CIS211: Data Structures

Delaware Tech, CT Department

Wilmington

(Instructor: Lee Hsu)

**Project #5: Recursive vs. Non-recursive Solution**

1. **Introduction:**

The computational complexity of the recursive and non-recursive functions is theoretically the same, but the overhead involved in manipulating the run-time stack for a recursive function does produce some inefficiency. Consequently in cases where both approaches can be developed, the non-recursive solution is usually preferred.

However, there are problems where recursive solutions are so natural that we are likely tempted to use them. Therefore, in this project, in addition to reviewing the basic concepts of recursive functions through simple problems of finding factorial and Fibonacci numbers, you are going to compare the time efficiency of recursive and non-recursive approaches, so that when the time comes, you may make a better decision whether to use the recursive solution or not.

1. **Assignments:**
2. Write the recursive and nonrecursive version of
   1. Int **factorial**(int n), (use **nFactorial** as the name for non-recursive version)  
      (where n >= 0)
   2. Int **fibonacci**(int n), (and **nFibonacci** as the name for non-recursive version)
   3. boolean **isPalindrome**(String s, int low, int high), (and **nIsPalindrome**)
   4. int **binarySearch**(int[] v, int key, int low, int high), (and **nBinarySearch**)
3. Develop testing schemes as following:
   1. For a and b: use at least 20 different values of n (from small to sufficiently large) and the time function to find the time spent for different values of n.
   2. For c, set up a string array of size 20 and initialize them with at least 10 palindromes.
   3. For d, do the following:
      1. Fill an integer array of 1000 multiples of 3 from 0 to 2997 (i.e., 0, 3, 6, 9,…,2997) and have them properly sorted.
      2. Test your function and see if it works.
      3. Set up a test array with 20 random integers between 0 and 3000 to test the program.
4. Use Spreadsheet to tabulate the results and use its graph function to plot the time vs. n graph.
5. You may combine all functions in one program (project) and send your output to a file named **p5Out(yourName).txt**. Also name your spreadsheet **p5Out(yourName).slxs.** Finally, combine all result into one word document and briefly summarize your finding into **p5Recursion(yourName).docx.**
6. A sample summary write-up is available for reference.
7. You may consider to use BigInteger Class in case long type is not big enough. For more information, see the following URL and the sample program below:

http://www.geeksforgeeks.org/biginteger-class-in-java/

import java.math.\*;

public class LargeFactorial {

public static void main(String[] args) {

System.out.println("50! is \n" + factorial(50));

}

public static BigInteger factorial(long n) {

BigInteger result = BigInteger.ONE;

for (int i = 1; i <= n; i++)

result = result.multiply(new BigInteger(i+""));

return result;

}

}

1. Name your Java program **p5Recursion.java**.
2. **Notes:**
3. This assignment challenges not only your programming skills but also your research skills. Programming is easy. You should have learned how to write these two functions in CSC164 or CIS150. So, the focus should be on how to select a proper series of N so that you can produce a more meaningful result. For example, you may select a series of 20 numbers for N from 5 to 100 with increments of 5.
4. When N get sufficiently large, long int type may not be big enough to handle the resulting value, so you should have the result converted to double data type (use “printf( “%E”…)” format to print the result) or use BigInteger class as noted above.
5. Before you apply the “nanoTime” function, you should print and compare the result for both recursive and non-recursive version to make sure your implementation is correct.
6. Your spreadsheet should contain graphs for your results. The graphs should look similar to the following:

|  |  |
| --- | --- |
|  |  |

1. Since the time taken to execute the program also depends on the hardware you use, so your results will vary. However, the graph should look similar.

import java.util.Scanner;

public class ComputeFactorial {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("Enter a non-negative integer: ");

int n = input.nextInt();

System.out.println("Factorial of " + n + " is " + factorial(n));

}

public static long factorial(int n) {

if (n == 0) // Base case

return 1;

else

return n \* factorial(n - 1); // Recursive call

}

}

import java.util.Scanner;

public class ComputeFibonacci {

public static void main(String args[]) {

Scanner input = new Scanner(System.in);

System.out.print("Enter an index for the Fibonacci number: ");

int index = input.nextInt();

System.out.println(

"Fibonacci number at index " + index + " is " + fib(index));

}

public static long fib(long index) {

if (index == 0) // Base case

return 0;

else if (index == 1) // Base case

return 1;

else // Reduction and recursive calls

return fib(index - 1) + fib(index - 2);

}

}

public class RecursivePalindrome {

public static boolean isPalindrome(String s) {

return isPalindrome(s, 0, s.length() - 1);

}

public static boolean isPalindrome(String s, int low, int high) {

if (high <= low) // Base case

return true;

else if (s.charAt(low) != s.charAt(high)) // Base case

return false;

else

return isPalindrome(s, low + 1, high - 1);

}

public static void main(String[] args) {

System.out.println("Is moon a palindrome? " + isPalindrome("moon"));

System.out.println("Is noon a palindrome? " + isPalindrome("noon"));

System.out.println("Is a a palindrome? " + isPalindrome("a"));

System.out.println("Is aba a palindrome? " + isPalindrome("aba"));

System.out.println("Is ab a palindrome? " + isPalindrome("ab"));

}

}

public class RecursiveBinarySearch {

public static int recursiveBinarySearch(int[] list, int key) {

int low = 0;

int high = list.length - 1;

return recursiveBinarySearch(list, key, low, high);

}

public static int recursiveBinarySearch(int[] list, int key, int low, int high) {

if (low > high) // The list has been exhausted without a match

return -low - 1;

int mid = (low + high) / 2;

if (key < list[mid])

return recursiveBinarySearch(list, key, low, mid - 1);

else if (key == list[mid])

return mid;

else

return recursiveBinarySearch(list, key, mid + 1, high);

}

}