- Matthew Vollkommer
- 8102494122
- HW 3

Q1) This statement is true for Java. Iteration is usually less expensive and will run faster than its recursive equivalent.

Q2) This statement is not true. See code for Q2 in the Addendum

The iterative method is faster than the recursive method.
Both iteration and recursion involve repetition: Recursion produces repetition through repeated method calls. Iteration explicitly uses repetition.

Q3) N(y) = N(y-1) * x;
N(0) = 1;
y>0;

Code written in addendum

Q4)

```
Matthews-MacBook-Pro:hanoi MattsMac$ make run
java SolveHanois
Move one disk from 1 to 2
Move one disk from 1 to 3
//recursion
Move one disk from 2 to 3
Move one disk from 1 to 2
//recursion
Move one disk from 3 to 1
Move one disk from 3 to 2
//recursion
Move one disk from 1 to 2
Move one disk from 1 to 3
//recursion
Move one disk from 2 to 3
Move one disk from 2 to 1
//recursion
Move one disk from 3 to 1
Move one disk from 2 to 3
//recursion
Move one disk from 1 to 2
Move one disk from 1 to 3
//recursion
Move one disk from 2 to 3
```

Q5)

Disks	Steps
n	(2^n) -1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023
15	32767
20	1048575
25	33554431

Q6)

In this experiment, we will test the difference between recursion and iterations.

Hypothesis: If the method for solving the problem is recursive, it will take longer to solve the problem than its iterative equivalent.

To test this Hypothesis, I will use to different trial examples. Example 1. I will test out Fibonacci numbers. I expect the Recursive method to have a O(n). So each additional trial will take an additional second.

I expect the Iterative method to have an O(1). I do not expect to see an increase in the amount of time the iterative method takes with each additional trial and increase in number.

For the second example, I will use the Hanoi example, which we have thoroughly covered in class as well as in the textbook. I expect once again that the Recursive method to solve the experiment to take longer than its iterative equivalent

See following pages for results:
See Addendum for code

Conclusion:
My Hypothesis was correct. The Recursive methods were definitely slower than their iterative equivalents. However, I did not completely understand just how much slower they were. For the Fibonacci experiment in particular, the recursive method took a significant amount of time, while the iterative method seemed to take no time at all. However, I do see an advantage for Recursive method calls, I find them to be more elegant and easier to write as a programmer. The mathematics behind recursion is far more beautiful than the fast, but brutish iterative loops.

Tasks that are difficult to program and do not take up significant amounts of memory (ie with few iterations) should be written recursively. It is easier to visualize as well as easier to code. Tasks that are larger and time important should be coded iteratively with loops. These may take the programmer longer to write, but they will create less time waiting for the end user. So recursive methods are for quicker programming, while iterative looping methods are for quicker end user experience.

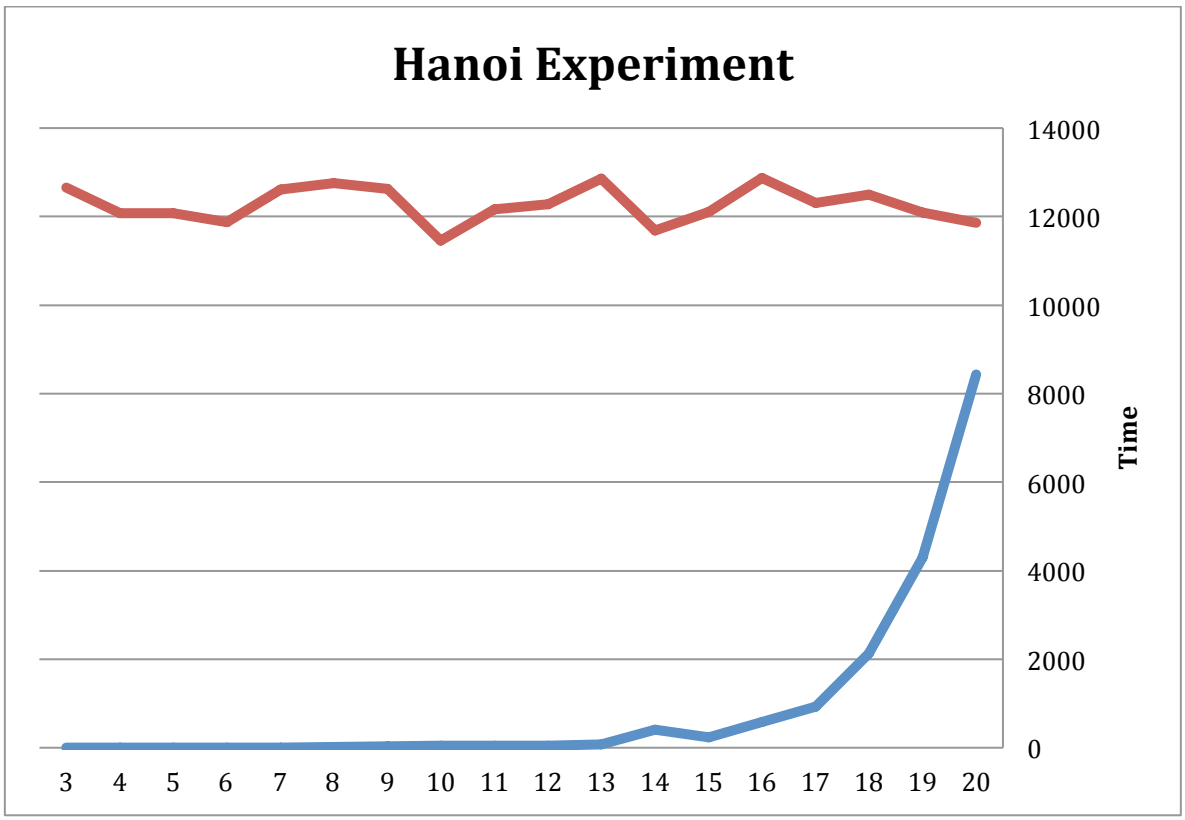
An interesting note to test in the future. I noticed that an overflow error occurred On both examples with the recursive method, yet the iterative method was able to solve the problem. It would be nice to experiment to measure just how much more memory the recursive method was using than the iterative method was using.

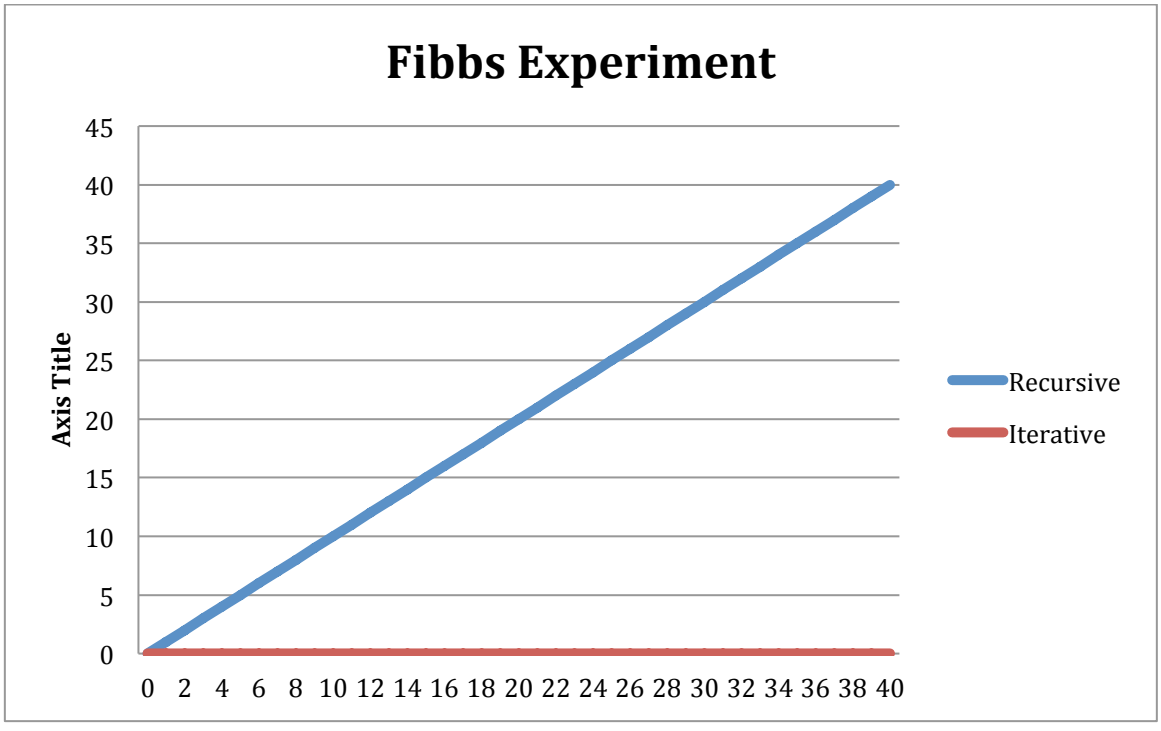
Hanoi Table

disks	RecurseTime	IterateTime
3	12653	0
4	12075	1
5	12072	2
6	11880	6
7	12608	9
8	12761	13
9	12628	26
10	11458	40
11	12167	51
12	12274	43
13	12851	72
14	11680	412
15	12111	231
16	12875	577
17	12307	926
18	12502	2128
19	12090	4296
20	11862	8437

Fibbs Table

Trials	Rtime	ITime
0	556	0
1	563	0
2	556	0
3	544	0
4	541	0
5	536	0
6	546	0
7	543	0
8	539	0
9	542	0
10	543	0
11	542	0
12	541	0
13	548	0
14	549	0
15	543	0
16	539	0
17	545	0
18	539	0
19	545	0
20	547	0
21	545	0
22	543	0
23	541	0
24	542	0
25	547	0
26	548	0
27	537	0
28	551	0
29	543	0
30	544	0
31	545	0
32	548	0
33	556	0
34	552	0
35	561	0
36	549	0
37	556	0
38	559	0
39	578	0
40	561	0


Hanoi Experiment
14000
12000
10000
8000
6000
4000
2000
0
Time
3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


Fibbs Experiment
45
40
35
30
25
20
15
10
5
0
Axis Title
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
Recursive
Iterative

Addendum

Q2) example. Calculate n! (n factorial)

```
//recursion,
static int Recursive(int n)
{
   if (n <= 1) return 1;
   return n * Recursive(n - 1);
}

//iterative, loop
static int Iterative(int n)
{
   int sum = 1;
   if (n <= 1) return sum;
   while (n > 1)
   {
      sum *= n;
      n--;
   }
   return sum;
}
```

Q3)

```
public int power(int x, int y){

        if (y == 0) {
                return 1;
                }

        return x*power(x, y-1);
}
```

Q6) Code:

Resources and Code modified from

Hanoi Iterative: (Rochester Institute of Technology)
http://www.cs.rit.edu/~jmg/courses/cs2/20012/code/hanoi/HanoiIterative.java

```
import java.io.*;
import java.lang.*;
```

```
public class hanoi3 // iterative hanoi. corners : 0, 1 and 2. Discs 0,1, .. N-1
{

	static public void main(String args[])
	{


		//output to file, Experiment2n!
		PrintWriter out = null;
		try {
			out = new PrintWriter(new FileWriter("IterativeHanoi.txt"));
		} catch (IOException e) {
			System.err.println("unable to output to file");

		}


		int disks = 20; // number of discs, will do each trial from n disks to 3
disks. n-3 trials

		for(int counter = 3; counter <= disks; counter++){

			long startTime = System.currentTimeMillis();

			solveHanoi(counter);

			long runTime = System.currentTimeMillis() - startTime;

			out.println("Iterative Hanoi: " + counter + ", " +runTime);

		}
		//close stream
		out.close();

	}


	public static int solveHanoi(int N){



		int nummoves,second=0,third,pos2,pos3,j,i = 1;

		int [] locations = new int[N+2]; // remembers which corner each disc
is on
		for (j=0; j<N; j++) locations[i] = 0; // initially all are on 0
```

```
		locations[N+1]=2; // 2 is destination

		nummoves = 1;
		for (i=1; i<=N; i++) nummoves*=2;
		nummoves -= 1;

		for (i=1; i<= nummoves; i++)
		{
			if (i%2==1)
			{
				// odd numbered move - move disc 1
				second = locations[1]; // remember where disc 1
moved from
				locations[1] = (locations[1]+ 1) %3;
				System.out.print("Move disc 1 to ");
				System.out.println((char)('A'+locations[1]));
			}
			else
			{
				// even numbered move make only move possible not
involving disc 1
				third = 3 - second - locations[1];

				// find smallest values on the other 2 corners
				pos2 = N+1; for (j=N+1; j>=2; j--) if
(locations[j]==second) pos2=j;
				pos3 = N+1; for (j=N+1; j>=2; j--) if (locations[j]==third)
pos3=j;

				System.out.print("Move disc ");

				// move smaller on top of larger
				if (pos2<pos3)
				{
					System.out.print(pos2);
					System.out.print(" to ");
					System.out.println((char)('A'+third));
					locations[pos2]=third;
				}
				else
				{
					System.out.print(pos3);
					System.out.print(" to ");
					System.out.println((char)('A'+second));
					locations[pos3]=second;
				}
```

```
                        }
                }
                return 0;

        }

}
```

Hanoi Recursive: from text

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

/**
 *SolveTowers useres recursion to solve the towers of Hanoi Puzzle
 * @author Java Foundations
 * @version 4.0
 */

public class SolveHanois
{
        /**
         * Creates a TowersOfHanoi puzzle and solves it.
         */
        public static void main(String[] args)
        {

                //output to file, Experiment2n!
                PrintWriter out = null;
                try {
                        out = new PrintWriter(new
FileWriter("RecursiveHanoiExperiment.txt"));
                } catch (IOException e) {
                        System.err.println("unable to output to file");

                }


                int disks = 20;

                for(int counter = 3; counter <= disks; counter++){

                        TowersOfHanoi towers = new TowersOfHanoi(disks);

                        long startTime = System.currentTimeMillis();
```

```
                towers.solve();

                long runTime = System.currentTimeMillis() - startTime;

                out.println("Recursive Hanoi: " + counter + ", " +runTime);

        }
                //close stream
                out.close();

}
}
/**Hanoi represents the classic Towers of Hanoi puzzle.
 *
 * @author Java Foundations
 * @version 4.0
 */

public class TowersOfHanoi
{
        private int totalDisks;
        /**
         * Sets up the puzzle with the specified number of disks.
         *
         * @param disks the number of disks
         */
        public TowersOfHanoi(int disks)
        {
                totalDisks = disks;
        }
        /**
         * Performs the initial call to moveTower to solve the puzzle.
         * Moves the disks from tower 1 to tower 3 using tower 2.
         */
        public void solve()
        {
                moveTower(totalDisks, 1, 3, 2);
        }
        /**
         * Moves the specified number of disks from one tower to another
         * by moving a subtower of n-1 disks out of the way, moving one
         * disk, then moving the subtower back. Base case of 1 disk.
         *
         * @param numDisks the number of disks to move
```

```
	 * @param start the starting tower
	 * @param end the ending tower
	 * @param temp the temporary tower
	 */
	private void moveTower(int numDisks, int start, int end, int temp)
	{

		if (numDisks == 1)
			moveOneDisk(start, end);
		else
		{
			moveTower(numDisks-1, start, temp, end);
			moveOneDisk(start, end);
			moveTower(numDisks-1, temp, end, start);
		}
	}

	/**
	 * Prints instructions to move one disk from the specified start
	 * tower to the specified end tower.
	 *
	 * @param start the starting tower
	 * @param end the ending tower
	 */
	private void moveOneDisk(int start, int end)
	{
		System.out.println("Move one disk from " + start + " to " + end);
	}
}
```

Fibonacci numbers: from Khan Academy
https://www.khanacademy.org/science/computer-science-subject/computer-science/v/recursive-fibonacci-example

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;


public class ExperimentFibs {



	public static void main(String[] args)
	 {
```

```
		//output to file, Experiment1Fibs
		PrintWriter out = null;

		try {
			out = new PrintWriter(new
FileWriter("Experiment1Fibs.txt"));
		} catch (IOException e) {
			System.err.println("unable to output to file");
		}


		//number of trials
		int trials = 40;

		//perform the each recursive trial
		for( int counter = trials; counter >= 0; counter--){


			long startTime = System.currentTimeMillis();

			int fib = Recurse(trials);

			long runTime = System.currentTimeMillis() - startTime;

			out.println("Recursive Fibs: " + counter + ", " +runTime);


	}

		//perform each iterative trial

		for(int counter = trials; counter >= 0; counter--){


			long startTime = System.currentTimeMillis();

			int fib = Iterate(trials);

			long runTime = System.currentTimeMillis() - startTime;

			out.println("Iterative Fibs: " + counter + ", " +runTime);

		}

		//close stream
				out.close();
```

```
 }

//-------------------------------------
//iteration method fibinocci

public static int Iterate(int n){
   if (n == 0) {
return 0;
   }

 if (n == 1){
 return 1;
    }

   int x = 0;
   int y= 1;
   int fibs = 0;

   for (int i = 2; i <= n; i++) {
      fibs = y + x;
      x= y;
      y = fibs;
   }
   return fibs;
}

//-------------------------------------
//recursion method fibinocci
public static int Recurse(int n){
   if (n == 0){
return 0;
}
   if (n == 1){
          return 1;
          }

   return Recurse(n - 1) + Recurse(n - 2);
}

//recursion, n!
public static int Recursive(int n)
{
   if (n <= 1) return 1;
   return n * Recursive(n - 1);
```

```
    }

    //iterative, loop n!
    public static int Iterative(int n)
    {
       int sum = 1;
       if (n <= 1) return sum;
       while (n > 1)
       {
          sum *= n;
          n--;
       }
       return sum;
    }


}
```