Problem Set 4 Solution

Hyungmo Gu

April 14, 2020

Question 1

a. **Statement:** $\forall f, g : \mathbb{N} \to \mathbb{R}^+,\ b \in \mathbb{R}^+,\ (g(n) \in \Theta(f(n))) \wedge (n_0 \in \mathbb{N},\ n \geq n_0 \Rightarrow f(n) \geq b \wedge g(n) \geq b) \wedge (b > 1) \Rightarrow \log_b(g(n)) \in \Theta(\log_b(f(n)))$

Statement Expanded: $\forall f, g : \mathbb{N} \to \mathbb{R}^+,\ b \in \mathbb{R}^+,\ \Big(\exists c_1, c_2, n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)\Big) \wedge \Big(\exists n_1 \in \mathbb{N},\ n \geq n_1 \Rightarrow f(n) \geq b \wedge g(n) \geq b\Big) \wedge \Big(b > 1\Big) \Rightarrow \Big(\exists d_1, d_2, n_2 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_2 \Rightarrow d_1 \cdot \log_b(g(n)) \leq \log_b(f(n)) \leq d_2 \cdot \log_b(g(n))\Big)$

Proof. Let $f, g : \mathbb{N} \to \mathbb{R}^+$, and $b \in \mathbb{R}^+$. Assume $c_1 = 1$, $c_2 = b$, and $n_0 = 1$, and $n \in \mathbb{N}$ such that $n \geq n_0$ and $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$. Assume $f(n)$ and $g(n)$ are eventually $\geq b$. Assume $b > 1$. Let $d_1 = 1$, $d_2 = 2$, and $n_2 = n_0$. Assume $n \geq n_2$.

We need to show $d_1 \cdot \log_b g(n) \leq \log_b f(n) \leq d_2 \cdot \log_b g(n)$.

We will do so in two parts. One for $(d_1 \cdot \log_b g(n) \leq \log_b f(n))$ and the other for $(\log_b f(n) \leq d_2 \cdot \log_b g(n))$.

Part 1 ($d_1 \cdot \log_b g(n) \leq \log_b f(n)$):

The assumption tell us

$$c_1 \cdot g(n) \leq f(n) \tag{1}$$

Then, it follows from the fact $\forall x, y \in \mathbb{R}^+, x \geq y \Leftrightarrow \log x \geq \log y$

$$\log(c_1 \cdot g(n)) \leq \log(f(n)) \tag{2}$$

Then, using the fact $b > 1$, we can calculate

$$\frac{\log(c_1 \cdot g(n))}{\log b} \leq \frac{\log(f(n))}{\log b} \tag{3}$$
$$\frac{\log(c_1) + \log(g(n))}{\log b} \leq \frac{\log(f(n))}{\log b} \tag{4}$$

Then,

$$\frac{\log(g(n))}{\log b} \leq \frac{\log(f(n))}{\log b} \tag{5}$$

by the fact $c_1 = 1$ and $\log c_1 = 0$.

Then, since $\frac{\log f(x)}{\log b} = \log_b f(x)$,

$$\log_b(g(n)) \leq \log_b(f(n)) \tag{6}$$

Then, because we know $d_1 = 1$, we can conclude

$$\log_b(g(n)) \leq d_1 \cdot \log_b(f(n)) \tag{7}$$

Part 2 ($\log_b f(n) \leq d_2 \cdot \log_b g(n)$):

The assumption tells us

$$f(n) \leq c_2 \cdot g(n) \tag{8}$$

Then, it follows from the fact $\forall x, y \in \mathbb{R}^+, x \geq y \Leftrightarrow \log x \geq \log y$

$$\log(f(n)) \leq \log(c_2 \cdot g(n)) \tag{9}$$

Then, using the fact $b > 1$, we can calculate

$$\frac{\log(f(n))}{\log b} \leq \frac{\log(c_2 \cdot g(n))}{\log b} \tag{10}$$

$$\frac{\log(f(n))}{\log b} \leq \frac{\log(c_2) + \log(g(n))}{\log b} \tag{11}$$

Then, since $c_2 = b$,

$$\frac{\log(f(n))}{\log b} \leq \frac{\log(b) + \log(g(n))}{\log b} \tag{12}$$

Then, using the fact $g(n)$ is eventually $\geq b$, we can write

$$\frac{\log(f(n))}{\log b} \leq \frac{\log(g(n)) + \log(g(n))}{\log b} \tag{13}$$

$$\frac{\log(f(n))}{\log b} \leq \frac{2 \cdot \log(g(n))}{\log b} \tag{14}$$

Then, since $\frac{\log f(x)}{\log b} = \log_b f(x)$,

$$\log_b(f(n)) \leq 2 \cdot \log_b(g(n)) \tag{15}$$

Then, because we know $d_2 = 2$, we can conclude

$$\log_b(f(n)) \leq d_2 \cdot \log_b(g(n)) \tag{16}$$

$\square$

Notes:

- $\forall x, y \in \mathbb{R}^+, x \geq y \Leftrightarrow \log x \geq \log y$
- $\exists c_1, c_2, n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N}, n \geq n_0 \Rightarrow c_1 \cdot g(n) \leq f(n) \leq c2 \cdot g(n)$
- **Definition of Eventually:** $\exists n_0 \in \mathbb{N}, n \geq n_0 \Rightarrow P$, where $P : \mathbb{N} \to \{\text{True}, \text{False}\}$

b. *Proof.* Let $k \in \mathbb{N}$.

First, we will analyze the cost of loop 2 over iteration of loop 1.

The code tells us loop 2 starts at $j_k = 1$ with j_k increasing by a factor of 3 per iteration until $j_k \geq 1$.

Using these facts, we can calculate that the terminating condition occurs when

$$3^k \geq i \tag{1}$$
$$k \geq \log_3 i \tag{2}$$

Because we know the number of iterations is the smallest value of k satisfying the above inequality, we can conclude loop 2 has

$$\lceil \log_3 i \rceil \tag{3}$$

iterations.

Next, we need to determine the total number of iterations of loop 2 over all iterations of loop 1.

The code tells us loop 1 starts at $i = 1$ and ends at $i = n$ with each i increasing by 1 per iteration.

Using these facts, we can conclude loop 2 has total of

$$\lceil \log_3 1 \rceil + \lceil \log_3 2 \rceil + \cdots + \lceil \log_3 n \rceil = \sum_{i=1}^{n} \lceil \log_3 i \rceil \tag{4}$$

iterations. □

c. After scratching head and looking at solution many times, I realized that there are many things I do not yet understand, and it's the best to write what I have and learn from the solution. Here is my best attempt :).

Proof. Let $n \in \mathbb{N}$.

The previous answer tells us the exact cost of the algorithm is

$$\sum_{i=1}^{n} \lceil \log_3 i \rceil \tag{1}$$

Then, it follows by changing the variable i to $i' = \log_3 i$ we can write

$$\sum_{i'=0}^{\lceil \log_3 n \rceil} i' \tag{2}$$

Then, because we know $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$, we can conclude

$$\sum_{i'=0}^{\lceil \log_3 n \rceil} i' = \frac{(\lceil \log_3 n \rceil)(\lceil \log_3 n \rceil + 1)}{2} \tag{3}$$

$$= \frac{\lceil \log_3 n \rceil^2 + \lceil \log_3 n \rceil}{2} \tag{4}$$

Then, we can conclude the runtime of the algorithm is $\Theta(\log_3^2 n)$. □

Correct Solution:

We need to determne Θ of the algorithm.

We will prove that the Θ of the algorithm is $\Theta(n \log n)$.

The answer to previous question tells us the total exact cost of the algorithm is

$$\sum_{i=1}^{n} \lceil \log_3 i \rceil \tag{5}$$

Then, by using fact 1 $\forall x \in \mathbb{R}, x \leq \lceil x \rceil \leq x + 1$, we can calculate

$$\sum_{i=1}^{n} \log_3 i \leq \sum_{i=1}^{n} \lceil \log_3 i \rceil \leq \sum_{i=1}^{n} \Big(\log_3 i + 1 \Big) \tag{6}$$

$$\sum_{i=1}^{n} \log_3 i \leq \sum_{i=1}^{n} \lceil \log_3 i \rceil \leq \Big(\sum_{i=1}^{n} \log_3 i + \sum_{i=1}^{n} 1 \Big) \tag{7}$$

$$\sum_{i=1}^{n} \log_3 i \leq \sum_{i=1}^{n} \lceil \log_3 i \rceil \leq \sum_{i=1}^{n} \log_3 i + n \tag{8}$$

Then,

$$\log_3 \Big(\prod_{i=1}^{n} i \Big) \leq \sum_{i=1}^{n} \lceil \log_3 i \rceil \leq \log_3 \Big(\prod_{i=1}^{n} i \Big) + n \tag{9}$$

$$\log_3(n!) \leq \sum_{i=1}^{n} \lceil \log_3 i \rceil \leq \log_3(n!) + n \tag{10}$$

by the fact $\forall a, b \in \mathbb{R}^+, \log(a) + \log(b) = \log(ab)$.

Then,

$$\frac{\ln n!}{\ln 3} \leq \sum_{i=1}^{n} \lceil \log_3 i \rceil \leq \frac{\ln(n!)}{\ln 3} + n \tag{11}$$

by changing the base to e using the formula $\log_3 n! = \frac{\log_e n!}{\log_e 3} = \frac{\ln n!}{\ln 3}$.

Now, the fact 2 tells us $n! \in \Theta(e^{n \ln n - n + \frac{1}{2} \ln n})$.

Because we know from fact 3 that $n \ln n - n + \frac{1}{2} \ln n$ is eventually ≥ 1, we can conclude $e^{n \ln n - n \frac{1}{2} \ln n}$ is eventually $\geq e$.

Since $n!$ is also eventually $\geq e$, by using solution to problem 1.a with $g(n) = n!$ and $f(n) = e^{n \ln n - n + \frac{1}{2} \ln}$ and $b = e$, we can write

$$\ln(n!) \in \Theta(\ln(e^{n \ln n - n + \frac{1}{2} \ln n})) \tag{12}$$

$$\ln(n!) \in \Theta(n \ln n - n + \frac{1}{2} \ln n) \tag{13}$$

Then,

$$\ln(n!) \in \Theta(n \ln n) \tag{14}$$

by the fact $n \ln n - n + \frac{1}{2} \ln n \in \Theta(n \ln n)$.

So, since the algorithm runs at least $\frac{\ln n!}{\ln 3}$, we can conclude it has asymptotic lower bound of $\Omega(n \ln n)$, and since the algorithm runs at most $\frac{\ln n!}{\ln 3} + n$, we can conclude it has upper bound running time of $\mathcal{O}(n \ln n)$.

Since the value of Ω and $\mathcal{O}$ are the same, we can conclude the algorithm has running time of $\Theta(n \ln n)$ or $\Theta(n \log n)$.

Notes:

- In a main flow of proof, when there is a huge interruption like showing $\ln(n!) \in \Theta(n \ln n)$, how can a sentence be started to tell the audience we are working on another major idea?
- When an interruption in proof has been occured for another major part of a proof, how can a sentence be started to combine parts together?
- How can a sentence be written to say condition x_1, x_2, and x_3 are satisfied, so a statement y can be used to an equation or an idea?

Question 2

a. We need to evaluate tight asymptotic upper bound.

We will prove that the tight asymptotic upper bound of the algorithm is $\mathcal{O}(n^2)$.

First, we need to analyze the number of iterations of loop 2 per iteration of loop 1.

The code tells us loop 2 starts at $j = 0$ and ends at most$j = i - 1$ with j increasing by 1 per iteration.

Then, using these facts, we can conclude loop 2 has at most

$$\left\lceil \frac{i-1-0+1}{1} \right\rceil = i \tag{1}$$

iterations.

Next, we need to determine the total number of iterations of loop 2 over all iterations of loop 1.

The code tells us that loop 1 starts at $i = n$ and ends at most $i = 0$ with i decreasing by 1 per iteration.

Because we know each iteration of loop 1 takes i iterations by loop 2, using these facts, we can conclude the total number of iterations of loop 2 is at most

$$n + (n-1) + (n-2) + \cdots + 0 = \sum_{i=1}^{n} \tag{2}$$

$$= \frac{n(n+1)}{2} \tag{3}$$

iterations, or $\mathcal{O}(n^2)$.

Correct Solution:

We need to evaluate tight asymptotic upper bound.

We will prove that the tight asymptotic upper bound of the algorithm is $\mathcal{O}(n^2)$.

First, we need to analyze the cost of loop 2.

The code tells us loop 2 starts at $j = 0$ and ends at most$j = i - 1$ with j increasing by 1 per iteration.

Then, since each iteration of loop 2 takes a constant step (1 step), using these facts, we can conclude the cost of loop 2 is at most

$$1 \cdot (i - 1 - 0 + 1) = i \tag{1}$$

steps.

Next, we need to determine cost of loop 1.

The code tells us that loop 1 starts at $i = n$ and ends at most $i = 0$ with i decreasing by 1 per iteration.

Because we know each iteration of loop 1 takes $i + 1$ steps (where i is from loop 2 and 1 from line 8), using these facts, we can conclude the total cost of loop 1 is at most

$$
\begin{aligned}
(n+1) + n + (n-1) + (n-2) + \cdots + 1 &= \sum_{i=0}^{n}(i+1) && (2) \\
&= \sum_{i=0}^{n} i + \sum_{i=0}^{n} 1 && (3) \\
&= \sum_{i=0}^{n} i + (n+1) && (4) \\
&= \frac{n(n+1)}{2} + (n+1) && (5) \\
&= \frac{(n+1)(n+2)}{2} && (6)
\end{aligned}
$$

steps.

Finally, adding the cost of line 6, we can conclude the algorithm has total cost of $\frac{(n+1)(n+2)}{2} + 1$ steps, which is $\mathcal{O}(n^2)$.

Notes:

- Noticed professor writes proof that gets to a point (i.e. ... where each iteration takes $i + 1$ **steps**), and provides more detailed explanation in brackets (i.e. ... where each iteration takes $i + 1$ steps **(Adding the cost of loop 2 and 1 step for other constant time operations)**).
- Noticed professor uses 'finally' when proof has reached the final step that leads to its conclusion.

b. Let $n, k \in \mathbb{N}$, and $list = [0, 0, \ldots, 0, 1, 0, \ldots, 0]$ where 1 is at $\lceil \frac{n}{2} \rceil$ position.

We will prove that the tight asymptotic lower bound running time of this algorithm is $\Omega(n^2)$.

First, we need to evaluate the cost of loop 2.

The code tells us loop 2 starts at $j_k = 0$, and j_k will increase by 1 until $j_k \geq \lceil \frac{n}{2} \rceil + 1$ (where $+1$ is because of loop 2 stopping at $j_k = \lceil \frac{n}{2} \rceil$ by the if condition on line 10).

Using the fact $j_k = k + 1$, we can calculate that loop 2 stops when

$$k + 1 \geq \left\lceil \frac{n}{2} \right\rceil + 1 \tag{1}$$
$$k \geq \left\lceil \frac{n}{2} \right\rceil \tag{2}$$

Since we are looking for the smallest value of k (because the smallest value of k translates to number of iterations), we can conclude the loop has

$$\left\lceil \frac{n}{2} \right\rceil \tag{3}$$

iterations.

Since each iteration takes a constant time (1 step), the cost of loop 2 is

$$\left\lceil \frac{n}{2} \right\rceil \cdot 1 = \left\lceil \frac{n}{2} \right\rceil \tag{4}$$

steps.

Next, we need to evaluate the cost of loop 1.

The code tell us loop 1 will start at $i_k = n$, and i_k will decrease by 1 per iteration until $i_k \leq \lceil \frac{n}{2} \rceil$.

Using the fact $i_k = k - 1$, we can write loop 1 stops when

$$k - 1 \leq \left\lceil \frac{n}{2} \right\rceil \tag{5}$$
$$k \leq \left\lceil \frac{n}{2} \right\rceil + 1 \tag{6}$$

Since we are looking for the largest value of k (because the largest value of k translates to number of iterations), we can conclude loop 1 has

$$\left\lceil \frac{n}{2} \right\rceil + 1 \tag{7}$$

iterations.

Since each costs $\lceil \frac{n}{2} \rceil + 1$ steps, we can conclude loop 1 has cost of

$$\left(\left\lceil \frac{n}{2} \right\rceil + 1\right)\left(\left\lceil \frac{n}{2} \right\rceil + 1\right) = \left\lceil \frac{n}{2} \right\rceil^2 + 2 \cdot \left\lceil \frac{n}{2} \right\rceil + 1 \tag{8}$$

steps.

Finally, by adding the cost of line 6 (1 step), the total running time of this algorithm is

$$\left\lceil \frac{n}{2} \right\rceil^2 + 2 \cdot \left\lceil \frac{n}{2} \right\rceil + 2 \tag{9}$$

steps, which is $\Omega(n^2)$

Correct Solution:

Let $n, k \in \mathbb{N}$, and $list = [0, 0, \ldots, 0, 1, 0, \ldots, 0]$ where 1 is at $\lceil \frac{n}{2} \rceil$ position.

We will prove that the tight asymptotic lower bound running time of this algorithm is $\Omega(n^2)$.

First, we need to evaluate the cost of loop 2.

The code tells us loop 2 starts at $j_k = 0$, and j_k will increase by 1 until $j_k \geq \lceil \frac{n}{2} \rceil + 1$ (where +1 is because of loop 2 stopping at $j_k = \lceil \frac{n}{2} \rceil$ by the if condition on line 10).

Using the fact $j_k = k$, we can calculate that loop 2 stops when

$$k \geq \left\lceil \frac{n}{2} \right\rceil + 1 \tag{1}$$

Since we are looking for the smallest value of k (because the smallest value of k translates to number of iterations), we can conclude the loop has

$$\left\lceil \frac{n}{2} \right\rceil + 1 \tag{2}$$

iterations.

Since each iteration takes a constant time (1 step), the cost of loop 2 is

$$\left(\left\lceil \frac{n}{2} \right\rceil + 1\right) \cdot 1 = \left\lceil \frac{n}{2} \right\rceil + 1 \tag{3}$$

steps.

Next, we need to evaluate the cost of loop 1.

The code tell us loop 1 will start at $i_k = n$, and i_k will decrease by 1 per iteration until $i_k \leq \lceil \frac{n}{2} \rceil$.

Using the fact $i_k = n - k$, we can write loop 1 stops when

$$n - k \leq \left\lceil \frac{n}{2} \right\rceil \tag{4}$$

$$-k \leq \left\lceil \frac{n}{2} \right\rceil - n \tag{5}$$

$$k \geq n - \left\lceil \frac{n}{2} \right\rceil \tag{6}$$

Since we are looking for the largest value of k (because the largest value of k translates to number of iterations), we can conclude loop 1 has

$$n - \left\lceil \frac{n}{2} \right\rceil \tag{7}$$

iterations.

Since each iteration costs $\lceil \frac{n}{2} \rceil + 2$ steps (where $\lceil \frac{n}{2} \rceil + 1$ is the cost of loop 2 and $+1$ is the cost of line 14), we can conclude loop 1 has cost of

$$\left(\left\lceil \frac{n}{2} \right\rceil + 2\right)\left(n - \left\lceil \frac{n}{2} \right\rceil\right) \tag{8}$$

steps.

Finally, since the loop takes $\lceil \frac{n}{2} \rceil + 1$ extra steps (where $\lceil \frac{n}{2} \rceil$ is the cost of traveling from $j = 0$ until $j = \lceil \frac{n}{2} \rceil$ and $+1$ is the cost of line 14) before coming to a full stop, the total running time is at least

$$\left(\left\lceil \frac{n}{2} \right\rceil + 2\right)\left(n - \left\lceil \frac{n}{2} \right\rceil\right) + \left\lceil \frac{n}{2} \right\rceil + 1 \tag{9}$$

steps, which is $\Omega(n^2)$

Notes:

- Noticed there is no room for errors. (most of mark deductions are from not being careful with the analysis).
- Realized I need to take time to verify and re-verify steps using examples at a very fine level (i.e at this step this happens ... at this step this happens) until conclusion.
- Noticed professor uses $i_k = n - k$ when going backward starting from n. And for the inequality, $i_k \leq$ is used as opposed to the normal $i_k \geq$.

c. *Proof.* Let $k, n \in \mathbb{N}$.

We will prove the statement using proof by cases.

Case 1: When all elements in $nums$ are even

Let $nums = [a_1, a_2, \ldots, a_n]$ where $a_1, \ldots, a_n$ are even numbers.

We want to prove the best-case lower bound running time of this algorithm is $\Omega(n)$.

First, we need to analyze the cost of loop 2.

Given the iteration count k, the code tells us, the loop starts at $j_k = 0$ and increases by 1 per iteration, and so we know $j_k = k$.

Because we know loop 2 runs until $j_k \geq i$, we can conclude loop 2 stops when

$$k \geq i \tag{1}$$

Since we are looking for the smallest value of k (because it represents the number of iterations), we can conclude loop 2 has i iterations.

Because we know each iteration of loop 2 costs a constant time (1 step), we can conclude loop 2 has cost of at least

$$k \cdot 1 = k \tag{2}$$

steps.

Now, we need to evaluate the cost of loop 1.

The code tells us loop 1 starts at $i = n$ and ends at $i = n$ due to the truthy condition of line 14.

Using these facts, we can conclude loop 1 has

$$\lceil n - n + 1 \rceil = 1 \tag{3}$$

iteration.

Because we know each iteration of loop 1 costs $i+2$ steps (where i is from the cost of loop 2, and $+2$ are from the cost of line 8 and line 16), we can conclude loop 1 has cost of at least

$$(i+2) \cdot 1 = i + 2 \tag{4}$$

steps.

Finally, because we know $i = n$, the total running time is at least $n + 2$, which is $\Omega(n)$.

Case 2: When one or more elements in $nums$ are odd

Let $nums = [1, a_2, a_3, \ldots, a_{n-1}]$ where $a_2, a_3, \ldots, a_{n-1}$ are even numbers.

We will prove the algorithm has best-case lower bound running time of $\Omega(n)$.

First, we need to evaluate the cost of loop 2.

The code tellus us loop 2 starts at $j = 0$ and ends at $j = 0$ due to the truthy condition of line 10.

Using these facts, we can calculate loop 2 has 1 iteration.

Because we know loop 2 takes constant time (1 step) per iteration, we can conclude loop 2 has cost of 1 step.

Next, we need to evaluate the cost of loop 1.

The code tells us that loop 1 starts at $i = n$, and i increases by 1 until $i_k \leq -1$, where k represents the iteration count of loop 1.

Because we know $i_k = n - k$, we can conclude the loop stops when

$$n - k \leq -1 \tag{5}$$
$$k \geq n + 1 \tag{6}$$

Since we are looking for the smallest value of k (because it represents the number of iterations), we can conclude loop 1 has

$$n + 1 \tag{7}$$

Since each iteration of loop 1 takes 2 steps (where 1 is the cost of loop 2 and the other 1 is the cost of line 8), we can conclude that loop 1 has cost of at least

$$2 \cdot (n + 1) \tag{8}$$

steps.

Finally, adding the cost of line 8, we can conclude the algorithm has running time of at least $2(n + 1) + 1$ steps, which is $\Omega(n)$. □

Attempt 2:

Let $k, n \in \mathbb{N}$.

We will prove this statement using proof by cases.

Case 1: When all elements in $nums$ are even

Let $nums = [a_1, a_2, \ldots, a_n]$ where $a_1, \ldots, a_n$ are even numbers.

We want to prove the best-case lower bound running time of this algorithm is $\Omega(n)$.

First, we need to analyze the cost of loop 2.

Given the iteration count k, the code tells us, the loop starts at $j_k = 0$ and increases by 1 per iteration, and so we know $j_k = k$.

Because we know loop 2 runs until $j_k \geq i$, we can conclude loop 2 stops when

$$k \geq i \tag{1}$$

Since we are looking for the smallest value of k (because it represents the number of iterations), we can conclude loop 2 has i iterations.

Because we know each iteration of loop 2 costs a constant time (1 step), we can conclude loop 2 has cost of at least

$$k \cdot 1 = k \tag{2}$$

steps.

Now, we need to evaluate the cost of loop 1.

The code tells us loop 1 starts at $i = n$ and ends at $i = n$ due to the truthy condition of line 14.

Using these facts, we can conclude loop 1 has

$$\lceil n - n + 1 \rceil = 1 \tag{3}$$

iteration.

Because we know each iteration of loop 1 costs $i + 2$ steps (where i is from the cost of loop 2, and +2 are from the cost of line 8 and line 16), we can conclude loop 1 has cost of at least

$$(i + 2) \cdot 1 = i + 2 \tag{4}$$

steps.

Finally, because we know $i = n$, the total running time is at least $n+2$, which is $\Omega(n)$.

Case 2: When one or more elements in $nums$ are odd

In this case, let m be the index of first odd number in $nums$.

We need to prove this algorithm has best-case lower bound running time of $\Omega(n)$.

First, we need to evaluate the cost of loop 2.

Given loop 2 iteration count k, the code tells us loop 2 starts at $j = 0$, and j increases by 1 until $j_k \geq m + 1$.

Since we know $j_k = k$, using these facts, we can calculate loop 2 terminates when

$$k \geq m + 1 \tag{5}$$

Since we are looking for the smallest value of k (because it represents the number of iterations), we can conclude loop 2 has

$$m + 1 \tag{6}$$

iterations.

Next, we need to evaluate the cost of loop 1.

Given loop 1 iteration count k,, The code tells us that loop 1 starts at $i = n$, and i decreases by 1 until $i_k \leq m - 1$.

Since we know $i_k = n - k$, using these facts, we can calculate loop 1 stops when

$$n - k \leq m - 1 \tag{7}$$
$$k \geq n - m + 1 \tag{8}$$

Since we are looking for the smallest value of k (because it represents the number of iterations), we can conclude loop 1 has

$$n - m + 1 \tag{9}$$

iterations.

Because we know that for the first $n-k$ iterations, each iteration of loop 1 costs $m+2$ steps (where $m+1$ is the cost of loop 2 and $+1$ is the cost of line 8), and last iteration of loop 1 costs another $m+2$ (where m is the cost of loop 2 and $+2$ are the cost of line 8 and 15), we can conclude loop 1 has cost of

$$(n - m + 1)(m + 2) \tag{10}$$

steps.

Next, adding the cost of line 6, we can conclude the algorithm has total cost of at least

$$(n - m + 1)(m + 2) + 1 \tag{11}$$

steps.

Finally, we need to show this algorithm has runtime of $\Omega(n)$.

Using the total cost of algorithm, we can calculate

$$\begin{aligned}
(n - m + 1)(m + 2) + 1 &= (n - m)(m + 2) + (m + 2) + 1 && (12)\\
&> (n - m)(m + 2) + (m + 2) && (13)\\
&= (n - m)m + 2(n - m) + (m + 2) && (14)\\
&> (n - m)m + (n - m) + m && (15)\\
&= (n - m)m + n && (16)
\end{aligned}$$

Because we know $n - m \geq 0$ and $m \geq 0$, we can conclude that

$$(n - m + 1)(m + 2) + 1 > n \tag{17}$$

and the algorithm has best case lower bound running time of $\Omega(n)$.

Notes:

- The solution in problem 2.b adds constant time operations into total cost where as the solution to this problem doesn't... Is there a rule behind when and when not they can be included?
- Noticed professor reduces the exact cost to n by separating it from the rest of the terms

$$(n - m + 1)(m + 2) > (n - m)m + n$$

- Realized the best-case lower bound running time doesn't use input family like worst-case lower bound running time

Question 3

a. *Proof.* Let $n \in \mathbb{N}$ and lst be a list with all negative numbers.

Then, the code tells us line 9-12 will run for all elements in the list.

Because we know i increases by a factor of 2 per iteration, we can conclude that at k^{th} iteration, i has value of $i_k = 2^k$.

Because we know loop terminates when $i_k \geq n$, we can conclude this is true when

$$2^k \geq n \tag{1}$$
$$k \geq \log n \tag{2}$$

Since we are looking for the smallest value of k (since it represents the number of iterations), we can conclude loop has

$$\lceil \log n \rceil \tag{3}$$

iterations.

Since each iteration of while loop takes a constant time (1 step), we can conclude the loop has cost of

$$\lceil \log n \rceil \tag{4}$$

steps.

Finally, since lines 2 to 4 have cost of 1 each, by adding to the costs together, we can conclude the algorithm has total running time of $\lceil \log n \rceil + 3$, which is $\Theta(\log n)$. $\square$

Correct Solution:

Let $n, k \in \mathbb{N}$ and lst be a list with all negative numbers.

In this case, the loop follows this pattern

- **iteration 1** - else condition executes and j increases by a factor of 2
- **iteration 2** - if condition executes and i increases by a factor of 2, and moves to where j is
- **iteration 3** - else condition executes again and j increases by a factor of 2
- **iteration 4** - if condition executes again and i increases by a factor of 2 and moves to where j is.
- and this pattern repeats until the end of while loop.

Now, we need to determine the total number of iterations in while loop.

Because we know i increases by a factor of 2 per execution of **if lst[i] >= 0:** condition, we can conclude that at k^{th} execution of **if lst[i] >= 0:** condition, i has value of $i_k = 2^k$.

Because we know loop terminates when $i_k \geq n$, we can conclude this is true when

$$2^k \geq n \tag{1}$$
$$k \geq \log n \tag{2}$$

Since we are looking for the smallest value of k (since it represents the number of executions caused by the **if lst[i] >= 0:** condition), we can conclude loop has

$$\lceil \log n \rceil \tag{3}$$

executions due to the **if lst[i] >= 0:** condition.

Because we know every execution of **if lst[i] >= 0:** condition in an iteration, is followed by the execution of **else:** condition in previous iteration, we can conclude while loop has total of

$$2 \cdot \lceil \log n \rceil \tag{4}$$

executions, or iterations.

Since each iteration of while loop takes a constant time (1 step), we can conclude the while loop has cost of

$$2 \cdot \lceil \log n \rceil \tag{5}$$

steps.

Finally, adding cost of 1 for the constant time operations on line 2-4, we can conclude the algorithm has total running time of $2 \cdot \lceil \log n \rceil + 1$ steps, which is $\Theta(\log n)$.

Notes:

- Noticed professor bundles up time of constant operations (i.e. line 2-4) to 1, and same for the ones within while loop.
- Noticed professor introduces k in body as 'k^{th} execution of the if/else branch', and he doesn't introduce the variable in header.
- Noticed professor uses the word 'execution' to focus on the number of iterations caused by the if condition.
- Noticed professor lays out the pattern in while loop before moving onto proof.

b. *Proof.* Let $n \in \mathbb{N}$, and lst be a list of integers where $lst[0]$ to $lst[\frac{n}{2}]$ have value 0, and $lst[\frac{n}{2} + 1]$ to $lst[n-1]$ have value -1.

In this case, the loop follows this pattern:

- When $j = 1$, if branch performs $\frac{n}{2} + 1$ executions, stops at $i = \frac{n}{2} + 1$

- else branch performs, and value of j doubles, and i resets to 0
- When $j = 2$, if branch performs $\frac{n}{4} + 1$ executions, stops at $i = \frac{n}{2} + 2$
- else branch performs, and value of j doubles, and i resets to 0
- When $j = 4$, if branch performs $\frac{n}{8} + 1$ executions, stops at $i = \frac{n}{2} + 4$
- else branch performs, and value of j doubles, and i resets to 0
- When $j = 8$, if branch performs $\frac{n}{16} + 1$ executions, stops at $i = \frac{n}{2} + 8$
- This pattern repeats until k^{th} execution of else branch of statements has value of j half the size of n.
- Loop terminates one iteration after i reaches the end of array.

Now, we will prove this statement in two parts: one for determining the number of executions of else branch of statements in while loop, and another for determining the runtime of whole algorithm.

Part 1: Determining the number of executions of else branch:

We need to prove the else branch executes $\Omega(\log n)$ times.

The pattern tells us while loop depends on j, and j increases by a factor of 2 per execution of else branch until $j_{k+1} \geq n$.

Because we know at k^th execution of else branch has j with value of $j_k = 2^k$, using these facts, we can calculate

$$2^{k+1} \geq n \tag{1}$$
$$k + 1 \geq \log n \tag{2}$$
$$k \geq \log n - 1 \tag{3}$$

So we know the else branch executes at least $\log n - 1$ times, which is $\Omega(\log n)$.

Part 2: Determining running time of algorithm:

We need to prove this algorithm has running time of $\Theta(n)$.

First, we need to determine number of executions of if branch in while loop.

The pattern tells us at k^{th} execution of else branch of statements in while loop, $\frac{n}{2^{k+1}} + 1$ many executions of if branch of statements are performed.

Since loop performs $\log n - 1$ many executions of the else branch of statements, we can conclude

$$\sum_{k=1}^{\log n-2} \left(\frac{n}{2^{k+1}} + 1\right) \tag{4}$$

many executions of the if branch are performed.

Then, since we know $\log n \in \mathbb{N}$ due to n being in factors of 2, using the fact $\forall n \in \mathbb{N}, \forall r \in \mathbb{R}, \sum_{i=0}^{n-1} r^i = \frac{1-r^n}{1-r}$, we can calculate that

$$\begin{aligned}
\sum_{k=1}^{\log n-2} \left(\frac{n}{2^{k+1}} + 1\right) &= \sum_{k=1}^{\log n-2} \frac{n}{2^{k+1}} + \sum_{k=1}^{\log n-2} 1 && (5)\\
&= \frac{n}{2} \cdot \sum_{k=1}^{\log n-2} \frac{1}{2^k} + \sum_{k=1}^{\log n-2} 1 && (6)\\
&= \frac{n}{2} \cdot \sum_{k=1}^{\log n-2} \frac{1}{2^k} + (\log n - 2) && (7)\\
&= \frac{n}{2} \cdot \left(\frac{1 - \frac{2}{2^{\log n}}}{1 - \frac{1}{2}}\right) + (\log n - 2) && (8)\\
&= n \cdot \left(1 - \frac{2}{n}\right) + (\log n - 2) && (9)\\
&= n - 2 + \log n - 2 && (10)\\
&= n + \log n - 4 && (11)
\end{aligned}$$

Now, adding the cost of the number of executions of else statements and the extra iteration taken to verify loop's terminating condition, we can conclude while loop has total of

$$n + \log n - 4 + (\log n - 1) + 1 = n + 2\log n - 4 \tag{12}$$

executions or iterations.

Since each execution takes a constant time (1 step), we can conclude while loop has cost of

$$1 \cdot (n + 2\log n - 4) = (n + 2\log n - 4) \tag{13}$$

steps.

Finally, adding constant time operations on line 2 to 4 (1 step), the algorithm has running time of

$$n + 2\log n - 3 \tag{14}$$

which is $\Theta(n)$.

$\square$

<u>Notes:</u>

- I analyzed the example $[0, 0, -1, -1]$. This is what I found.

  - **iteration 1:** if brach of statement executes and i increases by a 1 ($i = 1$, $j = 1$)
  - **iteration 2:** if brach of statement executes and i increases by a 1 ($i = 2$, $j = 1$)
  - **iteration 3:** if brach of statement executes and i increases by a 1 ($i = 3$, $j = 1$)
  - **iteration 4:** else branch of statement executes, causing **lst[i] = abs(lst[i])**, $i = 0$, and j to increase by twice of its size ($i = 0$, $j = 2$)

    The following is how the list looks after update

    $$[0, 0, 1, -1]$$

  - **iteration 5:** if brach of statement executes and i increases by a 2 ($i = 2$, $j = 2$)
  - **iteration 6:** if brach of statement executes and i increases by a 2 ($i = 4$, $j = 2$)
  - **iteration 7:** Loop terminates,

  <u>Here's what I found about j</u>

  - Loop terminates when $k+1^{th}$ execution of else statement is greater than or equal to n.

  <u>Here's what I found about i</u>

  - When $j = 1$, loop performs $\frac{n}{2} + 1$ executions, stops at $\frac{n}{2} + 1$
  - When $j = 2$, loop performs $\frac{n}{4} + 1$ executions, stops at $\frac{n}{2} + 2$
  - loop terminates 1 after

- Number of execution of if branch of statements depend on the number of execution of else branch of statements

$$\textbf{num of exec. of if statements} = \sum_{k=0}^{\text{num of exec. of else}} \left(\frac{n}{2^k + 1} + 1 \right) \tag{15}$$

- I analyzed the example $[0, 0, 0, 0, -1, -1, -1, -1]$. This is what I found.
  - **iteration 1:** if brach of statement executes and i increases by a 1 ($i = 1$, $j = 1$)
  - **iteration 2:** if brach of statement executes and i increases by a 1 ($i = 2$, $j = 1$)
  - **iteration 3:** if brach of statement executes and i increases by a 1 ($i = 3$, $j = 1$)
  - **iteration 4:** if brach of statement executes and i increases by a 1 ($i = 4$, $j = 1$)
  - **iteration 5:** else branch of statement executes, causing **lst[i] = abs(lst[i])**, $i = 0$, and j to increase by twice of its size ($i = 0$, $j = 2$)

    The following is how the list looks after update

    $$[0, 0, 0, 0, 1, -1, -1, -1]$$

  - **iteration 6:** if brach of statement executes and i increases by a 2 ($i = 2$, $j = 2$)
  - **iteration 7:** if brach of statement executes and i increases by a 2 ($i = 4$, $j = 2$)
  - **iteration 8:** if brach of statement executes and i increases by a 2 ($i = 6$, $j = 2$)
  - **iteration 9:** else branch of statement executes, causing **lst[i] = abs(lst[i])**, $i = 0$, and j to increase by twice of its size ($i = 0$, $j = 4$)

    The following is how the list looks after update

    $$[0, 0, 0, 0, 1, -1, 1, -1]$$

  - **iteration 10:** if brach of statement executes and i increases by a 4 ($i = 4$, $j = 4$)
  - **iteration 11:** if brach of statement executes and i increases by a 4 ($i = 8$, $j = 4$)
  - **iteration 12:** Loop terminates,

Here's what I found about j

  - Loop terminates when $k+1^{th}$ execution of else statement is greater than or equal to n.

Here's what I found about i

  - When $j = 1$, loop performs $\frac{n}{2} + 1$ iterations, stops at $\frac{n}{2} + 1$

- When $j = 2$, loop performs $\frac{n}{4} + 1$ iterations, stops at $\frac{n}{2} + 2$
- When $j = 4$, loop performs $\frac{n}{8} + 1$ iterations, stops at $\frac{n}{2} + 4$
- Loop terminates 1 after
- Number of execution of if branch of statements depend on the number of execution of else branch of statements

$$\textbf{num of exec. of if statements} = \sum_{k=0}^{\text{num of exec. of else}} \left(\frac{n}{2^k + 1} + 1 \right) \tag{16}$$

- I analyzed the example $[0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, -1, -1, -1, -1, -1]$. This is what I found.

  - **iteration 1:** if brach of statement executes and i increases by a 1 ($i = 1$, $j = 1$)
  - **iteration 2:** if brach of statement executes and i increases by a 1 ($i = 2$, $j = 1$)
  - **iteration 3:** if brach of statement executes and i increases by a 1 ($i = 3$, $j = 1$)
  - **iteration 4:** if brach of statement executes and i increases by a 1 ($i = 4$, $j = 1$)
  - **iteration 5:** if brach of statement executes and i increases by a 1 ($i = 5$, $j = 1$)
  - **iteration 6:** if brach of statement executes and i increases by a 1 ($i = 6$, $j = 1$)
  - **iteration 7:** if brach of statement executes and i increases by a 1 ($i = 7$, $j = 1$)
  - **iteration 8:** if brach of statement executes and i increases by a 1 ($i = 8$, $j = 1$)
  - **iteration 9:** if brach of statement executes and i increases by a 1 ($i = 9$, $j = 1$)
  - **iteration 10:** else branch of statement executes, causing **lst[i] = abs(lst[i])**, $i = 0$, and j to increase by twice of its size ($i = 0$, $j = 2$)

    The following is how the list looks after update

    $$[0, 0, 0, 0, 0, 0, 0, 0, 1, -1, -1, -1, -1, -1, -1, -1]$$

  - **iteration 11:** if brach of statement executes and i increases by a 2 ($i = 2$, $j = 2$)
  - **iteration 12:** if brach of statement executes and i increases by a 2 ($i = 4$, $j = 2$)
  - **iteration 13:** if brach of statement executes and i increases by a 2 ($i = 6$, $j = 2$)
  - **iteration 14:** if brach of statement executes and i increases by a 2 ($i = 8$, $j = 2$)
  - **iteration 15:** if brach of statement executes and i increases by a 2 ($i = 10$, $j = 2$)
  - **iteration 16:** else branch of statement executes, causing **lst[i] = abs(lst[i])**, $i = 0$, and j to increase by twice of its size ($i = 0$, $j = 4$)

    The following is how the list looks after update

    $$[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, -1, -1, -1, -1, -1, -1]$$

- **iteration 17:** if brach of statement executes and i increases by a 4 ($i = 4$, $j = 4$)
- **iteration 18:** if brach of statement executes and i increases by a 4 ($i = 8$, $j = 4$)
- **iteration 19:** if brach of statement executes and i increases by a 4 ($i = 12$, $j = 4$)
- **iteration 20:** else branch of statement executes, causing **lst[i] = abs(lst[i])**, $i = 0$, and j to increase by twice of its size ($i = 0$, $j = 8$)

  The following is how the list looks after update

$$[0, 0, 0, 0, 0, 0, 0, 0, 1, 1, -1, 1, -1, -1, -1, -1]$$

- **iteration 21:** if brach of statement executes and i increases by a 8 ($i = 8$, $j = 8$)
- **iteration 22:** if brach of statement executes and i increases by a 8 ($i = 16$, $j = 8$)
- **iteration 23:** Loop terminates.

  <u>**Here's what I found about j**</u>

  * Loop terminates when $k + 1^{th}$ execution of else statement is greater than or equal to n.

  <u>**Here's what I found about i**</u>

  * When $j = 1$, if branch performs $\frac{n}{2} + 1$ executions, stops at $\frac{n}{2} + 1$
  * When $j = 2$, if branch performs $\frac{n}{4} + 1$ executions, stops at $\frac{n}{2} + 2$
  * When $j = 4$, if branch performs $\frac{n}{8} + 1$ executions, stops at $\frac{n}{2} + 4$
  * When $j = 8$, if branch performs $\frac{n}{16} + 1$ executions, stops at $\frac{n}{2} + 8$
  * Loop terminates 1 after
  * Number of execution of if branch of statements depend on the number of execution of else branch of statements

$$\textbf{num of exec. of if statements} = \sum_{k=0}^{\text{num of exec. of else}} \left(\frac{n}{2^k + 1} + 1 \right) \tag{17}$$

- Realized the need to learn how to organize ideas for proof
- Realized the need to learn how to connect the dots or lay structure to proofs given sets of ideas
- Realized concepts involved are 1. finding examples 2. finding patterns in example 3. generalizing patterns 4. write how am i going to solve problem 5. lay out big ideas 6. chunk out big ideas into smaller parts 7. solve the small parts

- Realized building a large proof without organizing ideas feels like jumping into solving pramp problems without pseudocode on how to solve it.

  I wonder how to lay pseudocode or organize ideas for proofs...
- Realized I am keep losing details because my brain can't hold too much of information.
- Realized writing proof feels similar to writing algorithms

c. *Proof.* Let $n \in \mathbb{N}$.

We will prove the algorithm has worst-case running time of $\mathcal{O}(n)$.

First, we need to determine the total cost of algorithm.

The code tells us maximum number of while loop occurs when i increases by 1, and this is true when only if branch of statements occur.

Since i starts at 0, and finishes at $i = n - 1$, we can conclude the loop has

$$n - 1 - 0 + 1 = n \tag{18}$$

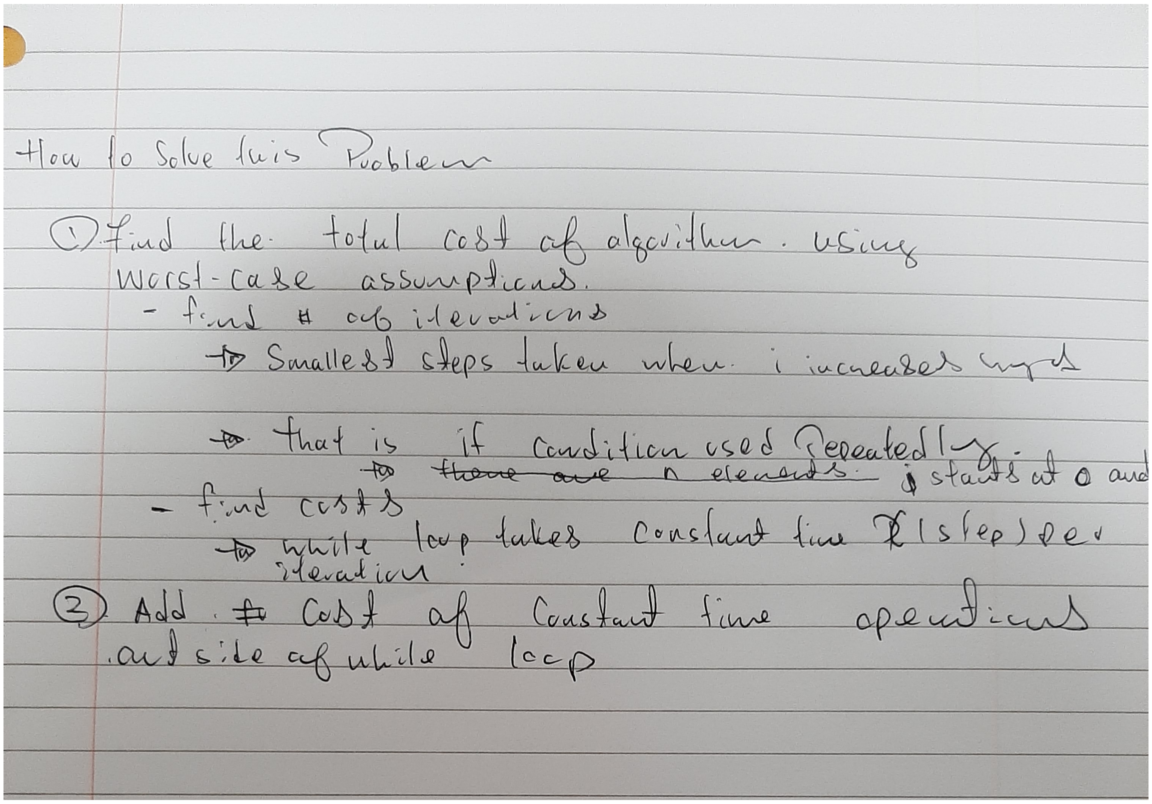
iterations.

Since each iteration of loop takes constant time operations (1 step), we can conclude the algorithm has total of n steps.

Finally, adding the cost of constant time operations outside of while loop, we can conclude look takes $n + 1$ steps, which is $\mathcal{O}(n)$. □

Notes:

- Laid out proof like done with pramp problems. Realized the writing of proof feels smoother.



How to Solve this Problem

① Find the total cost of algorithm using worst-case assumptions.

- find # of iterations
  - → Smallest steps taken when i increases [illegible]
  - → that is if condition used repeatedly.
  - → ~~there are n elements.~~ i starts at 0 and
- find costs
  - → while loop takes constant time (1 step) per iteration

② Add # cost of constant time operations outside of while loop

- Noticed professor has solution that is a lot different than what I thought... Is there concepts I misunderstood?

Question 4

a. **Statement:** $\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, \frac{n}{2^k} - \frac{2^k - 1}{2^k} \leq x_k \leq \frac{n}{2^k}$

Proof. We will prove by induction on k.

Base Case ($k = 0$):

Let $k = 0$ and $n \in \mathbb{Z}^+$.

We need to show $n \leq x_0 \leq n$, or $x_0 = n$.

It follows from the code that at 0^{th} iteration, the value of x is n.

Inductive Case ($k \in \mathbb{N}$):

Let $k \in \mathbb{N}$, and assume the statement is true at k.

We will to prove $\frac{n}{2^{k+1}} - \frac{2^{k+1}-1}{2^{k+1}} \leq x_{k+1} \leq \frac{n}{2^{k+1}}$ in two parts, by showing $\frac{n}{2^{k+1}} - \frac{2^{k+1}-1}{2^{k+1}} \leq x_{k+1}$ and $x_{k+1} \leq \frac{n}{2^{k+1}}$.

Part 1 (Showing $\frac{n}{2^{k+1}} - \frac{2^{k+1}-1}{2^{k+1}} \leq x_{k+1}$):

Starting from x_{k+1}, the code tells us

$$x_{k+1} = \left\lfloor \frac{x_k}{2} \right\rfloor \tag{1}$$

Then, by the hint ($\forall x \in \mathbb{Z}$, $\frac{x-1}{2} \leq \left\lfloor \frac{x}{2} \right\rfloor \leq \frac{x}{2}$), we can write

$$\begin{align} x_{k+1} = \left\lfloor \frac{x_k}{2} \right\rfloor &\geq \frac{x_k - 1}{2} \tag{2} \\ &= \frac{1}{2} \cdot (x_k - 1) \tag{3} \end{align}$$

Then, by inductive hypothesis,

$$\begin{align} x_{k+1} &\geq \frac{1}{2} \cdot \left(\frac{n}{2^k} - \frac{2^k - 1}{2^k} - 1 \right) \tag{4} \\ &= \frac{n}{2^{k+1}} - \frac{2^k - 1}{2^{k+1}} - \frac{1}{2} \tag{5} \\ &= \frac{n}{2^{k+1}} - \left(\frac{2^k - 1}{2^{k+1}} + \frac{2^k}{2^{k+1}} \right) \tag{6} \\ &= \frac{n}{2^{k+1}} - \left(\frac{2^k + 2^k - 1}{2^{k+1}} \right) \tag{7} \end{align}$$

Then, because we know $2^k + 2^k = 2^{k+1}$, we can conclude

$$x_{k+1} \geq \frac{n}{2^{k+1}} - \left(\frac{2^{k+1} - 1}{2^{k+1}} \right) \tag{8}$$

Part 2 (Showing $x_{k+1} \leq \frac{n}{2^{k+1}}$):

Starting from x_{k+1}, the code tells us

$$x_{k+1} = \left\lfloor \frac{x_k}{2} \right\rfloor \tag{9}$$

Then, by the hint ($\forall x \in \mathbb{Z}$, $\frac{x-1}{2} \leq \left\lfloor \frac{x}{2} \right\rfloor \leq \frac{x}{2}$), we can write

$$x_{k+1} = \left\lfloor \frac{x_k}{2} \right\rfloor \le \frac{x_k}{2} \tag{10}$$

Then, by the inductive hypothesis, we can conclude

$$\begin{aligned} x_{k+1} &\le \frac{n}{2^k \cdot 2} && (11) \\ &\le \frac{n}{2^{k+1}} && (12) \end{aligned}$$

□

b. **Statement:** $\forall n \in \mathbb{Z}^+$, $\forall k \in \mathbb{N}$, (**convert_to_binary**(n) takes exactly k loop iterations) $\Leftrightarrow 2^{k-1} \le n \le 2^k - 1$

Proof. Let $n \in \mathbb{Z}$, and $k \in \mathbb{N}$.

We will prove the statement in two parts (first is proving in $\Rightarrow$ direction, and the second is proving in $\Leftarrow$ direction).

Part 1 (Proving in $\Rightarrow$ direction):

Assume the loop in **convert_to_binary(n)** takes k iterations.

We need to prove $2^{k-1} \le n \le 2^k - 1$.

First, we need to show $2^{k-1} \le n$.

The code tells us at k^{th} iteration $x_k = \left\lfloor \frac{x_{k-1}}{2} \right\rfloor$ and $x_k = 0$.

Since the assumption tells us k iterations must occur in the loop, using these facts, we can conclude x_{k-1} is non-zero.

Then, because we know $0 < x_{k-1} = \left\lfloor \frac{x_{k-2}}{2} \right\rfloor \in \mathbb{N}$ and $0 = x_0 = \left\lfloor \frac{x_{k-1}}{2} \right\rfloor$, we can conclude $x_{k-1} = 1$.

Then, using this fact, with the inequality $\frac{n}{2^k} - \frac{2^k-1}{2^k} \le x_k \le \frac{n}{2^k}$ from question 4.a, we can conclude

$$1 = x_{k-1} \leq \frac{n}{2^{k-1}} \tag{1}$$
$$2^{k-1} \leq n \tag{2}$$

Now, we need to show $n \leq 2^k - 1$.

The code tells us that $x_0 = n$, $x_k = 0$, and from question 4.a, $\frac{n}{2^k} - \frac{2^k-1}{2^k} \leq x_k \leq \frac{n}{2^k}$.

Then, using these facts, we can conclude

$$n - \left(\frac{n}{2^k} - \frac{2^k-1}{2^k}\right) \geq x_0 - x_k = n \tag{3}$$
$$\frac{2^k \cdot n}{2^k} - \frac{n}{2^k} + \frac{2^k-1}{2^k} \geq n \tag{4}$$
$$\frac{n \cdot (n^k - 1)}{2^k} + \frac{2^k-1}{2^k} \geq n \tag{5}$$
$$\frac{2^k-1}{2^k} \geq n + \frac{n \cdot (2^k-1)}{2^k} \tag{6}$$
$$\frac{2^k-1}{2^k} \geq n \cdot \left(\frac{2^k - 2^k + 1}{2^k}\right) \tag{7}$$
$$2^k - 1 \geq n \tag{8}$$

Since $2^{k-1} \leq n$ and $n \leq 2^k - 1$ are true, we can conclude $2^{k-1} \leq n \leq 2^k - 1$ is true.

Part 2 (Proving in $\Leftarrow$ direction):

Assume $2^{k-1} \leq n \leq 2^k - 1$.

We need to prove that given n, the loop in **convert_to_binary(n)** takes k iterations.

First, we need to show that with the lower bound of n, the loop in **convert_to_binary(n)** does exactly k iterations.

The result of problem 4.a tells us

$$\forall n \in \mathbb{Z}^+, \, \forall k \in \mathbb{N}, \, \frac{n}{2^k} - \frac{2^k-1}{2^k} \leq x_k \leq \frac{n}{2^k} \tag{9}$$

Using this fact, we can calculate that the value of x at $k-1^{th}$ iteration is

$$\frac{2^{k-1}}{2^{k-1}} - \frac{2^{k-1}-1}{2^{k-1}} \leq x_{k-1} \leq \frac{2^{k-1}}{2^{k-1}} \tag{10}$$

$$\frac{1}{2^{k-1}} \leq x_{k-1} \leq 1 \tag{11}$$

Since $\frac{1}{2^{k-1}} > 0$, and $x_{k-1} \in \mathbb{N}$ (from the code), we can conclude $x_{k-1} = 1$.

Then, by taking an iteration further, we can conclude

$$x_k = \left\lfloor \frac{x_{k-1}}{2} \right\rfloor \tag{12}$$

$$= 0 \tag{13}$$

Because we know loop termination occurs when $x \leq 0$, we can conclude the loop with lower bound of n stops at k^{th} iteration.

Now, we need to show that with $2^k - 1$ as n, **convert_to_binary(n)** does exactly k iterations.

Using equation 9, we can calculate that the value of x at k^{th} iteration is

$$\frac{2^k-1}{2^k} - \frac{2^k-1}{2^k} \leq x_k \leq \frac{2^k-1}{2^k} \tag{14}$$

$$0 \leq x_k \leq \frac{2^k-1}{2^k} \tag{15}$$

Since we know $\frac{2^k-1}{2^k} < 1$, and $x_k \in \mathbb{N}$ (from the code), we can conclude $x_k = 0$.

Because we know loop termination occurs when $x \leq 0$, we can conclude the loop with the upper bound of n stops at k^{th} iteration.

So, since the loop stops at k^{th} iterations for both the upper and the lower bound of n, we can conclude n performs exactly k iterations. □

Notes:

- This is a tough problem.
- 형모 풀꼬얌!! 형모 궁뎡궁뎡 하고 한걸음띡 발쩐해죠 대학원 갈꼬얌!!
- 오예!!! 형모 해낼꼬다!!
- 형모 화이팅!!
- After hours of thinking, I found the rough idea: find range of values between ($x_1 and x_k$) and add to 2^{k-1} (where it's the last digit of binary number).

  (i.e 10000 and 11111 are two extreme range of values. Here we are finding last 4 0000 and 1111, and then adding to first 1).
- another one is using x_0 and x_k.

Pseudoproof:

Let $n \in \mathbb{Z}$, and $k \in \mathbb{N}$.

We will prove the statement in two parts (first is proving in $\Rightarrow$ direction, and the second is proving in $\Leftarrow$ direction).

Part 1 (Proving in $\Rightarrow$ direction):

Assume **convert_to_binary(n)** takes k step.

We need to show $2^{k-1} \leq n \leq 2^k - 1$.

1. Show $2^{k-1} \leq n$ is true
   - Show that x_{k-1} is greater than 1

     The code tells us $x_k = \left\lfloor \frac{x_{k-1}}{2} \right\rfloor$ and at k^{th} iteration $x_k = 0$.

     Since the assumption tells us k iterations must occur, we can conclude x_{k-1} is non-zero.

     Since we know from the code $x_{k-1} \in \mathbb{N}$, we can conclude $x_0 = 0$ will be true when $x_{k-1} = 1$.

   - Show $n \geq 2^{k-1}$

Then, using this fact, with the inequality $\frac{n}{2^k} - \frac{2^k-1}{2^k} \leq x_k \leq \frac{n}{2^k}$ from the result of question 4.a, we can conclude

$$1 = x_{k-1} \leq \frac{n}{2^{k-1}} \tag{16}$$
$$2^{k-1} \leq n \tag{17}$$

2. Show $n \leq 2^k - 1$ is true
   - start from the left and move to the right
     - Show that $x_0 = n$, $x_k = 0$ and $\frac{n}{2^k} - \frac{2^k-1}{2^k} \leq x_k \leq \frac{n}{2^k}$

       The code tells us that $x_0 = n$, $x_k = 0$, and from the result of question 4.a, we know $\frac{n}{2^k} - \frac{2^k-1}{2^k} \leq x_k \leq \frac{n}{2^k}$.

     - Use these facts to calculate that $2^k - 1 \geq n$

       Then, using these facts, we can conclude

$$n - \left(\frac{n}{2^k} - \frac{2^k-1}{2^k}\right) \geq x_0 - x_k = n \tag{18}$$
$$\frac{2^k \cdot n}{2^k} - \frac{n}{2^k} + \frac{2^k-1}{2^k} \geq n \tag{19}$$
$$\frac{n \cdot (n^k - 1)}{2^k} + \frac{2^k-1}{2^k} \geq n \tag{20}$$
$$\frac{2^k-1}{2^k} \geq n + \frac{n \cdot (2^k-1)}{2^k} \tag{21}$$
$$\frac{2^k-1}{2^k} \geq n \cdot \left(\frac{2^k - 2^k + 1}{2^k}\right) \tag{22}$$
$$2^k - 1 \geq n \tag{23}$$

3. Conclusion (combine parts together)

   Since $2^{k-1} \leq n$ and $n \leq 2^k - 1$ are true, we can conclude $2^{k-1} \leq n \leq 2^k - 1$ is true.

Part 2 (Proving in $\Leftarrow$ direction):

Assume $2^{k-1} \leq n \leq 2^k - 1$.

We need to show **convert_to_binary(n)** takes k step.

1. Show that with 2^{k-1} as n, **convert_to_binary(n)** does exactly k iterations.

For the lower bound of n, using the result of problem 4.a $\frac{n}{2^k} - \frac{2^k-1}{2^k} \leq x_k \leq \frac{n}{2^k}$, we can calculate that the value of x at $k-1^{th}$ iteration is

$$\frac{2^{k-1}}{2^{k-1}} - \frac{2^{k-1}-1}{2^{k-1}} \leq x_{k-1} \leq \frac{2^{k-1}}{2^{k-1}} \tag{24}$$
$$\frac{1}{2^{k-1}} \leq x_{k-1} \leq 1 \tag{25}$$

Since we know $\frac{1}{2^{k-1}} > 0$, and $x_{k-1} \in \mathbb{N}$ (from the code), we can conclude $x_{k-1} = 1$.

Then, by taking an iteration further, we can conclude

$$x_k = \left\lfloor \frac{x_{k-1}}{2} \right\rfloor \tag{26}$$
$$= 0 \tag{27}$$

Because we know loop termination occurs when $x \leq 0$, we can conclude the loop with the lower bound of n stops at k^{th} iteration.

2. Show that with $2^k - 1$ as n, **convert_to_binary(n)** does exactly k iterations.

For the upper bound of n, using the same result from problem 4.a, the value of x at k^{th} iteration is

$$\frac{2^k-1}{2^k} - \frac{2^k-1}{2^k} \leq x_k \leq \frac{2^k-1}{2^k} \tag{28}$$

$$0 \leq x_k \leq \frac{2^k-1}{2^k} \tag{29}$$

Since we know $\frac{2^k-1}{2^k} < 1$, and $x_k \in \mathbb{N}$ (from the code), we can conclude $x_k = 0$.

Because we know loop termination occurs when $x \leq 0$, we can conclude the loop with the upper bound of n stops at k^{th} iteration.

3. Conclude n performs k iterations.

So, since the loop stops at k^{th} iterations for both the upper and the lower bound of n, we can conclude n performs exactly k iterations.

c. Let $n \in \mathbb{Z}^+$, $k \in \mathbb{N}$ and let S_k denote the set of all numbers resulting in k many iterations in **convert_to_binary(n)**.

We need to evaluate the following expression

$$AVG_{convert\_to\_binary}(n) = \frac{1}{|\mathcal{I}_n|} \sum_{i \in \mathcal{I}_n} \text{Running time of convert\_to\_binary} \tag{1}$$

First, we need to show set of S_k over all k are partitions of $\mathcal{I}_n$. That is, the union of all of S_k form $\mathcal{I}_n$ and S_k over all k do not have any elements in common.

The question 4.b tells us

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, (\ \textbf{convert\_to\_binary(n)} \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \tag{2}$$

Using this fact, we can conclude S_k has highest element with the value of $2^k - 1$, and S_{k+1} has element with the lowest value of $2^{k+1-1} = 2^k$.

Then, we can calculate that

$$2^k - (2^k - 1) = 1 \tag{3}$$

Then, because we know the distance between the two sets have value greater than 0, we can conclude the two sets are non-overlapping, and do not have elements in common.

Now, we know S_k is the result of grouping elements in $\mathcal{I}_n$ by number of iterations k.

It follows from this fact that it's union form $\mathcal{I}_n$.

Second, we need to re-express equation 1 as sum of all elements in S_k over all iterations k.

Since we know

Third, we need to evaluate the number of input elements $|\mathcal{I}_n|$.

Because we know $\mathcal{I}_n$ has all integer elements from 1 to $2^n - 1$, we can conclude

$$\begin{align} |\mathcal{I}_n| &= 2^n - 1 - 1 + 1 \tag{4} \\ &= 2^n - 1 \tag{5} \end{align}$$

The qestion

Fourth, we need to find $|S_k|$.

The question 4.b tells us

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, \text{ (} \textbf{convert\_to\_binary(n)} \text{ takes exactly } k \text{ loop iterations)} \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \tag{6}$$

Using this fact, we can conclude S_k has values from 2^{k-1} to $2^k - 1$.

Then, we can calculate that

$$\begin{align} |S_k| &= 2^k - 1 - 2^K - 1 + 1 \tag{7} \\ &= 2^k - 2^{k-1} \tag{8} \\ &= 2^{k-1} \tag{9} \end{align}$$

Fifth, we need to determine the smallest and the largest value of k for the sum $\sum_k$.

We will do so in parts.

Part 1 (Finding the smallest value of k):

We need to find the smallest value of k.

The code tells us the value of k rises as n increases.

Because we know 1 is the smallest value in $\mathcal{I}_n$, we can conclude 1 is the value that will result in smallest value of k.

Now, the question 4.b tells us

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, \text{ (} \textbf{convert\_to\_binary(n)} \text{ takes exactly } k \text{ loop iterations)} \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \tag{10}$$

Because we know $1 = 2^{1-1}$, by using the fact, we can conclude $k = 1$.

Part 2 (Finding the largest value of k):

We need to find the largest value of k.

The code tells us the value of k rises as n increases.

Since the highest value in $\mathcal{I}_n$ is $2^n - 1$, we can conclude $2^n - 1$ is the value that will result in highest value of k.

Now, The question 4.b tells us

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, \text{ (} \textbf{convert\_to\_binary(n)} \text{ takes exactly } k \text{ loop iterations)} \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \tag{11}$$

Using this fact, we can conclude k has the highest value of n.

Finally, we need to evaluate the expression.

Rough Work:

1. Show set of S_k over all k are partitions of $\mathcal{I}_n$.
2. Re-express the average-case running time as sum over S_k.
   - Show $\sum_{i \in \mathcal{I}_n}$ is the same as $\sum_k \sum_{i \in S_k}$
3. Find the number of input elements $|\mathcal{I}_n|$.
   - Leave first and the last value of k as blank.
4. Find the value of $|S_k|$.
5. Find the first and last value of k.
6. Finish re-express the average-case running time as sum over S_k.
7. Evaluate S_k.

$$
\begin{aligned}
AVG_{\text{convert\_to\_binary}}(n) &= \frac{1}{|\mathcal{I}_n|} \cdot \sum_{i \in \mathcal{I}_n} \text{Runtime of convert\_to\_binary} && (12)\\
&= \frac{1}{2^n - 1} \cdot \sum_{i \in \mathcal{I}_n} \text{Runtime of convert\_to\_binary} && (13)\\
&= \frac{1}{2^n - 1} \cdot \sum_{k=1}^{n} \sum_{i \in S_k} k && (14)\\
&= \frac{1}{2^n - 1} \cdot \sum_{k=1}^{n} 2^{k-1} \cdot k && (15)\\
&= \frac{1}{2^n - 1} \cdot \left[\frac{1 - 2^{n+1}}{(1-2)^2} - \frac{(n+1)2^n}{1-2}\right] && (16)\\
&= \frac{1}{2^n - 1} \cdot \left[1 - 2^{n+1} + (n+1)2^n\right] && (17)\\
&= \frac{1}{2^n - 1} \cdot [1 - 2 \cdot 2^n + (n+1)2^n] && (18)\\
&= \frac{1}{2^n - 1} \cdot [1 - 2 \cdot 2^n + (n+1)2^n] && (19)\\
&= \frac{1}{2^n - 1} \cdot [1 + 2^n(n + 1 - 2)] && (20)\\
&= \frac{1}{2^n - 1} \cdot [1 + 2^n(n - 1)] && (21)
\end{aligned}
$$