Mobile: 916-206-2425

Programming Assignment 2

Nathalie Agustin

October 17, 2022

Algorithm Description:

The following algorithm takes in a recursion function and prints its corresponding recursion tree from depth 0 to depth 3. We extract the a, b, and nonrecursive cost of the input recursion functions using string slicing. This works if the given recursion functions are of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ or $T(n) = aT(n-b) + f(n)$ and f(n) is a polynomial function.

Function	T	(	n	)		=		a	T	(	n	/	b	)		+		f(n)
Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Let *a* equal the parameter from the recursion function which indicates how many children in the tree are created from each recursive call, and let *b* be the divisor of the problem size as we make recursive calls. For each depth, the recursive problem is further divided into sub problems of size $\frac{n}{b}$ or $n-b$. At each iteration, the problem size is passed into function $f(n)$ to get the cost of the recursive call. The total nonrecursive cost at each depth is calculated by summing up each tree nodes cost together.

(a) Algorithm Pseudocode:

Note: The *n* cost column refers to depth of the tree, **not** the size of the problem *T(n).*

	BUILD_TREE(r_function)	**Comments**	**Cost**
1	**if** r_function[8] is a number	// extract *a, b,* and the non-recursive cost (*f*)	c_1
2	a = int(r_function[8])	// when a is not equal to one	c_2
3	b = int(r_function[13])		c_3
4	f = r_function[18:]		c_4
5	**else**		c_5
6	a = 1	// when a is equal to one	c_6
7	b = int(r_function[12])		c_7
8	f = r_function[17:]		c_8
9	**if** "^" in f	// extracting the polynomial degree from *f(n)*	c_9
10	degree = r_function[4]	// assuming polynomial in form *cn^degree*	c_{10}
11	**else**		c_{11}
12	degree = 1		c_{12}
13	print "Depth of 0: [T(n) | " *f* "]"	// where *T(n)* is the first call followed by cost	c_{13}
14	**if** r_function[8:13] has "/"	// Check if divide and conquer algorithm or	c_{14}
15	DIVIDE_CONQUER(*a, b, f, degree)*	if chip and be conquered	$c_{15}a^n$
16	**else** CHIP_CONQUER(*a, b, f, degree*)		$c_{16}a^n$

	DIVIDE_CONQUER(*a, b, f, degree)*		
1	**for** *i* = 1 to 3	// prints recursion tree from depth 1 to 3	$3c_1$
2	*a_update = a^i*	// update *a* and *b* for each depth	c_2
3	*b_update = b^i*		c_3
4	**for** j in range *a_update*		c_4a^n
5	print "[T(n/" *b_update* ") |" *f* "(1/" *b_update* ")] "	// print nodes at depth *i*	c_5
6	*cost = a_update / b_update^degree*	// cost is sub problem size passed into *f(n)*	c_6
7	print "Total nonrecursive cost" *cost*		c_7
	CHIP_CONQUER(*a, b, f, degree)*		
1	**for** *i* = 1 to 3	// prints recursion tree from depth 1 to 3	$3c_1$
2	*b_update = -b*(i-1)-b*	// update *a* and *b* for each depth	c_2
3	*a_update = a^i*		c_3
4	**for** j in range *a_update*	// print nodes at depth *i*	c_4a^n
5	print "[T(n" *b_update* ") |" *f* "(" *b_update* ")] "		c_5
6	print "Total nonrecursive cost c(n" *b_update* ")^" *degree*	// cost is sub problem size passed into *f(n)*	c_6

(b) Worst-Case Running Time:

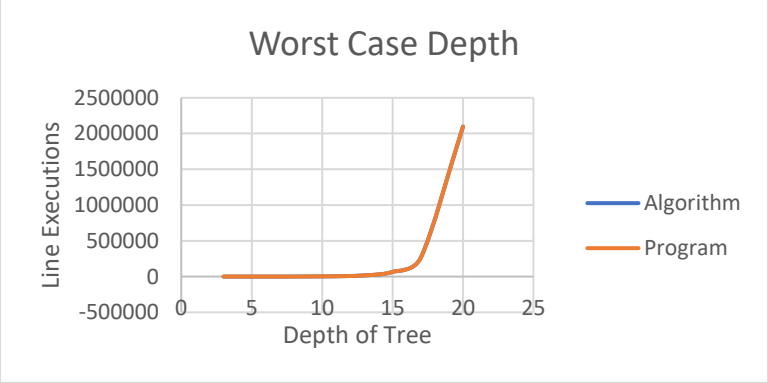
$c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 + c_9 + c_{10} + c_{11} + c_{12} + c_{13} + c_{14} + c_{15}a^n + c_{16}a^n = ca^n$

The worst-case running time is $\boxed{O(a^n),}$ where a = the number of sub-children in the tree created per parent at depth *n*. The complexity gets exponentially worse for large numbers of a at larger depths *n*. Complexity is dependent on the value of a depending on the provided recursion functions.

(c) The worst-case big-O asymptotic running time varies depending on the values for a in the recurrence function and the depth of the recursion tree. The worst-case running time is unaffected by different values of $f(n)$, the parameter b.

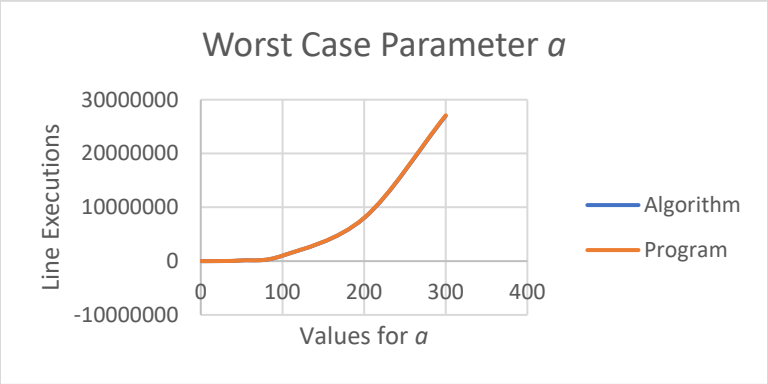
Analysis of Algorithm Running time to Program Behavior:

"Line Executions" in each graph is the key action in the algorithm and program that is executed the most. In the pseudocode, it would be Line 4 of DIVIDE_CONQUER and CHIP_CONQUER. The line executions can be found in the "Behavior Tests" folder of the assignment submission.



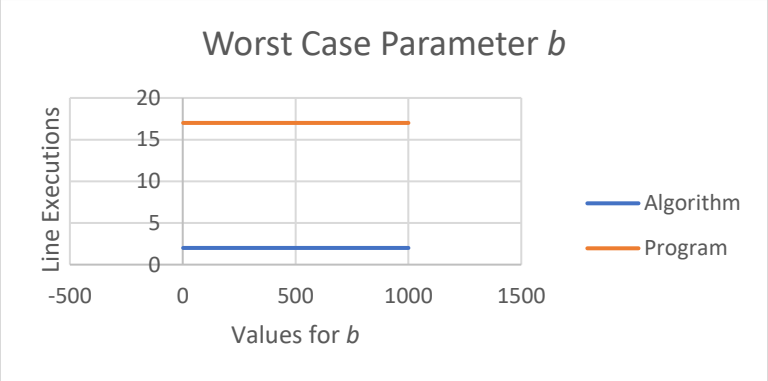
Tree Depth	Algorithm	Program
3	16	17
5	64	67
10	2048	2056
13	16384	16395
15	65536	65549
17	262144	262159
20	2097152	2097170

The algorithm and the program's running time both exponentially increase as we increase the depth of the recursion tree. This is intuitive, as child nodes are created exponentially at each successive depth level.



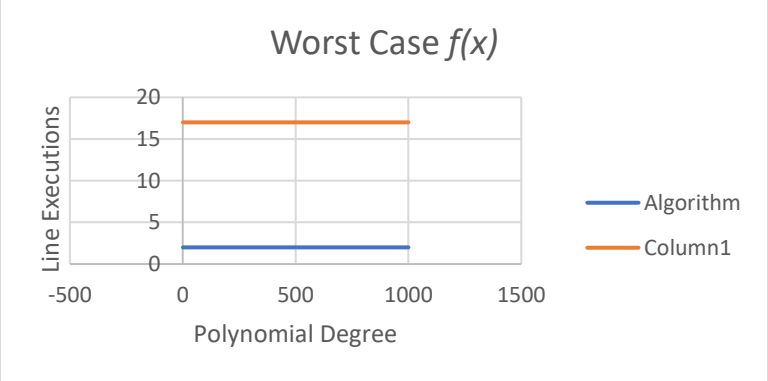
a	Algorithm	Program
1	1	6
10	1000	1113
50	125000	127553
100	1000000	1010103
200	8000000	8040203
300	27000000	27090303

The algorithm and the program's running time both increase by a^3. When we get to higher values of *a*, the number of line executions is increased by *a* to the power of three. This is because we are generating *a* subnodes for each node from depth 0 to depth 3.



b	Algorithm	Program
2	2	17
10	2	17
100	2	17
1000	2	17

Assuming $T(n) = aT(n/b) + f(n)$, let $a = 2$ and $f(n) = n$
If the estimated run time is $O(a^n)$, b has no affect on the runtime complexity. This is observed in the algorithm and the program execution.



f(n)	Algorithm	Program
n	2	17
n^{10}	2	17
n^{100}	2	17
n^{1000}	2	17

Assuming $T(n) = aT(n/b) + f(n)$, let $a, b = 2$ and $f(n) =$ some polynomial function
If the estimated run time is $O(a^n)$, nonrecurring cost $f(n)$ has no affect on the runtime complexity. This is observed in the algorithm and the program execution.