Runtime Analysis: Iterative Vs. Recursive Power Function

Mathew Lister
Department of Computer Science
Seattle Pacific University
Seattle, United States
listerm@spu.edu

***Abstract*—The objective of this paper is to analyze the difference in runtime for a basic power function using iterative and recursive implementation methods. This experiment was done to determine which implementation method was most effective across a large data set. The results of the experiment clearly show that the iterative implementation method is a little over twice as fast as the recursive method.**

Keywords—iteration, recursion, stack, power function, algorithm, runtime analysis

I. Introduction

The problem at hand is to determine which implementation method of a simple power function is faster. One implementation uses recursion while the other uses iteration. It is important to analyze possible problem-solving techniques to ensure a computerized system is as optimized as possible. This ensures that no power or memory is being wasted due to poor design of implementation.

II. Background

In computer science, recursion is a problem-solving technique in which the problem is continuously broken down into smaller parts and then combined to reach the results [1]. The breaking up of a problem into smaller parts is done by making the function recursively call itself [2]. However, with iteration, the problem is looped through a set of operations continuously until the problem is done [3]. The two styles of problem solving come with their own set of pros and cons and which one is better heavily depends on the problem that's trying to be solved. A stack is, "a container of objects that are inserted and removed according to the last-in-first-out (LIFO) principle" [3]. Stacks are utilized when recursion is used by pushing the function calls onto the stack and then popping them off to "combine the smaller parts of a problem into a solution" as mentioned above [4]. The power function used in this experiment simply raises the base number to the exponent (x^n) where (x) is the base and (n) is the exponent.

III. Methodology

The programming language Java (version 13) was used in this experiment to empirically determine the performance of two power function implementations across a large data set. Timestamps for start and end where captured using Javas nanoTime() function that is a part of the built-in System class [5]. The base number throughout the entire experiment was 3.14159265359, while the exponent started at one and increased one full integer each time the function was re-called.

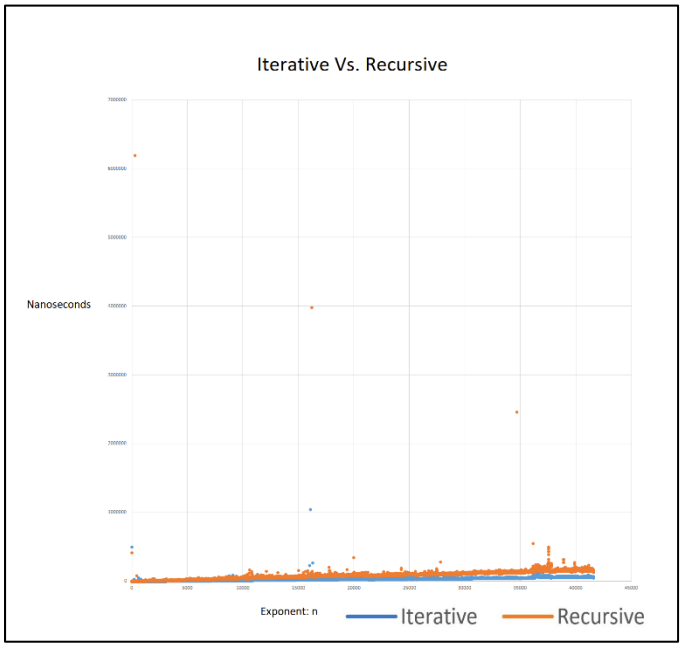
The experiment ran as follows: capture current timestamp, run iterative power function, capture current timestamp, subtract the two timestamps to find elapsed time, capture current timestamp, run recursive power function, capture current timestamp, subtract the two timestamps to find elapsed time, write exponent used with the time elapsed for the two functions to a .csv file. The exponent was then increased by one and the process started over. This was done until the local machine running the program ran out of memory. Javas OutOfMemoryError (OOME) was used in a try-catch to know when the local machines limits had been reached [6]. The data points were then graphed on a scatter plot using excel. The steps can be viewed in Appendix1 via the comments above each line of code within the try-catch block.

The local machine used in this experiment was a surface laptop 2. System specifications are, Intel Core i5-8250U CPU 1.80GHz, 8 GB RAM, Windows 10.

IV. Results

The surface laptop 2 was able to call the two functions about 40,000 – 42,000 times before Java reported an out of memory error. In this situation, the iterative power function performed almost always twice as fast as the recursive power function.

Graph I. Scatter Plot of Iterative Vs. Recursive



V. Conclusions

What bounds the possible values of n in this experiment is how much system memory you have or RAM. Because each time the recursive function is called a new frame on the stack must be created. As this process continues the stack will reach a point where there is no more memory to add another frame and that is when Java will report an OOME [7]. This is also the main reason why iteration was faster in this experiment because the program didn't need to waste processing power and memory on constantly adding frames onto the stack. This is also known as stack overflow and it happens when a program "recurses too deeply" [8]. This experiment was extremely useful for my professional development because it proves that I can analyze different ways of solving a problem to choose the best one. In business the main concern is making money and not wasting it, if a company is wasting valuable resources, they are wasting money. With experiments like this one, I can help businesses use their resources to the best of their ability and work from them, not against them.

References

[1] "Recursion Lecture Slides", CS211 School of Computer Science, Cornell University, Ithaca, NY.

[2] "Lecture 15 More Recursion", CS1110 School of Computer Science, Ithaca, NY, Oct, 21, 2014.

[3] V. s. Adamchik, "Stacks and Queues," *Carnegie Mellon*, 2009. [Online]. Available: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Stacks and Queues/Stacks and Queues.html. [Accessed: 15-Jan-2020].

[4] "What on Earth is Recursion?," *YouTube*, 16-May-2014. [Online]. Available: https://www.youtube.com/watch?v=Mv9NEXX1VHc. [Accessed: 15-Jan-2020].

[5] "Java Class System," *System (Java Platform SE 7), Oracle*, 06-Oct-2018. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/lang/System.html. [Accessed: 15-Jan-2020].

[6] "3.2 Understand the OutOfMemoryError Exception, *Oracle*, 22-Dec-2014. [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/memleaks002.html. [Accessed: 15-Jan-2020].

[7] V. S. Adamchik, "Recursive Programming," *Carnegie Mellon*, 2009. [Online]. Available: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Recursions/recursions.html. [Accessed: 15-Jan-2020].

[8] "StackOverflowError (Java Platform SE 7)," *Oracle*, 06-Oct-2018. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/java/lang/StackOverflowError.html. [Accessed: 15-Jan-2020].

Appendix 1

Source Code

```
package Main;

import java.io.FileWriter;
import java.io.IOException;

public class DriverCode {
    public static void main(String[] args) throws IOException {
        int n = 1;
         long recursiveStartTime, recursiveElapsedTime, iterativeStartTime,
        iterativeElapsedTime;
        FileWriter csvWriter = new FileWriter("results.csv");

        while (true) {
            try {
                //Start clock
                iterativeStartTime = System.nanoTime();
                //Run iterative algorithm
                Iterative.iterativePower(3.14159265359, n);
                //Stop clock
                iterativeElapsedTime = System.nanoTime() - iterativeStartTime;

                //Start clock
                recursiveStartTime = System.nanoTime();
                //Run recursive algorithm
                Recursive.recursivePower(3.14159265359, n);
                //Stop clock
                recursiveElapsedTime = System.nanoTime() - recursiveStartTime;

                //write to csv
                csvWriter.append(Integer.toString(n));
                csvWriter.append(",");
                csvWriter.append(Long.toString(iterativeElapsedTime));
                csvWriter.append(",");
                csvWriter.append(Long.toString(recursiveElapsedTime));
                csvWriter.append("\n");

                n++;
            } catch (OutOfMemoryError e) {
                System.out.println("OOME reached!");
                break;
            }
        }

        //Close fileReader
        csvWriter.flush();
        csvWriter.close();
    }
}
```

```java
package Main;

class Iterative {
    static double iterativePower(double base, int exponent){
        double retVal = 1.0;
        if (exponent < 0){
            return 1.0 / iterativePower(base, -exponent);
        }else{
            for (int i=0; i<exponent; i++)
                retVal *= base;
        }
        return retVal;
    }
}
```

```java
package Main;

class Recursive {
     static double recursivePower(double base, int exponent){
        if (exponent < 0){
            return 1.0 / recursivePower(base, -exponent);
        }else if (exponent == 0){
            return 1.0;
        }else {
            return base * recursivePower(base, exponent - 1);
        }
    }
}
```