# Unit 8: Recursion

## Introduction

A recursive method is a method that calls itself. The structure of all recursive methods is that there is a base case (this can be do nothing) and a recursive case separated in an if/else structure. The recursive case calls the method with different parameters that must ultimately call the recursive base case.

Typical Form of Recursion:

public static void recursive(int n) {

if (n == base\_case) { //base case

//process base case

} else { //recursive case

//recursively call recursive

}

}

## Common Errors

### Recursive calls don’t progress towards base case

A recursive method that does not progress towards a base case will cause infinite recursion (similar to a loop whose condition is always true). This is a programming logic error to avoid.

The following two code chunks are equivalent.

|  |  |
| --- | --- |
| Infinite Recursion | Infinite Loop |
| public static int recur(int n) {  if (n == 0) {  return 0;  } else {  return n + recur(n + 1);  }  recur(10); | int n = 0;  for (int i = 10; i >= 0; i++) {  n = n + i;  } |

Notice that each recursive call increases n but our base case is when n == 0. Similarly, the for loop condition is true while i >= 0 and since i is increasing at each step it will never end (until i equals Integer.MAX\_VALUE).

### No base case given

Forgetting a base case can cause infinite recursion as well.

|  |  |
| --- | --- |
| Infinite Recursion | Infinite Loop |
| public static int recur(int n) {  return n + recur(n - 1);  }  recur(10); | int n = 0;  for (int i = 10; true; i--) {  n = n + i;  } |

There is no base case, so the program will continue calling itself forever. This is similar to the infinite loop where the condition is always true.

## Tracing Recursive Code

To understand what a recursive method will do, it is helpful to make a table of all of the recursive calls. For example,

public static void funnyPrinter(String s) {

|  |  |  |
| --- | --- | --- |
| s | action | output |
| “cat” | print(“c”), funnyPrinter(“at”), print(“c”) | “cattac” |
| “at” | print(“a”), funnyPrinter(“t”), print(“a”) | “atta” |
| “t” | print(“t”), funnyPrinter(“”), print(“t”) | “tt” |
| “” | Do nothing |  |

if (s.length() > 0) {

System.out.print(s.substring(0,1));

funnyPrinter(s.substring(1));

System.out.print(s.substring(0,1));

}

}

funnyPrinter(“cat”);

# Unit 8: Recursion – Problems

1. Write a recursive method that adds and returns the numbers from 1 to n inclusive.

public static int sum(int n) {

}

1. Write a recursive method that returns a String in reverse order. For example, reverse(“cow”) would return “woc”.

public static String reverse(String s) {

}

1. The following method is designed to determine whether a String is a palindrome. A palindrome is a String that reads the same forwards and backwards. For example, racecar, madam and poop are palindromes.

public static boolean isPalindrome(String s) {

if (s.length() == 0 || s.length() == 1) {

return true;

}

return s.substring(0,1).equals(s.substring(s.length() – 1)) &&

/\* missing code \*/;

}

What should replace /\* missing code \*/ so that isPalindrome works as intended?

1. isPalindrome(s.substring(1))
2. isPalindrome(s.substring(0,1))
3. isPalindrome(s.substring(1, s.length()))
4. isPalindrome(s.substring(1, s.length() – 1))
5. isPalindrome(s.substring(0,1) + s.substring(s.length() – 1))
6. Suppose the following method is declared.

public static int doubleTrouble(int n) {

if (n == 0) {

return 1;

} else {

return doubleTrouble(n – 1) + doubleTrouble(n – 1);

}

|  |  |
| --- | --- |
| n | result |
|  |  |

}

What will the call doubleTrouble(5) return?

What does doubleTrouble compute?

Is this an efficient algorithm? Why or why not?

1. Suppose the following method is declared.

public static int mystery(int a, int b) {

if (b == 0) {

return a;

} else {

return mystery(b, a % b);

}

}

What will the following calls return?

mystery(10, 5);

mystery(6, 9);

mystery(24, 18);

mystery(48, 56);

1. Consider the implementation of mergesort below. Assume that merge has been written correctly.

public static void mergesort(int[] a) {

if (a.length > 1) {

int[] left = Arrays.copyOfRange(a, 0, a.length/2);

int[] right = Arrays.copyOfRange(a, a.length/2, a.length);

/\* missing code \*/

merge(a, left, right);

}

}

What should replace missing code so that mergesort works as intended?