

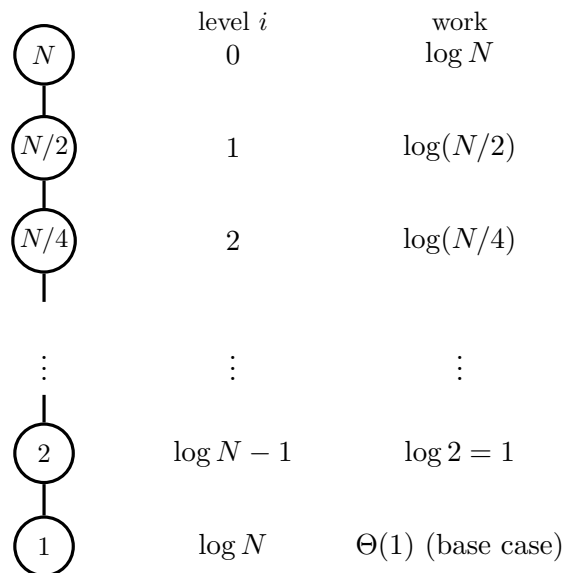
1 Welcome to Recursive City *Optional: (Challenge)*

- (a) Umbrella Corp's bio-containment system uses the following recursive function to monitor and isolate viral spread across N sectors. The system continuously divides the infected zone in half until containment is complete.

Give the tightest possible Big- Θ runtime bound for `containOutbreak(N)` as a function of N . Show your work drawing a recursion tree.

```
int containOutbreak(int N) {
    if (N <= 1) {
        return 1;
    }
    int M = N;
    while (M >= 1) {
        M = M / 2;
    }
    return containOutbreak(N / 2);
}
```

Solution: Each call makes **one** recursive call on $N/2$, so the recursion tree is a single chain. Since N is halved at every level, the tree has height $\log N$, which gives $\log N + 1$ levels: level $i = 0$ is the root and level $i = \log N$ is the base case.



At level i , the input is $N/2^i$, and the **while** loop halves M until it reaches 0. That

takes about $\log(N/2^i)$ iterations, so the total work is

$$\begin{aligned}
 \sum_{i=0}^{\log N} \log \frac{N}{2^i} &= \sum_{i=0}^{\log N} (\log N - \log 2^i) && \text{quotient rule: } \log \frac{a}{b} = \log a - \log b \\
 &= \sum_{i=0}^{\log N} \log N - \sum_{i=0}^{\log N} \log 2^i \\
 &= \sum_{i=0}^{\log N} \log N - \sum_{i=0}^{\log N} i \log 2 && \text{power rule: } \log a^k = k \log a \\
 &= \sum_{i=0}^{\log N} \log N - \log 2 \sum_{i=0}^{\log N} i
 \end{aligned}$$

Our logs are base 2, so $\log 2 = 1$. Now we evaluate each sum.

First sum: i runs from 0 to $\log N$, so we add up $\log N + 1$ copies of $\log N$ (one per level):

$$\sum_{i=0}^{\log N} \log N = \log N (\log N + 1) = (\log N)^2 + \log N$$

Second sum: this is an arithmetic sum, so we use $1 + 2 + \dots + k = \frac{k(k+1)}{2}$ with $k = \log N$:

$$\sum_{i=0}^{\log N} i = \frac{\log N (\log N + 1)}{2} = \frac{(\log N)^2 + \log N}{2}$$

Combining:

$$\begin{aligned}
 [(\log N)^2 + \log N] - 1 \cdot \frac{(\log N)^2 + \log N}{2} &= (\log N)^2 - \frac{1}{2}(\log N)^2 + \log N - \frac{1}{2} \log N \\
 &= \frac{1}{2}(\log N)^2 + \frac{1}{2} \log N
 \end{aligned}$$

Let $X = (\log N)^2$. The leftover $\frac{1}{2} \log N$ term is lower order, so it doesn't contribute asymptotically:

$$X - \frac{1}{2}X = \frac{1}{2}X \in \Theta(\frac{1}{2}X) = \Theta(X) = \boxed{\Theta((\log N)^2)}$$

Note: precisely, the **while** loop runs $\lfloor \log(N/2^i) \rfloor + 1$ times, and the base case does 1 unit of work. These extra +1's add only $\Theta(\log N)$ in total, so they don't change the bound.

Alternative solution. Using the quotient rule up front, each level's work simplifies to $\log \frac{N}{2^i} = \log N - i$. So the levels do $\log N$, $\log N - 1$, $\log N - 2$, ..., 1 units of work, and the base case at level $\log N$ does constant work. Reading the tree from the bottom up, this is just an arithmetic sum:

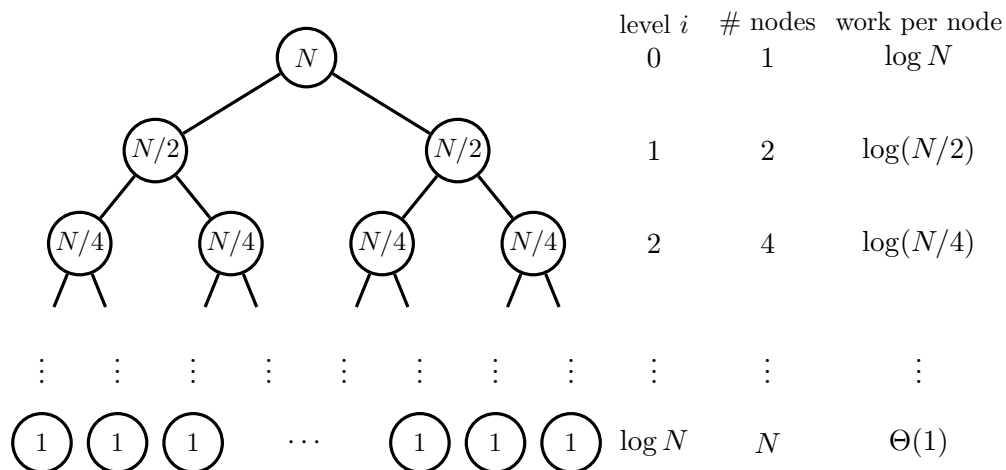
$$\sum_{i=0}^{\log N} (\log N - i) = 0 + 1 + 2 + \dots + \log N = \frac{\log N (\log N + 1)}{2} \in \Theta((\log N)^2)$$

2. Bryan Hodukavich, an organ delivery driver for Lightfoot Medical Transport has just breached Raccoon City General. Inside, Umbrella Corporation's bio-containment network is executing a binary search and quarantine routine. However, the G-Virus has mutated, causing the system to bifurcate its diagnostic scanning into two parallel recursive sub-tasks at every step.

```
int gVirusOutbreak(int N) {
    if (N <= 1) {
        return 1;
    }
    int M = N;
    while (M >= 1) {
        M = M / 2;
    }
    return gVirusOutbreak(N / 2) + gVirusOutbreak(N / 2);
}
```

- (a) Find the tightest possible Big- Θ bound for `gVirusOutbreak(N)` as a function of N . You may show your work using a recursion tree diagram. Final steps for this runtime analysis involve simplifying a common yet messy summation that appears in math, CS, and courses in probability. A change-of-index, infinite geometric series, algebraic manipulation, and/or calculus may help with this final part.

Solution: Each call makes **two** recursive calls on $N/2$, so the number of nodes doubles at every level. The tree has height $\log N$: level $i = 0$ is the root and level $i = \log N$ is the leaf (base case) level, which has $2^{\log N} = N$ nodes.



The work done by a single node depends on whether it is an internal node or a leaf:

$$\text{work per node at level } i = \begin{cases} \log \frac{N}{2^i} & \text{if } i < \log N \quad (\text{internal node}) \\ \Theta(1) & \text{if } i = \log N \quad (\text{leaf, base case}) \end{cases}$$

Notice that plugging $i = \log N$ into the internal-node formula gives $\log \frac{N}{2^{\log N}} = \log 1 = 0$. That formula counts **while** loop iterations, but a leaf never reaches the loop: it hits the base case and returns. So the leaves still do $\Theta(1)$ work each, and we count them separately.

Summing over the levels, with 2^i nodes at level i :

$$\underbrace{\sum_{i=0}^{\log N-1} 2^i \log \frac{N}{2^i}}_{\text{internal nodes (while loop)}} + \underbrace{2^{\log N} \cdot \Theta(1)}_{\text{leaves (base case)}} = \sum_{i=0}^{\log N-1} 2^i \log \frac{N}{2^i} + \Theta(N)$$

Strategy: Instead of computing the internal-node sum exactly, we trap it between an **upper bound** and a **lower bound**. If both bounds grow like the same function of N , that function is our Big- Θ bound. We start with the upper bound.

Upper bound. First, we unravel the $\log \frac{N}{2^i}$ term using log properties. Then we let $X = \log N$, so the upper index $\log N - 1$ becomes $X - 1$:

$$\begin{aligned} \sum_{i=0}^{\log N-1} 2^i \log \frac{N}{2^i} &= \sum_{i=0}^{\log N-1} 2^i (\log N - \log 2^i) && \text{quotient rule: } \log \frac{a}{b} = \log a - \log b \\ &= \sum_{i=0}^{X-1} 2^i (X - \log 2^i) && \text{let } X = \log N \\ &= \sum_{i=0}^{X-1} 2^i (X - i) && \text{power rule: } \log 2^i = i \log 2 = i \end{aligned}$$

Next, we change the index. Let $j = X - i$, which rewrites to $i = X - j$. Substituting into each term:

$$\begin{aligned} 2^i (X - i) &= 2^{X-j} (X - (X - j)) \\ &= 2^{X-j} \cdot j \\ &= 2^{\log N - j} \cdot j \\ &= \frac{2^{\log N}}{2^j} \cdot j && \text{exponent rule: } a^{m-n} = \frac{a^m}{a^n} \\ &= \frac{N}{2^j} \cdot j && 2^{\log N} = N \end{aligned}$$

The endpoints change too. Plugging each original endpoint into $j = X - i$:

$$i = 0 \implies j = X - 0 = X \qquad i = X - 1 \implies j = X - (X - 1) = 1$$

So j runs from X down to 1. Since addition is commutative, the order we add the terms in doesn't matter, so we can swap the endpoints and put $j = 1$ at the bottom of the summation:

$$\sum_{i=0}^{X-1} 2^i (X - i) = \sum_{j=1}^X \frac{N}{2^j} \cdot j = N \sum_{j=1}^{\log N} \frac{j}{2^j}$$

We can upper bound $\sum \frac{j}{2^j}$ by letting j run all the way from 1 to ∞ . This helps us because adding additional positive terms to a sum only increases the sum:

$$\sum_{j=1}^{\log N} \frac{j}{2^j} \leq \sum_{j=1}^{\infty} \frac{j}{2^j} = M$$

where M is the resulting sum. Now we find M :

$$\begin{aligned} M &= \frac{1}{2^1} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \cdots \\ &= \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots \end{aligned}$$

How can we massage this to look like a toy geometric series that we're familiar with? Say,

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots$$

We know this toy series evaluates to $S = \frac{a_1}{1-r}$, where a_1 is the first term of the series and r is the common ratio, provided that $|r| < 1$. Here $a_1 = \frac{1}{2}$ and $r = \frac{1}{2}$:

$$S = \frac{a_1}{1-r} = \frac{\frac{1}{2}}{1-\frac{1}{2}} = \frac{\frac{1}{2}}{\frac{1}{2}} = 1$$

Now we rewrite M . If we take $\frac{1}{2} \cdot M$, every term is cut in half:

$$\frac{1}{2}M = \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \frac{4}{32} + \cdots$$

But $M - \frac{1}{2}M$ lines up like a classic subtraction problem, where we subtract terms with the same denominator:

$$\begin{array}{r} M = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \cdots \\ - \quad \frac{1}{2}M = \quad \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \cdots \\ \hline M - \frac{1}{2}M = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots \end{array}$$

This is a geometric series we're familiar with! With $a_1 = \frac{1}{2}$ and $r = \frac{1}{2}$:

$$M - \frac{1}{2}M = \frac{\frac{1}{2}}{1-\frac{1}{2}} = 1 \implies \frac{1}{2}M = 1 \implies M = 2$$

We now revisit the original summation with this new knowledge. We don't know the exact value of the finite sum, but it is at most the infinite sum $M = 2$:

$$N \sum_{j=1}^{\log N} \frac{j}{2^j} \leq N \sum_{j=1}^{\infty} \frac{j}{2^j} = N \cdot 2 = 2N$$

Finally, we return to summing over the levels, with 2^i nodes at level i . Calling the total work $T(N)$, the internal-node summation is now upper bounded by $2N$:

$$T(N) = \underbrace{\sum_{i=0}^{\log N - 1} 2^i \log \frac{N}{2^i}}_{\leq 2N} + \underbrace{2^{\log N} \cdot \Theta(1)}_{\Theta(N)} \leq 2N + \Theta(N)$$

So $T(N) \in O(N)$.

Lower bound. We're not finished (unfortunately) ☹. We need to find a lower bound now. Recall the summations we found by adding up all the work in the recursion tree:

$$T(N) = \underbrace{\sum_{i=0}^{\log N - 1} 2^i \log \frac{N}{2^i}}_{\text{internal nodes (while loop)}} + \underbrace{2^{\log N} \cdot \Theta(1)}_{\text{leaves (base case)}}$$

We focus on the first summation. Recall that we were able to massage it with a change of index, which lets us rewrite it as:

$$\sum_{i=0}^{\log N - 1} 2^i \log \frac{N}{2^i} = N \sum_{j=1}^X \frac{j}{2^j}, \quad \text{where } X = \log N$$

Now we find a lower bound for the factor representing the summation, $\sum_{j=1}^X \frac{j}{2^j}$:

$$\sum_{j=1}^X \frac{j}{2^j} \geq \sum_{j=1}^X \frac{1}{2^j}$$

How does this effectively create a lower bound for us to work with? Every j is at least 1, so $\frac{j}{2^j} \geq \frac{1}{2^j}$ for every term, and replacing each term with a smaller one can only make the sum smaller. We do this because in rigorous asymptotic analysis, we typically take advantage of lower and upper bounds, and here the lower bound leads us to a finite geometric series:

$$\sum_{j=1}^X \frac{j}{2^j} \geq \sum_{j=1}^X \frac{1}{2^j} = \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \cdots + \frac{1}{2^X} = Q$$

We'll apply the same trick we used to find the upper bound (recall $M - \frac{1}{2}M$), but now with respect to Q :

$$\begin{aligned} Q &= \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^X} \\ - \frac{1}{2}Q &= \frac{1}{2^2} + \frac{1}{2^3} + \cdots + \frac{1}{2^X} + \frac{1}{2^{X+1}} \\ \hline Q - \frac{1}{2}Q &= \frac{1}{2} + 0 + 0 + \cdots + 0 - \frac{1}{2^{X+1}} \end{aligned}$$

Every middle term cancels, leaving only the first term of Q and the last term of $\frac{1}{2}Q$:

$$Q - \frac{1}{2}Q = \frac{1}{2} - \frac{1}{2^{X+1}} \implies \frac{1}{2}Q = \frac{1}{2} - \frac{1}{2^{X+1}} \implies Q = 1 - \frac{2}{2^{X+1}} = 1 - \frac{1}{2^X}$$

So:

$$\sum_{j=1}^X \frac{j}{2^j} \geq \sum_{j=1}^X \frac{1}{2^j} = 1 - \frac{1}{2^X}, \quad \text{where } X = \log N$$

Since $2^X = 2^{\log N} = N$, the right-hand side equals $1 - \frac{1}{N}$. Whenever $N \geq 2$, we have $\frac{1}{N} \leq \frac{1}{2}$ (for example, $N = 2$ makes the right-hand side exactly $\frac{1}{2}$). Thus:

$$1 - \frac{1}{2^X} = 1 - \frac{1}{N} \geq 1 - \frac{1}{2} = \frac{1}{2} \quad \implies \quad \sum_{j=1}^{\log N} \frac{j}{2^j} \geq \frac{1}{2}, \quad \text{for } N \geq 2$$

Now we multiply both sides of the inequality by N . Since $N > 0$, the inequality doesn't flip:

$$N \sum_{j=1}^{\log N} \frac{j}{2^j} \geq \frac{N}{2}$$

Tying this back to $T(N)$:

$$T(N) = \underbrace{N \sum_{j=1}^{\log N} \frac{j}{2^j}}_{\geq \frac{N}{2}} + \underbrace{2^{\log N} \cdot \Theta(1)}_{\Theta(N)} \geq \frac{N}{2} + \Theta(N)$$

So $T(N) \in \Omega(N)$.

We've achieved a lower bound! Since the lower bound and the upper bound are in terms of the same function of N ,

$$\underbrace{\frac{N}{2} + \Theta(N)}_{\text{lower bound: } \Omega(N)} \leq T(N) \leq \underbrace{2N + \Theta(N)}_{\text{upper bound: } O(N)}$$

we have shown that $\text{gVirusOutbreak}(N) \in \boxed{\Theta(N)}$.

PHEW!

- (b) (Space Complexity) What is the tightest Big- Θ bound on the memory space (call stack) required by $\text{gVirusOutbreak}(N)$?

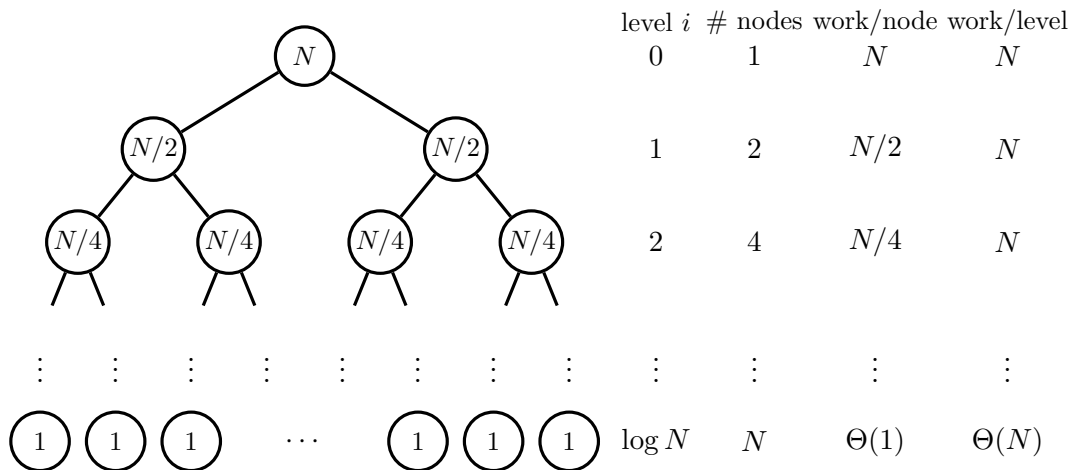
Solution: $\Theta(\log N)$. Even though the tree has about $2N$ nodes, they are never all on the call stack at once. The first call to $\text{gVirusOutbreak}(N / 2)$ runs to completion and returns before the second one starts, so the stack only ever holds one root-to-leaf path. That path has $\log N + 1$ frames, and each frame stores only a constant amount of data (N and M), so the space used is $\Theta(\log N)$.

3. As Bryan escapes Raccoon City General, Umbrella Corporation unleashes its Nemesis-T Type, a bio-weapon programmed to hunt down every surviving witness. Unlike the earlier containment routines, Nemesis leaves nothing to chance: before splitting its pursuit into two parallel search teams, it sweeps every single one of the N sectors in its current zone.

```
int nemesisProtocol(int N) {
    if (N <= 1) {
        return 1;
    }
    for (int i = 0; i < N; i++) {
        searchSector(i); // Assume searchSector() runs in  $\Theta(1)$ 
    }
    return nemesisProtocol(N / 2) + nemesisProtocol(N / 2);
}
```

- (a) Find the tightest possible Big- Θ bound for `nemesisProtocol(N)` as a function of N . You may show your work using a recursion tree diagram.

Solution: Each call makes **two** recursive calls on $N/2$, so level i has 2^i nodes, and the tree has height $\log N$. A node at level i has input $N/2^i$, so its `for` loop calls the $\Theta(1)$ `searchSector` method $N/2^i$ times.



At every internal level, the number of nodes doubles while the work per node halves, so each level does the same amount of work:

$$2^i \cdot \frac{N}{2^i} = N$$

The leaf level has $2^{\log N} = N$ base-case nodes doing $\Theta(1)$ work each, which is another $\Theta(N)$. Summing over the $\log N$ internal levels plus the leaves:

$$\underbrace{\sum_{i=0}^{\log N - 1} N}_{\text{internal nodes}} + \underbrace{\Theta(N)}_{\text{leaves}} = N \log N + \Theta(N) \in \boxed{\Theta(N \log N)}$$

Unlike `gVirusOutbreak`, no single level dominates here: every level does the same N work, so the total is (work per level) $\times$ (number of levels).

- (b) (Space Complexity) What is the tightest Big- Θ bound on the memory space (call stack) required by `nemesisProtocol(N)`? How does it compare to `gVirusOutbreak(N)`, and why?

Solution: $\Theta(\log N)$, the same as `gVirusOutbreak(N)`. The `for` loop changes how much *time* each call takes, but not how deep the recursion goes. As before, the first recursive call finishes before the second one starts, so the stack only ever holds one root-to-leaf path of $\log N + 1$ frames, each storing a constant amount of data (N and i).