1. SEE HANDWRITTEN PAPER

2. **Explanation:** The recursive and iterative methods matched the pseudocode presented in lecture. For the matrix method, I implemented repeated squaring and stored the squared matrices of interest in an array and multiplied those which were needed to reach the nth Fibonacci number. The recursive method, naturally, slowed rapidly. But it's worth noting that the iterative method performed better than the matrix method until we reached a high threshold (also, the increased frequency of mod operations may have slowed the matrix method).

   My Benchmarked Results:

n	Recursive	Iterative	Matrix
2	0.000001	0.000001	0.000011
10	0.000003	0.000001	0.000017
30	0.021920	0.000001	0.000012
40	1.973016	0.000003	0.000011
50	240.048973	0.000001	0.000013
1000	Too Long	0.000027	0.000013
10000	Too Long	0.000272	0.000017
50000	Too Long	0.001136	0.000011

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include <stdint.h>
#include <vector>
#include <cmath>

//fib1

int fib1(int n) {
    if (n == 0) {
        return 0;
    }
    else if (n == 1) {
        return 1;
    }
    else {
        return (fib1(n-1)% 65536+fib1(n-2)% 65536)% 65536;
    }
}


//fib2

int fib2(int n) {
    int fibarray[n+1];
```

```
    fibarray[0] = 0;
    fibarray[1] = 1;

    for (int i = 2; i <= n; i++) {
        fibarray[i] = (fibarray[i-1]% 65536 + fibarray[i-2]% 65536)% 65536;
    }
    return fibarray[n];
}

//fib3

struct matrix_holder
{
    int matrix[2][2];
};

// ==============================================
// Implementation of naive matrix squaring
// resulting = mat1^2
// use pass by reference
// ==============================================

void naivematrixsquaring(matrix_holder input, matrix_holder &result) {
    result.matrix[0][0] = (input.matrix[0][0] % 65536 *input.matrix[0][0] % 65536 + input
    result.matrix[0][1] = (input.matrix[0][0] % 65536 *input.matrix[0][1] % 65536 + input
    result.matrix[1][0] = (input.matrix[1][0] % 65536 *input.matrix[0][0] % 65536 + input
    result.matrix[1][1] = (input.matrix[1][0] % 65536 *input.matrix[0][1] % 65536 + input
}

void naivematrixmult(matrix_holder input1, matrix_holder input2, matrix_holder &result) {
    result.matrix[0][0] = (input1.matrix[0][0]% 65536 *input2.matrix[0][0]% 65536 + input
    result.matrix[0][1] = (input1.matrix[0][0]% 65536 *input2.matrix[0][1]% 65536 + input
    result.matrix[1][0] = (input1.matrix[1][0]% 65536 *input2.matrix[0][0]% 65536 + input
    result.matrix[1][1] = (input1.matrix[1][0]% 65536 *input2.matrix[0][1]% 65536 + input
}


int fib3(int n) {

    // don't want to deal with these base cases

    if (n == 0) {
        return 0;
    }

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

```
    int arraysize = floor(log2(n)+2);

    matrix_holder base_matrix;
    matrix_holder final_matrix;
    matrix_holder identity_matrix;
    matrix_holder intermediary_matrix;

    // initialize matrices
    for  (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            base_matrix.matrix[i][j] = 0;
            final_matrix.matrix[i][j] = 0;
            identity_matrix.matrix[i][j] = 0;
        }
    }

    // fix the base matrix such that it is [0 1; 1 1]
    base_matrix.matrix[0][0] = 0;
    base_matrix.matrix[0][1] = 1;
    base_matrix.matrix[1][0] = 1;
    base_matrix.matrix[1][1] = 1;
    final_matrix.matrix[0][0] = 1;
    final_matrix.matrix[1][1] = 1;
    identity_matrix.matrix[0][0] = 1;
    identity_matrix.matrix[1][1] = 1;



    matrix_holder repeated_squares_arrray[arraysize];
    repeated_squares_arrray[0] = identity_matrix; // i.e., the identity (at this point)
    repeated_squares_arrray[1] = base_matrix;

    // populate all relevant repeating squares matrices
    for (int i = 2; i < arraysize; i++) {
        naivematrixsquaring(repeated_squares_arrray[i-1], repeated_squares_arrray[i]);
    }

    for (int i = 1; i < arraysize+1; i++) {
        // and with one and then bit shift for comparison and determining whether we use
        if ((n & 1) == 1) {
            naivematrixmult(final_matrix, repeated_squares_arrray[i], intermediary_matrix
            final_matrix = intermediary_matrix;
        }
        n = n >> 1;
    }
    return final_matrix.matrix[0][1]% 65536;// % 65536;
}
```

```
int main(int argc, char const *argv[])
{
    clock_t fib1_start_time, fib1_end_time, fib2_start_time, fib2_end_time, fib3_start_ti
    double fib1_cpu_time, fib2_cpu_time, fib3_cpu_time;

    int n = 50000;

    fib3_start_time = clock();
    int fib3ret = fib3(n);
    fib3_end_time = clock();
    fib3_cpu_time = ( (double) (fib3_end_time - fib3_start_time) ) / (double) CLOCKS_PER_
    printf("FIB3, %lf\n Fib3 returns: %i\n", fib3_cpu_time, fib3ret);

    fib2_start_time = clock();
    int fib2ret = fib2(n);
    fib2_end_time = clock();
    fib2_cpu_time = ( (double) (fib2_end_time - fib2_start_time) ) / (double) CLOCKS_PER_
    printf("FIB2, %lf\n Fib2 returns: %i\n", fib2_cpu_time, fib2ret);

    fib1_start_time = clock();
    int fib1ret = fib1(n);
    fib1_end_time = clock();
    fib1_cpu_time = ( (double) (fib1_end_time - fib1_start_time) ) / (double) CLOCKS_PER_
    printf("FIB1, %lf\n Fib1 returns: %i\n", fib1_cpu_time, fib1ret);

    return 0;
}
```

3.

A	B	O	o	Ω	ω	Θ
$\log n$	$\log(n^2)$	Y	N	Y	N	Y
$\log(n!)$	$\log(n^n)$	Y	N	Y	N	Y
$\sqrt[3]{n}$	$(\log n)^6$	N	N	Y	Y	N
$n^2 2^n$	3^n	Y	Y	N	N	N
$(n^2)!$	n^n	N	N	Y	Y	N
$\frac{n^2}{\log n}$	$n \log(n^2)$	N	N	Y	Y	N
$(\log n)^{\log n}$	$\frac{n}{\log(n)}$	N	N	Y	Y	N
$100n + \log n$	$(\log n)^3 + n$	Y	N	Y	N	Y

4. SEE HANDWRITTEN PROOFS

5. SEE HANDWRITTEN PROOFS

HW1

Dhilan Ramaprasad

Preface: I'm sorry for not LaTeX'ing this — didn't alot enough time to do so. I will typeset all future psets

For #1, I had a different solution initially, but at Professor M's office hours, he recommended changing to a more efficient solution — it's messier (sorry.)

1.1

6-sided biased die.

Take each side with probability, p_n.

We can generate unbiased coin flips (i.e., heads & tails)

by using the following scheme:

Roll die twice.

① If you roll a larger number then a smaller number, we will assign a coin flip of heads.

② If you roll a smaller number then a larger number, we will assign a coin flip of tails.

③ If you roll the same number twice, restart.

$$\underset{\text{large} \to \text{small}}{P(\text{heads})} = P_2 \cdot P_1 + P_3(P_1+P_2) + P_4(P_1+P_2+P_3) + P_5(P_1+P_2+P_3+P_4) + P_6(P_1+P_2+P_3+P_4+P_5)$$

which we can rewrite:

$$P(\text{heads}) = P_1P_2 + P_1P_3 + P_1P_4 + P_1P_5 + P_1P_6 + P_2P_3 + P_2P_4 + P_2P_5 + P_2P_6 + P_3P_4 + P_3P_5 + P_3P_6$$
$$+ P_4P_5 + P_4P_6 + P_5P_6$$
$$= P_1(P_2+P_3+P_4+P_5+P_6) + P_2(P_3+P_4+P_5+P_6) + P_3(P_4+P_5+P_6) + P_4(P_5+P_6) + P_5P_6$$

$$\underset{\text{small} \to \text{large}}{P(\text{tails})} = P_1(P_2+P_3+P_4+P_5+P_6) + P_2(P_3+P_4+P_5+P_6) + P_3(P_4+P_5+P_6) + P_4(P_5+P_6) + P_5P_6$$

$$\therefore \quad P(\text{heads}) = P(\text{tails}) \implies \text{unbiased coin flip.}$$

Expected # die rolls until fair (unbiased) coin flip:

$$= T \qquad T = \frac{\text{rolls per round}}{P(\text{heads}) + P(\text{tails})} = \frac{1}{P_1(P_2+P_3+P_4+P_5+P_6) + P_2(P_3+P_4+P_5+P_6) + P_3(P_4+P_5+P_6) + P_4(P_5+P_6) + P_5P_6}$$

1.2 6-sided biased die.

Take each side with probability $p_1 \cdots p_n$

We can generate an unbiased die roll by using the following scheme:

Roll the die three times:

① If any/all numbers repeat, roll again.

② If the numbers (i.e., of each of the 3 rolls) differ, assign as follows:

If you rolled the smallest of the three numbers first (we'll call this event S), then the middle of the three numbers second (we'll call this M), and finally the largest (L) of the three rolls last → assign die roll of #1

Here are the ensuing possible permutations by this method: (and assignments)

1st roll	2nd roll	3rd roll		
S	M	L	→	1
S	L	M	→	2
M	S	L	→	3
M	L	S	→	4
L	S	M	→	5
L	M	S	→	6

1.2 (contined)

Now we show that the probabilities of each permutation are equivalent. (I show $P(1) = P(2) = P(3)$, but I assert it holds for all of my resulting die rolls.)

$P(1) = P(S, M, L)$:

$$p_1 (p_2 (p_3 + p_4 + p_5 + p_6) + p_3 (p_4 + p_5 + p_6) + p_4 (p_5 + p_6) + p_5 p_6)$$
$$+ p_2 (p_3 (p_4 + p_5 + p_6) + p_4 (p_5 + p_6) + p_5 p_6)$$
$$+ p_3 (p_4 (p_5 + p_6) + p_5 p_6$$
$$+ p_4 (p_5 p_6)$$

$P(2) = P(S, L, M)$:

$$p_1 (p_3 p_2 + p_4 (p_2 + p_3) + p_5 (p_2 + p_3 + p_4) + p_6 (p_2 + p_3 + p_4 + p_5))$$
$$+ p_2 (p_4 p_3 + p_5 (p_3 + p_4) + p_6 (p_3 + p_4 + p_5))$$
$$+ p_3 (p_5 p_4 + p_6 (p_4 + p_5))$$
$$+ p_4 (p_6 p_5)$$

* when distributed / regrouped
$P(1) = P(2)$ → see appendix

$P(3) = P(M, S, L)$:

$$p_2 (p_1 (p_3 + p_4 + p_5 + p_6))$$
$$+ p_3 (p_1 (p_4 + p_5 + p_6) + p_2 (p_4 + p_5 + p_6))$$
$$+ p_4 (p_1 (p_5 + p_6) + p_2 (p_5 + p_6) + p_3 (p_5 + p_6))$$
$$+ p_5 (p_1 p_6 + p_2 p_6 + p_3 p_6 + p_4 p_6)$$

* when distributed / regrouped
$P(1) = P(2) = P(3)$ → see appendix

We take all permutations, then, as equally likely ∴ our scheme generates an unbiased die roll.

1.2 (continued, again)

Expected # $\underbrace{\text{biased die rolls}}$ until unbiased (fair) die roll:

$$= T$$

$$T = \frac{\text{rolls/round}}{P(SML) + P(SLM) + P(MSL) + P(MLS) + P(LSM) + P(LMS)} = \frac{3}{6\,P(SML)}$$

$$\boxed{T = \frac{1}{2\,P(SML)}}$$

for the sake of both my sanity, as well as the sanity of the grader, I will not substitute $P(SML)$, but it can be seen on the preceding page.

APPENDIX #1.2 expansion to prove equal probabilities.

SML expanded

$$p_1(p_2p_3 + p_2p_4 + p_2p_5 + p_2p_6 + p_3p_4 + p_3p_5 + p_3p_6 + p_4p_5 + p_4p_6 + p_5p_6)$$
$$+ p_2(p_3p_4 + p_3p_5 + p_3p_6 + p_4p_5 + p_4p_6 + p_5p_6)$$
$$+ p_3(p_4p_5 + p_4p_6 + p_5p_6)$$
$$+ p_4p_5p_6$$

SLM expanded

$$p_1(p_2p_3 + p_2p_4 + p_2p_5 + p_2p_6 + p_3p_4 + p_3p_5 + p_3p_6 + p_4p_5 + p_4p_6 + p_5p_6)$$
$$+ p_2(p_3p_4 + p_3p_5 + p_3p_6 + p_4p_5 + p_4p_6 + p_5p_6)$$
$$+ p_3(p_4p_5 + p_4p_6 + p_5p_6)$$
$$+ p_4p_5p_6$$

MSL expanded:

$$p_1(p_2p_3 + p_2p_4 + p_2p_5 + p_2p_6)$$
$$+ p_1(p_3p_4 + p_3p_5 + p_3p_6) + p_1(p_4p_5 + p_4p_6) + p_1(p_5p_6)$$
$$+ p_2(p_3p_4 + p_3p_5 + p_3p_6) + p_2(p_4p_5 + p_4p_6) + p_2(p_5p_6)$$
$$+ p_3(p_4p_5 + p_4p_6) + p_3(p_5p_6)$$
$$+ p_4p_5p_6$$

$\Longrightarrow$ same as prior 2

$\therefore$ equal probabilities

$\boxed{4}$

(1) $f_1(2n)$ is $O(f_1(n))$

$f_1(2n)$ is $O(f_1(n)) \iff \exists\, c, N$ s.t.

$f_1(2n) \leq c \cdot f_1(n) \quad \forall n \geq N.$

take $f_1 : n$

$f_1(n) = n$

$f_1(2n) = 2n$

$f_1(2n) \overset{?}{\leq} c \cdot f_1(n)$, take $c = 2$

$f_1(2n) = 2n \leq 2 \cdot f_1(n) = 2 \cdot n$ $\boxed{\checkmark}$

$\therefore$ by def'n $f_1(2n)$ is $O(f_1(n))$

for $f_1(x) = x$ $\square$

(2) $f_2(2n)$ is not $O(f_2(n))$ if there exists no c, N s.t. $f_2(2n) \leq c \cdot f_2(n) \quad \forall n \geq N$

take $f_2 : n^n$

$f_2(n) = n^n$

$f_2(2n) = (2n)^{2n} = (4n^2)^n$

We cannot satisfy $(4n^2)^n \leq c \cdot n^n \rightarrow \left(\frac{4n^2}{n}\right)^n \leq c$

$\rightarrow (4n)^n \leq c$

There is no c, N s.t. $(4n)^n \leq c \quad \forall n \geq N$

$\therefore \exists\, f_2(n)$ s.t. $f_2(2n)$ is not $O(f_2(n))$

4 (cont.)

(3) By def'n if $f(n)$ is $O(g(n))$,

$\exists\ c, N$ s.t. $f(n) \leq c \cdot g(n) \quad \forall\ n \geq N$

Similarly, if $g(n)$ is $O(h(n))$,

$\exists\ c_2, N_2$ s.t. $g(n) \leq c_2\, h(n) \quad \forall\ n \geq N_2$

We can take that multiplying both sides by a constant, c,

$$c \cdot g(n) \leq c \cdot c_2\, h(n) \qquad \forall\ n \geq N_2$$

does not change the inequality.

Given the previous and $f(n) \leq c \cdot g(n)$,

We can take

$$f(n) \leq c \cdot g(n) \leq c \cdot c_2\, h(n) \qquad \forall\ n \geq \max\{N, N_2\}$$

$\therefore \quad f(n) \leq C \cdot h(n)$, $C = c \cdot c_2$, $\forall\ n \geq N_{max}$

Which meets the def'n : $f(n)$ is $O(h(n))$

$\boxed{4 \text{ (cont.)}}$

(4) Counterexample

$\sqrt{n}$? $n^{\sin n}$

say $f : \sqrt{n}$

$g : n^{\sin(n)}$

these do not share a big-O relationship because $n^{\sin(n)}$, though bounded by 0 and n, it oscillates between the two.

(5) If f is $o(g(n))$, $\lim\limits_{n \to \infty} \frac{f(n)}{g(n)} = 0$

Given the formal def'n of a limit to ∞ we can say $\forall\ \varepsilon$, where $\varepsilon > 0$, $\exists$ $N > 0$ s.t.

$$\left| \frac{f(n)}{g(n)} \right| < \varepsilon \quad \text{for} \quad n > N$$

thereby $f(n) < \varepsilon g(n)$ for $n > N$

Which matches (though more strictly) the def'n for $f(n)$ is $O(g(n))$: $f(n) \leq c \cdot g(n)$ for some c, N and $\forall\ n > N$

and we can take $c = \varepsilon$. [We can arbitrarily choose ε (which is synonymous with c) so long as $\varepsilon > 0$.]

$\therefore$ $f(n)$ is $o(g(n)) \Rightarrow f(n)$ is $O(g(n))$. $\square$

$\boxed{5}$ By guess & check + graphing

we find exact sol'n

$$T(n) = \lceil \log_2 n \rceil n - 2^{\lceil \log_2 n \rceil} + 1$$

We prove a base case, using given $T(1)=0$, $T(2)=1$

$$T(2) = \lceil \log_2 2 \rceil (2) - 2^{\lceil \log_2 2 \rceil} + 1$$

$$= 2 - 2 + 1 = 1$$ which matches

sol'n using recurrence.

Now that we've established a base case

lets assume our formula holds true

for all $n \leq N$

Let's show that our formula holds true

for $N+1$

case 1 : assume $N+1$ is a power of 2

i.e., $N+1 = 2^k$

because we assume our sol'n holds true $n \leq N$

$T\left(\frac{N+1}{2}\right)$ using our formula also holds true.

Then: $T(N+1) = T\left(\left\lceil \frac{N+1}{2} \right\rceil\right) + T\left(\left\lfloor \frac{N+1}{2} \right\rfloor\right) + (N+1) - 1$

because $N+1$ is a power of 2, it's evenly divisible by 2, so we can drop the $\lceil\,\rceil$ and $\lfloor\,\rfloor$, and since $T\left(\frac{N+1}{2}\right)$ holds true given our induction hypothesis, we can substitute:

$\lceil \log_2 (N+1) \rceil \equiv k$

$$
\begin{aligned}
T(N+1) &= 2T\left(\frac{N+1}{2}\right) + (N+1) - 1 \\
&= 2\left(\left\lceil \log_2\left(\frac{N+1}{2}\right)\right\rceil \left(\frac{N+1}{2}\right) - 2^{\left\lceil \log_2\left(\frac{N+1}{2}\right)\right\rceil} + 1\right) + N \\
&= 2\left(\lceil \log_2 2^{k-1} \rceil \left(\frac{N+1}{2}\right) - 2^{\lceil \log_2 2^{k-1} \rceil} + 1\right) + N \\
&= 2\left[(k-1)\left(\frac{N+1}{2}\right) - 2^{k-1} + 1\right] + N \\
&= (k-1)(N+1) \underbrace{- 2 \cdot 2^{k-1}} + 2 + N \\
&= (N+1)k \cancel{-(N+1)} - 2^k \cancel{+N} + \cancel{2}\, 1 \\
&= k(N+1) - 2^k + 1 = \lceil \log_2 (N+1) \rceil (N+1) - 2^{\lceil \log_2 (N+1) \rceil} + 1
\end{aligned}
$$

Case 2 : assume $N+1$ even — i.e., $N+1 = 2j$ — but not a power of 2

$$T(N+1) = T\left(\left\lceil \tfrac{2j}{2} \right\rceil\right) + T\left(\left\lfloor \tfrac{2j}{2} \right\rfloor\right) + (N+1) - 1$$

$$= 2T(j) + N + 1 - 1$$

$$= 2T\left(\tfrac{N+1}{2}\right) + N$$

b/c Ind. Hyp assumes true $\forall n \leq N$ and $\frac{N+1}{2} < N$ we subst.

$$= 2\left(\left\lceil \log_2\left(\tfrac{N+1}{2}\right)\right\rceil\right)\left(\tfrac{N+1}{2}\right) - 2^{\lceil \log_2 (\frac{N+1}{2}) \rceil} + 1\Big) + N$$

$$= 2\left(\lceil \log_2(N+1) - \underbrace{\cancel{\log_2 2}}_{-1} \rceil \left(\tfrac{N+1}{2}\right. \ldots\right.$$

$$= 2\left[\left(\lceil \log_2(N+1)\rceil - 1\right)\left(\tfrac{N+1}{2}\right) - 2^{(\lceil \log_2(N+1)\rceil - 1)} + 1\right] + N$$

$$= \left(\lceil \log_2(N+1)\rceil - 1\right)(N+1) - \cancel{2} \cdot 2^{(\lceil \log_2(N+1)\rceil \cancel{-1})} + 2 + N$$

$$= \lceil \log_2(N+1)\rceil (N+1) \cancel{-N} - 1 - 2^{\lceil \log_2(N+1)\rceil} + 2 \cancel{+N}$$

$$= \underline{\lceil \log_2(N+1)\rceil (N+1) - 2^{\lceil \log_2(N+1)\rceil} + 1}$$

this equals what we expect from our original form

what we wanted to show

$$\lceil \log_2(N+1)\rceil (N+1) - 2^{\lceil \log_2(N+1)\rceil} + 1$$

✓

Case 3

let's once more assume that our formula holds true $\forall\, n \leq N$ and we hope to show that the sol'n holds for $N+1$ where $N+1$ is odd

i.e., $N+1 = 2J+1 \Rightarrow N = 2J$, $J \in \mathbb{Z}$

let us also take $\exists\, k$ s.t. $2^{k-1} < N+1 < 2^k$

non inclusive, again, because $(N+1)$ is odd.

Using the RECURRENCE GEN. FORM:

$$T(N+1) = T\left(\left\lceil \tfrac{N+1}{2} \right\rceil\right) + T\left(\left\lfloor \tfrac{N+1}{2} \right\rfloor\right) + (N+1) - 1$$

$$= T\left(\left\lceil \tfrac{2J+1}{2} \right\rceil\right) + T\left(\left\lfloor \tfrac{2J+1}{2} \right\rfloor\right) + (N+1) - 1$$

$$= T\left(\left\lceil J + \tfrac{1}{2} \right\rceil\right) + T\left(\left\lfloor J + \tfrac{1}{2} \right\rfloor\right) + (N+1) - 1$$

$$= T(J+1) + T(J) + (N+1) - 1$$

because we take (via Ind. Hyp) our sol'n as true $\forall\, n \leq N$ and $J+1 = \frac{N}{2} + 1 < N$ and $J = \frac{N}{2} < N$, we can use our found sol'n taken as true for $T(J+1)$ and $T(J)$, substituting:

$$= \lceil \log_2 (J+1) \rceil (J+1) - 2^{\lceil \log_2 (J+1) \rceil} + 1 + \lceil \log_2 J \rceil (J) - 2^{\lceil \log_2 J \rceil} + 1 + N + 1 - 1$$

$$= \left\lceil \log_2 \left(\tfrac{N}{2} + 1\right) \right\rceil \left(\tfrac{N}{2} + 1\right) - 2^{\lceil \log_2 (\frac{N}{2} + 1) \rceil} + 1 + \left\lceil \log_2 \tfrac{N}{2} \right\rceil \left(\tfrac{N}{2}\right) - 2^{\lceil \log_2 \frac{N}{2} \rceil} + 1 + N$$

$$= \left(\lceil \log_2 (N+2) \rceil - 1\right)\left(\tfrac{N+2}{2}\right) - 2^{(\lceil \log_2 (N+2) \rceil - 1)} + 1 + \left(\lceil \log_2 N \rceil - 1\right)\left(\tfrac{N}{2}\right) - 2^{\lceil \log_2 N \rceil - 1} + 1 + N$$

Case 3 (cont.)

Now, given our initial bounds

$2^{k-1} < N+1 < 2^k$, $N+2$ is always $\leq 2^k$

$\therefore \lceil \log_2 (N+2) \rceil \equiv k$

but $N \geq 2^{k-1}$, so $\lceil \log_2 N \rceil = \{k-1, k\}$

let's take the case where $N > 2^{k-1} \Rightarrow \lceil \log_2 N \rceil \equiv k$

returning to eqn:

$$T(N+1) = (k-1)\left(\frac{N+2}{2}\right) - 2^{k-1} + 1 + (k-1)\left(\frac{N}{2}\right) - 2^{k-1} + 1 + N$$

$$= (k-1)\left(\frac{N}{2} + \frac{N}{2} + 1\right) - 2 \cdot 2^{k-1} + 2 + N$$

$$= (k-1)(N+1) - 2^k + 2 + N$$

$$= k(N+1) \cancel{-N} - 1 - 2^k + 2 \cancel{+N} = k(N+1) - 2^k + 1$$

Returning to the stipulation made @ start ($2^{k-1} < N+1 < 2^k$)

we can say $k \equiv \lceil \log_2 (N+1) \rceil$

$$\therefore \quad T(N+1) = \lceil \log_2 (N+1) \rceil (N+1) - 2^{\lceil \log_2 (N+1) \rceil} + 1$$

✓

case 3 (cont.)

finally, let's take the case where $N = 2^{k-1}$ (but $N+2$ still $\leq 2^k$):

$$\lceil \log_2 (N+2) \rceil \equiv k \quad , \quad \lceil \log_2 N \rceil \equiv k-1$$

returning to eqn.

$$T(N+1) = (k-1)\left(\frac{N+2}{2}\right) - 2^{k-1} + 1 + (k-1-1)\left(\frac{N}{2}\right) - 2^{k-1-1} + 1 + N$$

Some algebraic manipulation shown on next page.

✓

$$(k-1)\left(\frac{N+2}{2}\right)-2^{k-1}+1+(k-1-1)\left(\frac{N}{2}\right)-2^{k-1-1}+1+N$$

$f = k-1$

$$f\left(\frac{N+2}{2}\right)-2^{f}+1+(f-1)\left(\frac{N}{2}\right)-2^{f-1}+1+N$$

$$f\frac{N}{2}+f-2^{f}+1+f\frac{N}{2}-\frac{N}{2}-\cancel{2^{f-1}}+1+N$$

$$-2^{f}\cdot 2^{-1}$$

$$fN+f-2^{f}\left(1+\frac{1}{2}\right)+1+1+\frac{N}{2}$$

$$(k-1)(N+1)-2^{k-1}\left(\frac{3}{2}\right)+2+\frac{N}{2}$$

$$(N+1)k-N-1-2^{k-2}\cdot 3+2+\frac{N}{2}$$

$$(N+1)k-\frac{N}{2}+1-2^{k-2}\cdot 3$$

$$(N+1)k-\frac{2^{k-1}}{2}+1-2^{k-1}\left(\frac{3}{2}\right)$$

$*\ N = 2^{k-1}$ in this case so I substitute.

$$(N+1)k-2^{k}\cancel{\left(\frac{1}{4}+\frac{3}{4}\right)}+1$$

$$(N+1)k-2^{k}+1$$

given original stipulation $2^{k-1} < N+1 < 2^{k}$

$$\lceil \log_2(N+1)\rceil \equiv k$$

so we can substitute : $\lceil \log_2(N+1)\rceil (N+1)-2^{\lceil \log_2(N+1)\rceil}+1$ as desired $\square$