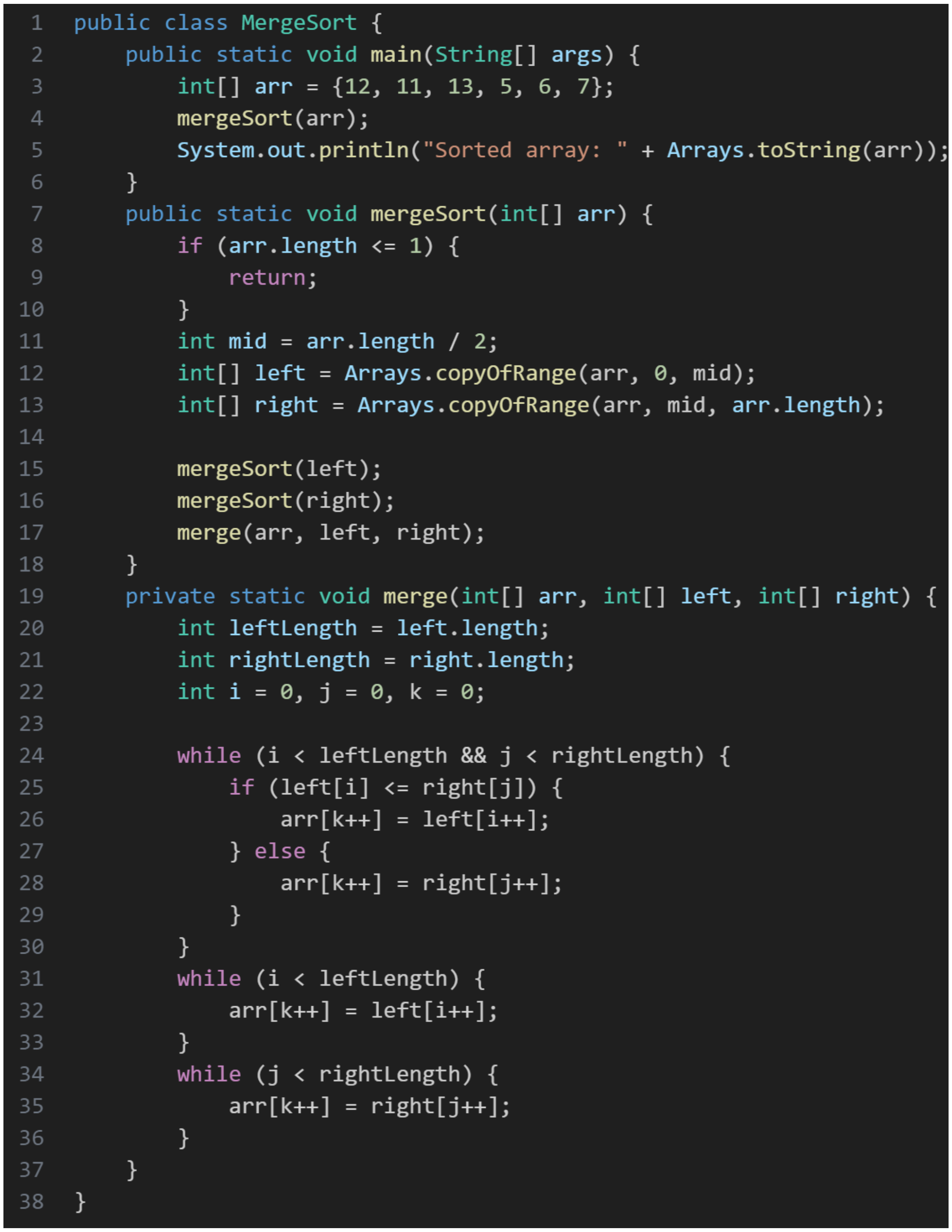
Explanation:

The mergeSort method is a recursive function that splits the input array into halves until each sub-array has a single element.

The merge method then combines these sorted sub-arrays back into a single sorted array.

The algorithm repeatedly divides the array into halves until individual elements are reached and then merges them back together in sorted order.

This implementation follows the functional paradigm by avoiding mutable states and instead creating new arrays for each division, maintaining the principle of immutability.



```java
public class MergeSort {
    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6, 7};
        mergeSort(arr);
        System.out.println("Sorted array: " + Arrays.toString(arr));
    }
    public static void mergeSort(int[] arr) {
        if (arr.length <= 1) {
            return;
        }
        int mid = arr.length / 2;
        int[] left = Arrays.copyOfRange(arr, 0, mid);
        int[] right = Arrays.copyOfRange(arr, mid, arr.length);

        mergeSort(left);
        mergeSort(right);
        merge(arr, left, right);
    }
    private static void merge(int[] arr, int[] left, int[] right) {
        int leftLength = left.length;
        int rightLength = right.length;
        int i = 0, j = 0, k = 0;

        while (i < leftLength && j < rightLength) {
            if (left[i] <= right[j]) {
                arr[k++] = left[i++];
            } else {
                arr[k++] = right[j++];
            }
        }
        while (i < leftLength) {
            arr[k++] = left[i++];
        }
        while (j < rightLength) {
            arr[k++] = right[j++];
        }
    }
}
```

In conclusion, recursion is an important concept in computer science that can make code easier to write and reduce the length of code. However, it also has its disadvantages, including slower computation times and the potential for the computer to run out of memory if the recursive calls are not properly checked. Nonetheless, with proper implementation, recursion can be a powerful tool in algorithm design

Sources :

Books:

- Data Structures and Algorithms in Java Fourth Edition
- Programming Languages: Principles and Paradigms

Websites:

- https://en.wikipedia.org/wiki/Recursion_(computer_science)
- https://www.youtube.com/watch?v=ivl5-snqul8
- https://www.youtube.com/watch?v=k-7jJP7QFEM
- https://www.youtube.com/watch?v=cum3OrpURzc
- https://www.collegenote.net/curriculum/data-structures-and-algorithms/41/454/