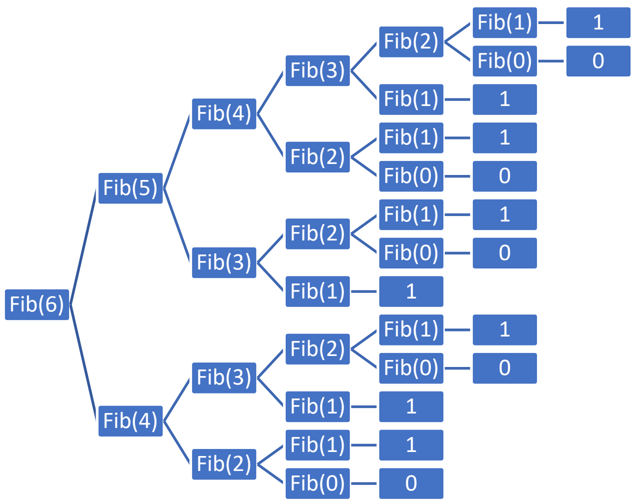
Amritpal Saini, 30039983

The recursive function performs redundant calculations, as shown below.

Let Fib(x) be the recursive function for calculating the Fibonacci of x.



Evaluation	Number of Evaluations
Fib(2)	5
Fib(3)	3
Fib(4)	2
Fib(5)	1
Fib(6)	1

The big-O running time for the iterative algorithm is $O(n)$ because, as seen in the Growth Rate analysis, the time units required to run the iterative algorithm grows linearly.

Line #	Function Loop	cost	time	Comments
16	public static long loop(int nMax) {	-	-	-
17	long low = 0;	C1=1	1	Assignment operation
18	long high = 1;	C2=1	1	Assignment operation
19	long ans = 0;	C3=1	1	Assignment operation
20	if (nMax == 0) {	C4=1	1	Relational operator
21	return 0;	C5=1	1	Returning from method
22	} else if (nMax == 1) {	C6=1	1	Relational operator
23	return 1;	C7=1	1	Returning from method
24	}else {	-	-	-
25	for (int i = 0; i <= nMax - 2; i++) {	C8=1 C9=1 C10=2	1 n-1 n-2	Assignment operator Relational operator (n+1 times but loop runs to n-2) Assignment and arithmetic operator
26	ans = low + high;	C11=2	n-2	Assignment and arithmetic operator
27	low = high;	C12=1	n-2	Assignment operator
28	high = ans;	C13=1	n-2	Assignment operator
29	}	-	-	-
30	}	-	-	-
31	return ans;	C14=1	1	Returning from method
32	}	-	-	-
7n+3	**Thus big O of O(n)**			

$$(C1 * 1) + (C2 * 1) + (C3 * 1) + (C4 * 1) + (C5 * 1) + (C6 * 1) + (C7 * 1) + (C8 * 1) + \big(C9 * (n-1)\big) + \big(C10 * (n-2)\big) + \big(C11 * (n-2)\big) + \big(C12 * (n-2)\big) + \big(C13 * (n-2)\big) + (C14 * 1) = 7n + 3$$

$$g(n) = (7+3)n = 10n$$

The big-O running time for the matrix algorithm is $O(log_2 n)$ because, as seen in the growth rate analysis, the time units required to run the matrix algorithm grows logarithmic.

Line #	Function Matrix	Cost	Time	Comments
1	If (n = 0) return 0;	C1 = 2	1	Relational Operator and returning to method
2	Initialize FM;	C2 = 12	1	2 * array accesses by index 1 * assignment operation
3	Call MatrixPower (n - 1);	C3 = 1	1	Method call
4	Return the element that is in the first column of the first row of FM;	C4 = 3	1	1 * return 2 * array access

$$f(n) = (C1 * 1) + (C2 * 1) + (C3 * 1) + (C4 * 1) = 18$$

Line #	Function MatrixPower	Cost	Time	Comments
1	If (n > 1)	C1 = 1	1	Relational operative
2	Call MatrixPower (n / 2)	C2 = 1	f(n/2)	Recursive method call
3	Update FM = FM * FM;	C3 = 56	1	10 * array access 3 * arithmetic operations 1 * assignment
4	If (n is odd)	C4 = 2	1	Relational and arithmetic
5	Update FM = FM * $\begin{matrix} 1 & 1 \\ 1 & 0 \end{matrix}$	C5 = 56	1	10 * array access 3 * arithmetic operations 1 * assignment

$$f(n) = (C1 * 1) + \left(C2 * \left(f\left(\frac{n}{2}\right)\right)\right) + (C3 * 1) + (C4 * 1) + (C5 * 1) = 1 + f\left(\frac{n}{2}\right) + 56 + 2 + 56$$

$$f(n) = 115 + f\left(\frac{n}{2}\right) \rightarrow f(n) = (115 * k) + f\left(\frac{n}{2^k}\right)$$

$$base\ case, let\ \frac{n}{2^k} = 1, thus\ n = 2^k\ which\ means\ k = log_2 n \rightarrow f(n) = 115 log_2 n + 1$$

Thus, f(n) (Matrix Power) + f(n) (Matrix) = $\mathbf{115 log_2 n + 19}$

$\boldsymbol{g(n)} = (115 + 18) log_2 n = \mathbf{134 log_2 n}$

Based on the plots, the recursive function is better from around Fibonacci(0) to Fibonacci(2). From around Fibonacci(3) to around Fibonacci(158), the iterative function is, on average, higher. From around Fibonacci(159) onwards, the matrix function is faster.

Appendix

Sousa, M., CPSC 319, asgmt-1—hints, 2019

Sousa, M., CPSC 319, 01 - Algorithm Design Patterns - (1) Recursion - soln-HW -- 1 sld-pp, 2019