

# DNHI Homework 2 Solutions Recursion

## **Problem 1**

**Part A** Write an iterative method that computes a value of  $x^n$  for a positive integer n and a real number x.

Answer

The return value of -1 indicates an error condition.

```
public static double powerIter (double x, int n) {
   double result = 1;
   if (n < 0 ) return -1;
   while (n > 0) {
      result = result * x;
      n--;
    }
}
return result;
```

**Part B** Write a recursive method that computes a value of  $x^n$  for a positive integer n and a real number x. Answer: The return value of -1 indicates an error condition. X

```
1 public static double powerRecursive (double x, int n) {
2
3   if (n < 0) return -1;
4   if (n == 0) return 1;
5   return x * powerRecursive (x, n - 1);
6 }</pre>
```

## **Problem 2**

Consider the following recursive method

```
public int recMethod ( int number ) {
   if ( number <= 0 )
      return 0;
   if ( number % 2 == 0 )
      return recMethod ( number - 1 );
   else
      return number + recMethod ( number - 1);
   }
}</pre>
```

#### Part A

How many times is this method called (including the initial call) when we run recMethod (10) ?

Answer: Called 11 times.

How many times is this method called (including the initial call) when we run recMethod (-10)?

Answer: Called 1 time.

#### Part B

What does recMethod do (i.e. what does it compute)?

Answer: It computes the sum of odd numbers from zero to number.

## **Problem 3**

Write a recursive method to compute the following series:

$$\frac{1}{3} + \frac{2}{5} + \frac{3}{7} + \frac{4}{9} + \ldots + \frac{i}{2i+1}$$
.

Answer: The crucial part in this code is casting to double so that the fractions do not become all zero. Other than that it should be a straight forward implementation of the recursive method.

```
public static double summation ( int num ) {
   //base case
   if ( num <= 0 ) return 0;
   //recursion
   return (double) num/(2*num+1) + summation(num-1);
   6}</pre>
```

## **Problem 4**

Write a recursive method that computes the sum of the digits in an integer. Use the following method header:

```
public static int sumOfDigits ( long n )
```

For example, sumOfDigits(234) should return 9 (since 2+3+4=9) and sumOfDigits(390) should return 12 (since 3+9+0=12). Answer: A possible solution could be:

```
public static int sumOfDigits ( long n ) {
    //base case is when the number is zero

if( n==0 )
    return 0;

//recursive case
return ( (int) (n%10) + sumOfDigits( n/10 ) );

}
```

## Problem 5

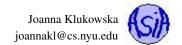
For each of the following recursive methods, rewrite it using iterations instead of recursion. HINT: in order to do so you should first figure out what these methods do.

## Part A

```
public int recur ( int n ) {
   if (n < 0 )
      return -1;
   else if ( n < 10 )
      return 1;
   else
      return ( 1 + recur ( n / 10 ) );
}</pre>
```

## Answer:

The code above computes the number of digits in the parameter n.



```
public int recur (int n) {
   int solution = 0;
   while (n > 0 ) {
       solution++;
       n = n/10;
   }
   return solution;
   8}
```

## Part B

```
public int recur2 ( int n ) {
   if (n < 0 )
     return -1;
   else if ( n < 10 )
     return n;
   else
     return ( n % 10 + recur2 ( n / 10 ) );
}</pre>
```

#### Answer:

This method computes sum of digits in a parameter n.

```
public int recur2 ( int n ) {
  int sum = 0;
  while (n > 0 ) {
     sum += n%10;
     n = n/10;
     }
  return sum; fs
}
```

## Problem 6

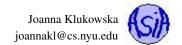
What would be printed by the following programs

#### Part A)

```
1 public class CatsAndDogs {
    public static void main(String[] args) {
 3
     foo("Cats and Dogs", 4);
4
5
6
   public static void foo ( String s, int n ) {
7
      if (n <= 1)
8
9
        System.out.println("Cats");
10
11
        System.out.println(s);
12
        foo (s, n-1);
13
14
15 }
```

## Answer:

Cats and Dogs Cats and Dogs



```
Cats and Dogs
Cats
```

## Part B)

```
1 public class Numbers {
 3
    public static void main(String[] args) {
 4
      int [] list = {1, 2, 3, 4, 5};
5
      System.out.println(foo (list, 0, list.length-1));
 6
7
8
    public static int foo ( int [] nums, int begin, int end ) {
      if ( begin == end )
9
10
        return nums[begin];
11
        return nums[begin] + foo(nums, begin+1, end);
12
13
14 }
```

#### Answer:

The foo method computes the sum of the values in the list from between index begin and index end. So in this case it computes the sum of all elements in the list. It prints 15

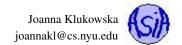
## **Problem 7**

**Part A** Write a method that generates all sequences of a given length that contain digits 0 through 9 (all ten digits are allowed, repetitions are allowed)? Given length of the sequence equal to n, how many possible sequences are there?

## Answer:

With length of n digits, the number of possible sequences is equal to  $10^n$ , for example, with length of n = 4, we have 10,000 different sequences.

```
1 / * *
    \star Generate all decimal sequences of the specified length.
    \star @param length the length of the sequences to be generated
5 public static void getAllDecimalSequences ( int length ) {
   String seq = new String () ;
   int counter = 0;
    getAllDecimalSequences( length, seq);
8
9 }
10
11 /\star Generate all decimal sequences of a specified length
   * using the seq String as storage for partial sequences.
   * Oparam length the length of the sequence to be generated
   \star @param seq stores partial sequences between recursive calls
14
15
16 private static void getAllDecimalSequences ( int length, String seq ) {
    if (seq.length() == length ) {//reached the desired length
17
      System.out.printf("%s %n", seq );
18
19
20
    else { //add the next digit to the sequence (two possibilities)
21
      for (int i = 0; i < 10; i++) {</pre>
22
        //add digit i to the current sequence
2.3
        getAllDecimalSequences( length, seq + Integer.toString(i));
24
25
26
27 }
```



**Part B** Modify the above method so that none of the generated sequences start with zero. How many of those sequences exist, given the length of n digits?

## Answer:

With this restriction, we only have 9 possiblities for the first digit and 10 for all the remaining digits. So there will be total of  $9 \times 10^{n-1}$  sequences of length n that do not start with a zero.

```
1 / * *
   * Generate all decimal sequences of the specified length with added
    * constraint that the first digit is never zero.
    \star Oparam length the length of the sequences to be generated
6 public static void getDecimalSequencesNoLeadingZero ( int length ) {
    String seq = new String () ;
    getDecimalSequencesNoLeadingZero( length, seq);
8
9 }
10
11 / *
    \star Generate all decimal sequences of the specified length with added
    \star constraint that the first digit is never zero. seq is used for storage
13
    * of partial sequences.
14
    * @param length the length of the sequences to be generated
15
    \star @param seq stores partial sequences between recursive calls
16
17
18 private static void getDecimalSequencesNoLeadingZero ( int length, String seq ) {
    if (seq.length() == length ) {//reached the desired length
      System.out.printf("%s %n", seq );
20
21
22
    else if (seq.length() == 0) { //do not start any sequence with a zero}
      for (int i = 1; i < 10; i++) {</pre>
23
        //add digit i to the current sequence
24
        getAllDecimalSequences( length, seq + Integer.toString(i));
25
26
27
28
    else {
      for (int i = 0; i < 10; i++) {</pre>
29
        //add digit i to the current sequence
30
        getAllDecimalSequences( length, seq + Integer.toString(i));
31
32
33
34 }
```