ATM Analogy.

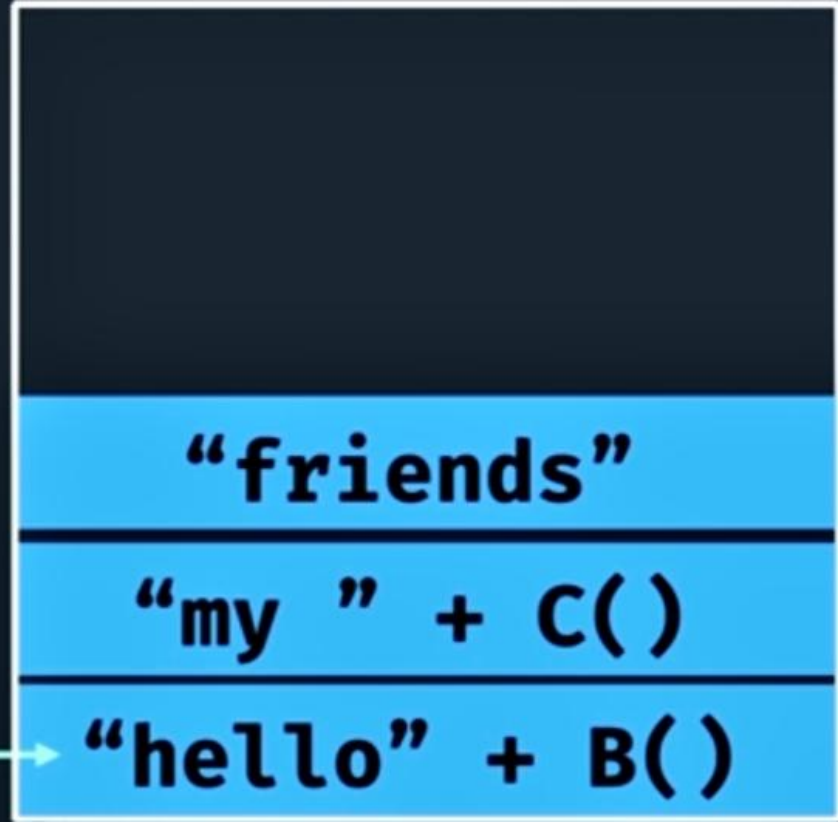
ATM Analogy.

```
1 public int getMyPositionInLine(Person person) {
2   if (person.nextInLine == null) {
3     return 1;
4   }
5
6   return 1 + getMyPositionInLine(person.nextInLine);
7 }
```

```
1 public void recursion(int someValue) {
2   if (someValue == 10){
3     return;
4   }
5   return recursion(someValue + 1);
6 }
```

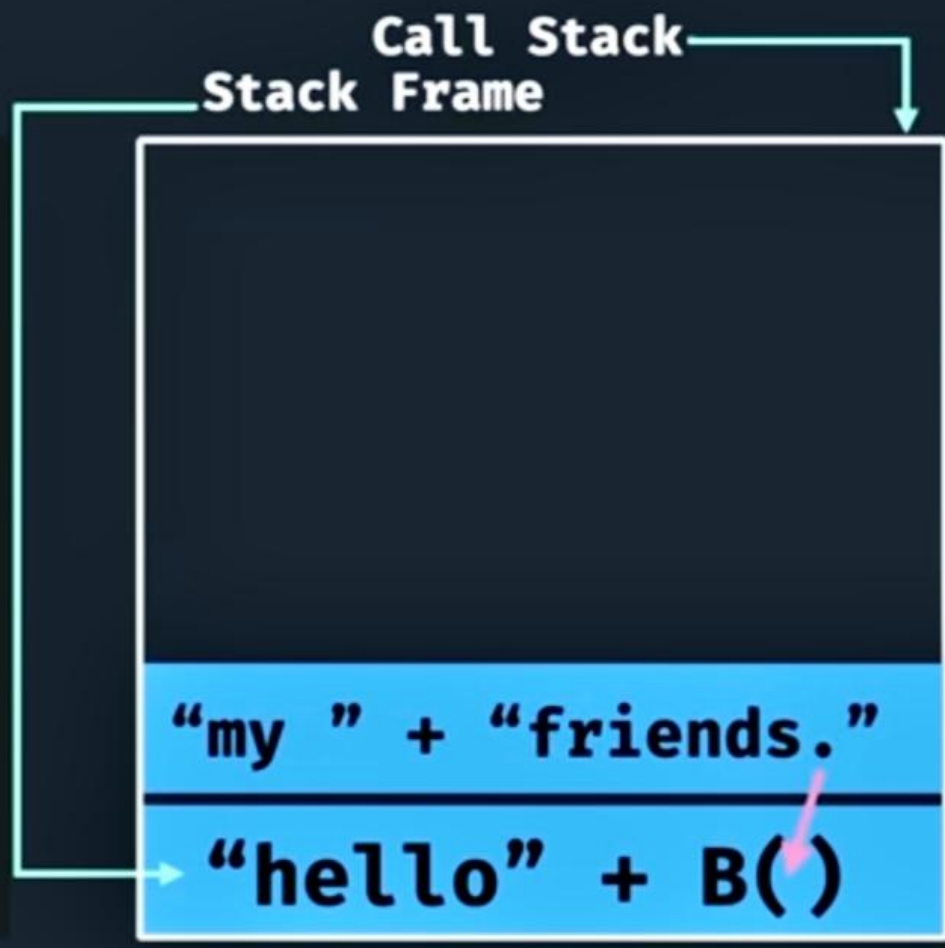
Pros	Cons
Bridges the gap between elegance and complexity	Slowness due to CPU overhead
Reduces the need for complex loops and auxiliary data structures	Can lead to out of memory errors / stack overflow exceptions
Can reduce time complexity easily with **memoization**	Can be unnecessarily complex if poorly constructed
Works really well with recursive structures like tress and graphs	

```java
// Expected output = "hello my friends."
public String A() {
  return "hello " + B();
}

public String B() {
  return "my " + C();
}

public String C() {
  return "friends.";
}
```



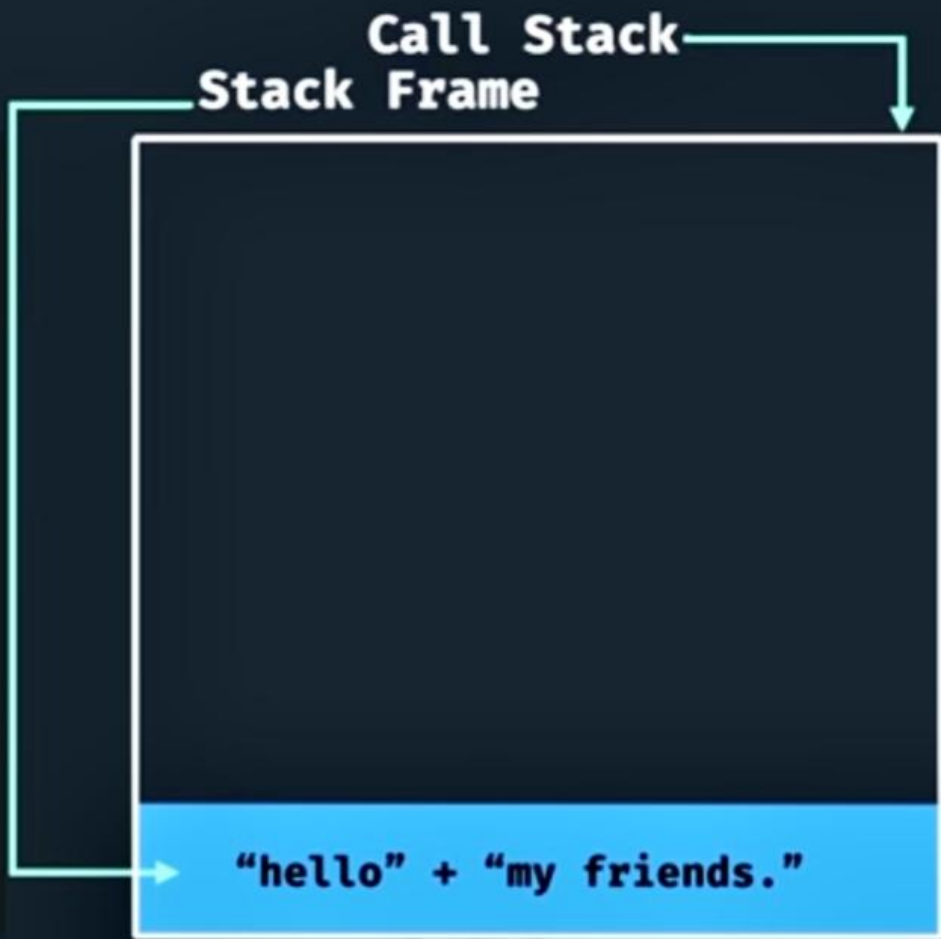
Call Stack.

```
1 // Expected output = "hello my friends."
2 public String A() {
3   return "hello " + B();
4 }
5
6 public String B() {
7   return "my " + C();
8 }
9
10 public String C() {
11   return "friends.";
12 }
```



Call Stack.

```
// Expected output = "hello my friends."
public String A() {
  return "hello " + B();
}

public String B() {
  return "my " + C();
}

public String C() {
  return "friends.";
}
```



Call Stack With Recursion.

```
1 public String A() {
2   return A();
3 }
```

String Reversal.

input: the simple engineer

output: reenigne elpmis eht

```java
public String reverseString(String input) {
  if (input.equals("")) {
    return "";
  }
  // What is the smallest amount of work I can do in each iteration?
  return reverseString(input.substring(1)) + input.charAt(0);
}
```

```java
public String reverseString(String input) {
  if (input.equals("")) {
    return "";
  }
  // What is the smallest amount of work I can do in each iteration?
  return reverseString(input.substring(1)) + input.charAt(0);
}
```

reverseString("") + "o"

reverseString("o") + "l"

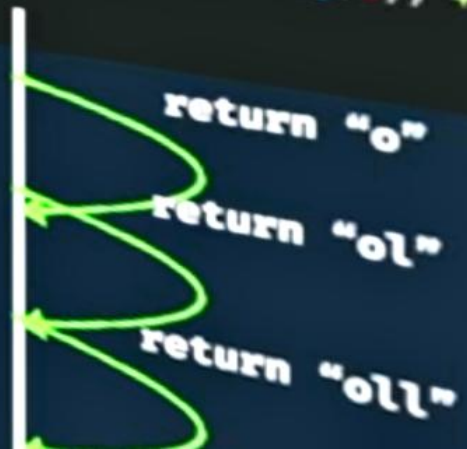
reverseString("lo") + "l"

reverseString("llo") + "e"

reverseString("ello") + "h"

input: "hello"

```
1 public String reverseString(String input) {
2   if (input.equals("")) {
3     return "";
4   }
5   // What is the smallest amount of work I can do in each iteration?
6   return reverseString(input.substring(1)) + input.charAt(0);
7 }
```



Palindrome.

kayak

```java
public static boolean isPalindrome(String input) {
    // Define the base-case / stopping condition
    if (input.length() == 0 || input.length() == 1) {
        return true;
    }

    // Do some work to shrink the problem space
    if (input.charAt(0) == input.charAt(input.length() - 1)) {
        return isPalindrome(input.substring(1, input.length() - 1));
    }

    // Additional base-case to handle non-palindromes
    return false;
}
```

```java
public static boolean isPalindrome(String input) {
    // Define the base-case / stopping condition
    if (input.length() == 0 || input.length() == 1) {
        return true;
    }

    // Do some work to shrink the problem space
    if (input.charAt(0) == input.charAt(input.length() - 1)) {
        return isPalindrome(input.substring(1, input.length() - 1));
    }

    // Additional base-case to handle non-palindromes
    return false;
}
```

Decimal To Binary.

```
233 // 2 = 116
116 // 2 = 58
58 // 2 = 29
29 // 2 = 14
14 // 2 = 7
7 // 2 = 3
3 // 2 = 1
```

```
rem = 1
rem = 1
rem = 1
rem = 0
rem = 1
rem = 0
rem = 0
rem = 1
```

```java
public static String findBinary(int decimal, String result) {
    if (decimal == 0)
        return result;

    result = decimal % 2 + result;
    return findBinary(decimal / 2, result);
}
```

```java
public static int recursiveSummation(int inputNumber) {
    if (inputNumber <= 1)
        return inputNumber;
    return inputNumber + recursiveSummation(inputNumber - 1);
}
```

VARIABLES

Local

args: String[0]@7

WATCH

CALL STACK

Thread [main] PAUSED ON BREAKPOINT

SumOfNaturalNumbers.main(String[]) S..

Thread [Reference Handler] RUNNING

Thread [Finalizer] RUNNING

Thread [Signal Dispatcher] RUNNING

Thread [Common-Cleaner] RUNNING

BREAKPOINTS

Uncaught Exceptions

Caught Exceptions

DecimalToBinary.java 3

DecimalToBinary.java 4

DecimalToBinary.java 12

SumOfNaturalNumbers.java 14

SumOfNaturalNumbers.java > SumOfNaturalNumbers > main(String[])

```
1  /**
2   * Sample class to demonstrate numerical recursion in Java by checking the sum of a number f
3   * Natural number recursion.
4   */
5  public class SumOfNaturalNumbers {
6
7      public static int recursiveSummation(int inputNumber) {
8          if (inputNumber <= 1)
9              return inputNumber;
10         return inputNumber + recursiveSummation(inputNumber - 1);
11     }
12
       Run | Debug
13     public static void main(String[] args) { args = String[0]@7
14         int result = recursiveSummation(5);
15         int result2 = recursiveSummation(10);
16         System.out.println(result);
17         System.out.println(result2);
18     }
19 }
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE Java Debug Console

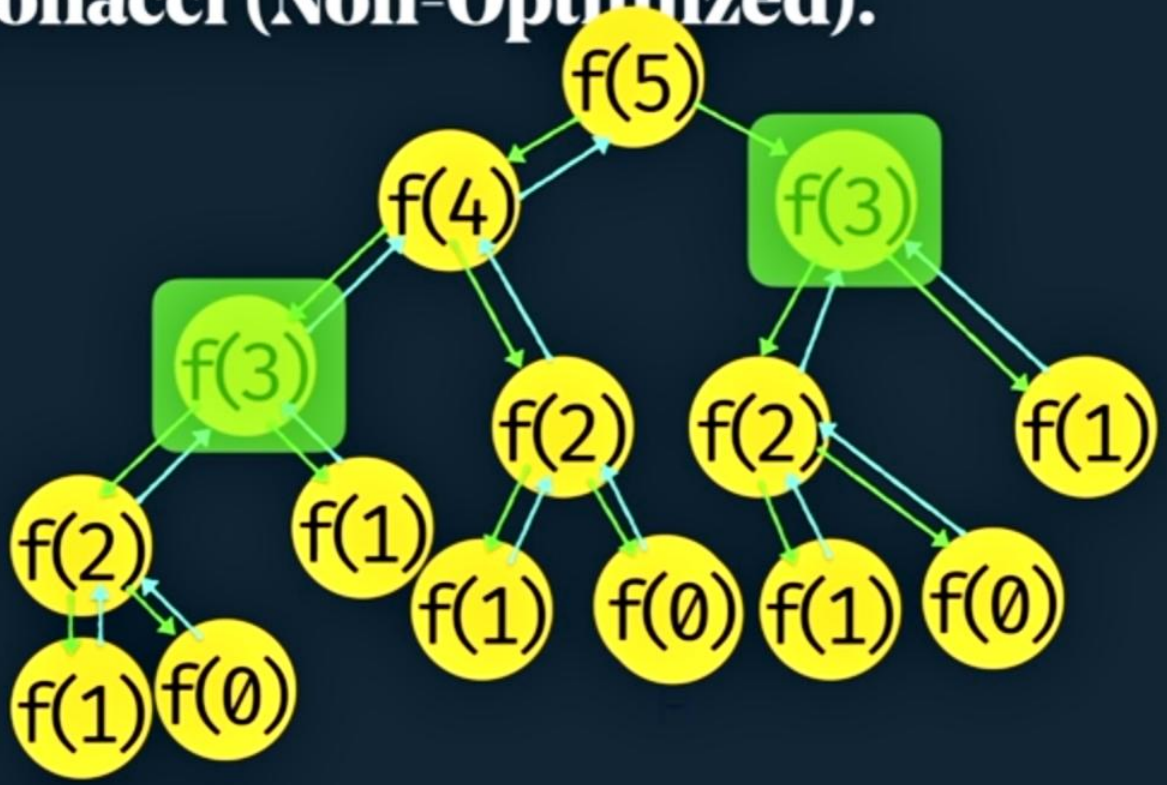
```
> cd /Users/schachte/Desktop/LiveCode ; /usr/bin/env /Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/b
in/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:63939 -Dfile.encoding=UTF-8 -cp "/Users/sch
achte/Library/Application Support/Code/User/workspaceStorage/f2cd88f30f4bda116e292ab7f651b4fd/redhat.java/jdt_ws/LiveCode_5
bfbfa06/bin" DecimalToBinary
> cd /Users/schachte/Desktop/LiveCode ; /usr/bin/env /Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/b
in/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:64366 -Dfile.encoding=UTF-8 -cp "/Users/sch
achte/Library/Application Support/Code/User/workspaceStorage/f2cd88f30f4bda116e292ab7f651b4fd/redhat.java/jdt_ws/LiveCode_5
bfbfa06/bin" SumOfNaturalNumbers
```

```java
public static int binarySearch(int[] A, int left, int right, int x) {
    if (left > right) {
        return -1;
    }
    int mid = (left + right) / 2;

    if (x == A[mid]) {
        return mid;
    }

    if (x < A[mid]) {
        return binarySearch(A, left, mid - 1, x);
    }

    return binarySearch(A, mid + 1, right, x);
}
```

binarySearch(A, 8, 9, 10)

```
1 public static long fib(long n) {
2     if ((n == 0) || (n == 1))
3         return n;
4     else
5         return fib(n - 1) + fib(n - 2);
6 }
```

Fibonacci (Non-Optimized).



```java
public Node SortedMerge(Node A, Node B) {
    if(A == null) return B;
    if(B == null) return A;

    if(A.data < B.data) {
        A.next = SortedMerge(A.next, B);
        return A;
    } else {
        B.next = SortedMerge(A, B.next);
        return B;
    }
}
```

```java
public Node SortedMerge(Node A, Node B) {
    if(A == null) return B;
    if(B == null) return A;

    if(A.data < B.data) {
        A.next = SortedMerge(A.next, B);
        return A;
    } else {
        B.next = SortedMerge(A, B.next);
        return B;
    }
}
```

sortedMerge(1, 4)

```java
public Node SortedMerge(Node A, Node B) {
    if(A == null) return B;
    if(B == null) return A;

    if(A.data < B.data) {
        A.next = SortedMerge(A.next, B);
        return A;
    } else {
        B.next = SortedMerge(A, B.next);
        return B;
    }
}
```

~~sortedMerge(22, null)~~
sortedMerge(22, 20)
sortedMerge(22, 16)
sortedMerge(22, 11)
sortedMerge(8, 11)
sortedMerge(8, 4)
sortedMerge(1, 4)

```java
public Node SortedMerge(Node A, Node B) {
    if(A == null) return B;
    if(B == null) return A;

    if(A.data < B.data) {
        A.next = SortedMerge(A.next, B);
        return A;
    } else {
        B.next = SortedMerge(A, B.next);
        return B;
    }
}
```

return 22

sortedMerge(22, null)
sortedMerge(22, 20)
sortedMerge(22, 16)
sortedMerge(22, 11)
sortedMerge(8, 11)
sortedMerge(8, 4)
sortedMerge(1, 4)

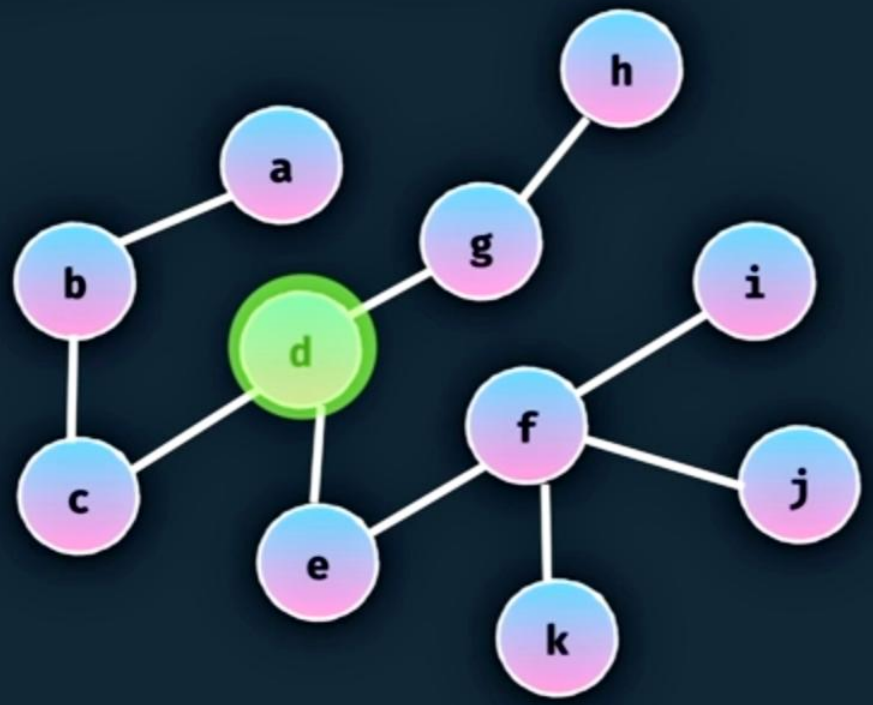
```java
public static ListNode reverseList(ListNode head) {
    if (head == null || head.next == null) return head;
    ListNode p = reverseList(head.next);
    head.next.next = head;
    head.next = null;
    return p;
}
```

```java
public static ListNode reverseList(ListNode head) {
    if (head == null || head.next == null) return head;
    ListNode p = reverseList(head.next);
    head.next.next = head;
    head.next = null;
    return p;
}
```

```
1 public Node insertNode(Node head, int data) {
2     if(head == null){
3         head = new Node();
4         head.data = data;
5         return head;
6     }
7     if(head.data < data) {
8         head.right = insertNode(head.right,data);
9     } else {
10        head.left = insertNode(head.left, data);
11    }
12    return head;
13 }
```

```java
public static void printLeaves(Node root) {
    if (root == null) return;

    if (root.left == null && root.right == null) {
        System.out.print(root.val + ", ");
        return;
    }
    if (root.left != null)
        printLeaves(root.left);
    if (root.right != null)
        printLeaves(root.right);
}
```

Depth-First Search.



```java
boolean depthFirstSearch(Node node, Set<Node> visited, int goal) {
  if (node == null) return false;

  if (node.val == goal) {
    return true;
  }

  for (Node neighbor : node.getNeighbors()) {
    if (visited.contains(neighbor)) continue;
    visited.add(neighbor);
    boolean isFound = depthFirstSearch(neighbor, visited, goal);

    if (isFound) return true;
  }
  return false;
}
```