

# Problem Set 4 Solution

Hyungmo Gu

April 14, 2020

## Question 1

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

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

*Proof.* Let  $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$ , 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

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

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

Then,

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

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

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

$$\log_b(g(n)) \leq \log_b(f(n)) \quad (6)$$

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

$$\log_b(g(n)) \leq d_1 \cdot \log_b(f(n)) \quad (7)$$

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

The assumption tells us

$$f(n) \leq c_2 \cdot g(n) \quad (8)$$

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

$$\log(f(n)) \leq \log(c_2 \cdot g(n)) \quad (9)$$

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

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

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

Then, since  $c_2 = b$ ,

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

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

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

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

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

$$\log_b(f(n)) \leq 2 \cdot \log_b(g(n)) \quad (15)$$

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

$$\log_b(f(n)) \leq d_2 \cdot \log_b(g(n)) \quad (16)$$

□

**Notes:**

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

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  $j_k \geq i$ .

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

$$\sum_{i=1}^n \lceil \log_3 i \rceil \quad (1)$$

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

$$\sum_{i'=0}^{\lceil \log_3 n \rceil} i' \quad (2)$$

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

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

$$= \frac{\lceil \log_3 n \rceil^2 + \lceil \log_3 n \rceil}{2} \quad (4)$$

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

### Correct Solution:

We need to determine  $\Theta$  of the algorithm.

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

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

$$\sum_{i=1}^n \lceil \log_3 i \rceil \quad (5)$$

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

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

$$\sum_{i=1}^n \log_3 i \leq \sum_{i=1}^n \lceil \log_3 i \rceil \leq \left( \sum_{i=1}^n \log_3 i + \sum_{i=1}^n 1 \right) \quad (7)$$

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

Then,

$$\log_3 \left( \prod_{i=1}^n i \right) \leq \sum_{i=1}^n \lceil \log_3 i \rceil \leq \log_3 \left( \prod_{i=1}^n i \right) + n \quad (9)$$

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

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

Then,

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

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  $\geq 1$ , 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  $f(n) = e^{n \ln n - n + \frac{1}{2} \ln n}$  and  $b = e$ , we can write

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

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

Then,

$$\ln(n!) \in \Theta(n \ln n) \quad (14)$$

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  $\Omega$  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 occurred 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

$$\left\lceil \frac{i - 1 - 0 + 1}{1} \right\rceil = i \quad (1)$$

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

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

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

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

$$1 \cdot (i - 1 - 0 + 1) = i \quad (1)$$

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

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

$$= \sum_{i=0}^n i + \sum_{i=0}^n 1 \quad (3)$$

$$= \sum_{i=0}^n i + (n + 1) \quad (4)$$

$$= \frac{n(n + 1)}{2} + (n + 1) \quad (5)$$

$$= \frac{(n + 1)(n + 2)}{2} \quad (6)$$

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  $list = [0, 0, \dots, 0, 1, 0, \dots, 0]$  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

$$k + 1 \geq \lceil \frac{n}{2} \rceil + 1 \quad (1)$$

$$k \geq \lceil \frac{n}{2} \rceil \quad (2)$$

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

$$\lceil \frac{n}{2} \rceil \quad (3)$$

iterations.

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

$$\lceil \frac{n}{2} \rceil \cdot 1 = \lceil \frac{n}{2} \rceil \quad (4)$$

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

$$k - 1 \leq \lceil \frac{n}{2} \rceil \quad (5)$$

$$k \leq \lceil \frac{n}{2} \rceil + 1 \quad (6)$$

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

$$\lceil \frac{n}{2} \rceil + 1 \quad (7)$$

iterations.

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

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

steps.

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

$$\left\lceil \frac{n}{2} \right\rceil^2 + 2 \cdot \left\lceil \frac{n}{2} \right\rceil + 2 \quad (9)$$

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

### **Correct Solution:**

Let  $n, k \in \mathbb{N}$ , and  $list = [0, 0, \dots, 0, 1, 0, \dots, 0]$  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

$$k \geq \left\lceil \frac{n}{2} \right\rceil + 1 \quad (1)$$

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

$$\left\lceil \frac{n}{2} \right\rceil + 1 \quad (2)$$

iterations.

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

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

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  $i_k \leq \left\lceil \frac{n}{2} \right\rceil$ .

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

$$n - k \leq \left\lceil \frac{n}{2} \right\rceil \quad (4)$$

$$-k \leq \left\lceil \frac{n}{2} \right\rceil - n \quad (5)$$

$$k \geq n - \left\lceil \frac{n}{2} \right\rceil \quad (6)$$

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

$$n - \left\lceil \frac{n}{2} \right\rceil \quad (7)$$

iterations.

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

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

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

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

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  $nums = [a_1, a_2, \dots, a_n]$  where  $a_1, \dots, a_n$  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

$$k \geq i \quad (1)$$

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  $nums = [1, a_2, a_3, \dots, a_{n-1}]$  where  $a_2, a_3, \dots, a_{n-1}$  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 tells 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

$$n - k \leq -1 \quad (5)$$

$$k \geq n + 1 \quad (6)$$

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

$$n + 1 \quad (7)$$

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

$$2 \cdot (n + 1) \quad (8)$$

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  $\Omega(n)$ .  $\square$

### **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  $nums = [a_1, a_2, \dots, a_n]$  where  $a_1, \dots, a_n$  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

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

$$> (n - m)(m + 2) + (m + 2) \tag{13}$$

$$= (n - m)m + 2(n - m) + (m + 2) \tag{14}$$

$$> (n - m)m + (n - m) + m \tag{15}$$

$$= (n - m)m + n \tag{16}$$

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

$$(n - m + 1)(m + 2) + 1 > n \quad (17)$$

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

$$2^k \geq n \quad (1)$$

$$k \geq \log n \quad (2)$$

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

$$\lceil \log n \rceil \quad (3)$$

iterations.

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

$$\lceil \log n \rceil \quad (4)$$

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  $lst[i] \geq 0$ : condition**, we can conclude that at  $k^{th}$  execution of **if  $lst[i] \geq 0$ : 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

$$2^k \geq n \quad (1)$$

$$k \geq \log n \quad (2)$$

Since we are looking for the smallest value of  $k$  (since it represents the number of executions caused by the **if  $lst[i] \geq 0$ : 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

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

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

$$\sum_{k=1}^{\log n - 2} \left( \frac{n}{2^{k+1}} + 1 \right) = \sum_{k=1}^{\log n - 2} \frac{n}{2^{k+1}} + \sum_{k=1}^{\log n - 2} 1 \quad (5)$$

$$= \frac{n}{2} \cdot \sum_{k=1}^{\log n - 2} \frac{1}{2^k} + \sum_{k=1}^{\log n - 2} 1 \quad (6)$$

$$= \frac{n}{2} \cdot \sum_{k=1}^{\log n - 2} \frac{1}{2^k} + (\log n - 2) \quad (7)$$

$$= \frac{n}{2} \cdot \left( \frac{1 - \frac{2}{2^{\log n}}}{1 - \frac{1}{2}} \right) + (\log n - 2) \quad (8)$$

$$= n \cdot \left( 1 - \frac{2}{n} \right) + (\log n - 2) \quad (9)$$

$$= n - 2 + \log n - 2 \quad (10)$$

$$= n + \log n - 4 \quad (11)$$

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

$$n + \log n - 4 + (\log n - 1) + 1 = n + 2 \log n - 4 \quad (12)$$

executions or iterations.

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

$$1 \cdot (n + 2 \log n - 4) = (n + 2 \log n - 4) \quad (13)$$

steps.

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

$$n + 2 \log n - 3 \quad (14)$$

which is  $\Theta(n)$ .

□

### Notes:

- 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 brach of statement executes, causing  $\mathbf{lst[i] = abs(lst[i])}$ ,  $i = 0$ , 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,

### 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  $\frac{n}{2} + 1$  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

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

- 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  $\text{lst}[i] = \text{abs}(\text{lst}[i])$ ,  $i = 0$ , 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  $\text{lst}[i] = \text{abs}(\text{lst}[i])$ ,  $i = 0$ , 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  $k + 1^{\text{th}}$  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

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

- 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  $\text{lst}[i] = \text{abs}(\text{lst}[i])$ ,  $i = 0$ , 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  $\text{lst}[i] = \text{abs}(\text{lst}[i])$ ,  $i = 0$ , 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  $\text{lst}[i] = \text{abs}(\text{lst}[i])$ ,  $i = 0$ , 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]

- **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.

#### 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  $j = 1$ , 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

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

- 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  $\mathcal{O}(n)$ .  $\square$

### Notes:

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



- 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

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

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

$$x_{k+1} = \left\lfloor \frac{x_k}{2} \right\rfloor \geq \frac{x_k - 1}{2} \quad (2)$$

$$= \frac{1}{2} \cdot (x_k - 1) \quad (3)$$

Then, by inductive hypothesis,

$$x_{k+1} \geq \frac{1}{2} \cdot \left( \frac{n}{2^k} - \frac{2^k - 1}{2^k} - 1 \right) \quad (4)$$

$$= \frac{n}{2^{k+1}} - \frac{2^k - 1}{2^{k+1}} - \frac{1}{2} \quad (5)$$

$$= \frac{n}{2^{k+1}} - \left( \frac{2^k - 1}{2^{k+1}} + \frac{2^k}{2^{k+1}} \right) \quad (6)$$

$$= \frac{n}{2^{k+1}} - \left( \frac{2^k + 2^k - 1}{2^{k+1}} \right) \quad (7)$$

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

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

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

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

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

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

$$x_{k+1} = \left\lfloor \frac{x_k}{2} \right\rfloor \leq \frac{x_k}{2} \quad (10)$$

Then, by the inductive hypothesis, we can conclude

$$x_{k+1} \leq \frac{n}{2^k \cdot 2} \quad (11)$$

$$\leq \frac{n}{2^{k+1}} \quad (12)$$

□

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

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

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

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  $\frac{n}{2^k} - \frac{2^k - 1}{2^k} \leq x_k \leq \frac{n}{2^k}$  from question 4.a, we can conclude

$$1 = x_{k-1} \leq \frac{n}{2^{k-1}} \quad (1)$$

$$2^{k-1} \leq n \quad (2)$$

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

$$n - \left( \frac{n}{2^k} - \frac{2^k-1}{2^k} \right) \geq x_0 - x_k = n \quad (3)$$

$$\frac{2^k \cdot n}{2^k} - \frac{n}{2^k} + \frac{2^k-1}{2^k} \geq n \quad (4)$$

$$\frac{n \cdot (2^k-1)}{2^k} + \frac{2^k-1}{2^k} \geq n \quad (5)$$

$$\frac{2^k-1}{2^k} \geq n + \frac{n \cdot (2^k-1)}{2^k} \quad (6)$$

$$\frac{2^k-1}{2^k} \geq n \cdot \left( \frac{2^k-2^k+1}{2^k} \right) \quad (7)$$

$$2^k-1 \geq n \quad (8)$$

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

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



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

$$\frac{2^{k-1}}{2^{k-1}} - \frac{2^{k-1} - 1}{2^{k-1}} \leq x_{k-1} \leq \frac{2^{k-1}}{2^{k-1}} \quad (10)$$

$$\frac{1}{2^{k-1}} \leq x_{k-1} \leq 1 \quad (11)$$

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

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

$$= 0 \quad (13)$$

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

$$\frac{2^k - 1}{2^k} - \frac{2^k - 1}{2^k} \leq x_k \leq \frac{2^k - 1}{2^k} \quad (14)$$

$$0 \leq x_k \leq \frac{2^k - 1}{2^k} \quad (15)$$

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.  $\square$

**Notes:**

- This is a tough problem.
- 형모 풀꼬얌!! 형모 궁덩궁덩 하고 한걸음씩 발전해쥬 대학원 갈꼬얌!!
- 오예!!! 형모 해낼꼬다!!
- 형모 화이팅!!
- After hours of thinking, I found the rough idea: find range of values between  $(x_1 \text{ and } x_k)$  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  $\Rightarrow$  direction, and the second is proving  $\Leftarrow$  direction).

#### **Part 1 (Proving $\Rightarrow$ 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  $x_k = \lfloor \frac{x_{k-1}}{2} \rfloor$  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

$$1 = x_{k-1} \leq \frac{n}{2^{k-1}} \quad (16)$$

$$2^{k-1} \leq n \quad (17)$$

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

$$n - \left( \frac{n}{2^k} - \frac{2^k-1}{2^k} \right) \geq x_0 - x_k = n \quad (18)$$

$$\frac{2^k \cdot n}{2^k} - \frac{n}{2^k} + \frac{2^k-1}{2^k} \geq n \quad (19)$$

$$\frac{n \cdot (2^k - 1)}{2^k} + \frac{2^k-1}{2^k} \geq n \quad (20)$$

$$\frac{2^k-1}{2^k} \geq n + \frac{n \cdot (2^k-1)}{2^k} \quad (21)$$

$$\frac{2^k-1}{2^k} \geq n \cdot \left( \frac{2^k-2^k+1}{2^k} \right) \quad (22)$$

$$2^k-1 \geq n \quad (23)$$

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

$$\frac{2^{k-1}}{2^{k-1}} - \frac{2^{k-1}-1}{2^{k-1}} \leq x_{k-1} \leq \frac{2^{k-1}}{2^{k-1}} \quad (24)$$

$$\frac{1}{2^{k-1}} \leq x_{k-1} \leq 1 \quad (25)$$

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

$$x_k = \left\lfloor \frac{x_{k-1}}{2} \right\rfloor \quad (26)$$

$$= 0 \quad (27)$$

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

$$\frac{2^k - 1}{2^k} - \frac{2^k - 1}{2^k} \leq x_k \leq \frac{2^k - 1}{2^k} \quad (28)$$

$$0 \leq x_k \leq \frac{2^k - 1}{2^k} \quad (29)$$

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

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

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

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

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

$$2^k - (2^k - 1) = 1 \quad (3)$$

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  $\mathcal{I}_n$ .

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

Second, 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

$$|\mathcal{I}_n| = 2^n - 1 - 1 + 1 \quad (4)$$

$$= 2^n - 1 \quad (5)$$

Third, we need to determine the smallest and the largest value of  $k$  of  $S_k$  in  $\mathcal{I}_n$

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

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

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

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, (\text{convert\_to\_binary}(n) \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \quad (7)$$

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

Fourth, we need to show  $|S_k| = 2^{k-1}$  for  $k = 1 \dots n$ .

We will do so using proof by cases.

**Case 1** ( $k = 1 \dots n - 1$ ):

In this case, we need to show  $|S_k| = 2^{k-1}$  for  $k = 1 \dots n - 1$ .

The question 4.b tells us

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, (\text{convert\_to\_binary}(n) \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \quad (8)$$

Because we know no elements are missing in  $S_k$ , by using this fact, we can calculate that

$$|S_k| = 2^k - 1 - 2^{k-1} + 1 \quad (9)$$

$$= 2^k - 2^{k-1} \quad (10)$$

$$= 2^{k-1} \quad (11)$$

**Case 2** ( $k = n$ ):

In this case, we need to show  $|S_k| = 2^{k-1}$  for  $k = n$ .

The question 4.b tells us

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, ( \text{convert\_to\_binary}(\mathbf{n}) \text{ takes exactly } k \text{ loop iterations} ) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \quad (12)$$

Using this fact, we know the value of last element in  $S_{n-1}$  is  $2^{n-1} - 1$ .

Because we know elements in  $\mathcal{I}_n$  increases by 1, we can conclude the value of first element in  $S_n$  is

$$2^{n-1} - 1 + 1 = 2^{n-1} \quad (13)$$

Now, because we know the first element in  $S_n$  is  $2^{n-1}$  and the last element in  $S_n$  is  $2^n - 1$ , using these fact, we can conclude

$$|S_k| = 2^{n-1} - (2^{n-1} - 1) + 1 \quad (14)$$

$$= 2^n - 2^{n-1} \quad (15)$$

$$= 2^{n-1} \quad (16)$$

$$= 2^{k-1} \quad (17)$$

Fifth, we need to evaluate the running time of **convert\_to\_binary**(**n**) for all elements in  $S_k$ .

The header tells us that elements in  $S_k$  result in loop with  $k$  iterations, and the code tells us each loop takes constant time (1 step).

Using these facts, we can calculate the loop has total time of

$$k \cdot 1 = k \quad (18)$$

steps.

Since we are ignoring the time of constant operations outside of the loop, we can conclude **convert\_to\_binary**(**n**) has running time of  $k$  steps.

Sixth, we need to re-express the average-case running time as sum over  $S_k$ .



Because we know the sets  $S_k$  over  $k = 1 \dots n$  are partitions of  $\mathcal{I}_n$ , we can conclude  $\sum_{i \in \mathcal{I}_n}$  is the same as  $\sum_{k=1}^n \sum_{i \in S_k}$ .

Using this fact, we can write

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{|\mathcal{I}_n|} \cdot \sum_{k=1}^n \sum_{i \in S_k} \text{Runtime of convert\_to\_binary} \quad (19)$$

Then, because we know all values in  $|S_k|$  has the same running time, we can write

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{|\mathcal{I}_n|} \cdot \sum_{k=1}^n |S_k| \cdot \text{Runtime of convert\_to\_binary} \quad (20)$$

Then, since we know  $|\mathcal{I}_n| = 2^n - 1$ , and  $|S_k| = 2^{k-1}$ , we can write

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{2^n - 1} \cdot \sum_{k=1}^n 2^{k-1} \cdot \text{Runtime of convert\_to\_binary} \quad (21)$$

Then, because we know all elements in  $S_k$  has runtime of  $k$ , we can conclude

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{2^n - 1} \cdot \sum_{k=1}^n 2^{k-1} \cdot k \quad (22)$$

Finally, we need to evaluate the average-case running time of **convert\_to\_binary(n)**.

The hint tells us

$$\forall m \in \mathbb{N}, \forall r \in \mathbb{R}, \sum_{i=1}^m i r^{i-1} = \frac{1 - r^{m+1}}{(1 - r)^2} - \frac{(m+1)r^m}{1 - r} \text{ where } r \neq 1 \quad (23)$$

Using the hint, we can conclude

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{2^n - 1} \cdot \sum_{k=1}^n 2^{k-1} \cdot k \quad (24)$$

$$= \frac{1}{2^n - 1} \cdot \left[ \frac{1 - 2^{n+1}}{(1 - 2)^2} - \frac{(n + 1)2^n}{1 - 2} \right] \quad (25)$$

$$= \frac{1}{2^n - 1} \cdot [1 - 2^{n+1} + (n + 1)2^n] \quad (26)$$

$$= \frac{1}{2^n - 1} \cdot [1 - 2 \cdot 2^n + (n + 1)2^n] \quad (27)$$

$$= \frac{1}{2^n - 1} \cdot [1 - 2 \cdot 2^n + (n + 1)2^n] \quad (28)$$

$$= \frac{1}{2^n - 1} \cdot [1 + 2^n(n + 1 - 2)] \quad (29)$$

$$= \frac{1}{2^n - 1} \cdot [1 + 2^n(n - 1)] \quad (30)$$

**Rough Work:**

1. Show set of  $S_k$  over all  $k$  are partitions of  $\mathcal{I}_n$ .

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

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, (\text{convert\_to\_binary}(\mathbf{n}) \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \quad (31)$$

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

$$2^k - (2^k - 1) = 1 \quad (32)$$

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  $\mathcal{I}_n$ .

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

2. Find the number of input elements  $|\mathcal{I}_n|$ .

Second, 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

$$|\mathcal{I}_n| = 2^n - 1 - 1 + 1 \quad (33)$$

$$= 2^n - 1 \quad (34)$$

3. Find the first and last value of  $k$  of  $S_k$  in  $\mathcal{I}_n$ .

Third, we need to determine the smallest and the largest value of  $k$  of  $S_k$  in  $\mathcal{I}_n$

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

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, (\text{convert\_to\_binary}(n) \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \quad (35)$$

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

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, (\text{convert\_to\_binary}(n) \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \quad (36)$$

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

4. Show the number of  $|S_k| = 2^{k-1}$ .

Fourth, we need to show  $|S_k| = 2^{k-1}$  for  $k = 1 \dots n$ .

We will do so using proof by cases.

1. Case 1 ( $k = 1 \dots n - 1$ ):

In this case, we need to show  $|S_k| = 2^{k-1}$  for  $k = 1 \dots n - 1$ .

The question 4.b tells us

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, (\text{convert\_to\_binary}(\mathbf{n}) \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \quad (37)$$

Because we know no elements are missing in  $S_k$ , by using this fact, we can calculate that

$$|S_k| = 2^k - 1 - 2^{k-1} + 1 \quad (38)$$

$$= 2^k - 2^{k-1} \quad (39)$$

$$= 2^{k-1} \quad (40)$$

2. Case 2 ( $k = n$ ):

In this case, we need to show  $|S_k| = 2^{k-1}$  for  $k = n$ .

The question 4.b tells us

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, (\text{convert\_to\_binary}(\mathbf{n}) \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \quad (41)$$

Using this fact, we know the value of last element in  $S_{n-1}$  is  $2^{n-1} - 1$ .

Because we know elements in  $\mathcal{I}_n$  increases by 1, we can conclude the value of first element in  $S_n$  is

$$2^{n-1} - 1 + 1 = 2^{n-1} \quad (42)$$

Now, because we know the first element in  $S_n$  is  $2^{n-1}$  and the last element in  $S_n$  is  $2^n - 1$ , using these fact, we can conclude

$$|S_k| = 2^{n-1} - (2^{n-1} - 1) + 1 \quad (43)$$

$$= 2^n - 2^{n-1} \quad (44)$$

$$= 2^{n-1} \quad (45)$$

$$= 2^{k-1} \quad (46)$$

Fourth, we need to show  $|S_k| = 2^{k-1}$  for  $k = 1 \dots n$ .

We will do so using proof by cases.

**Case 1** ( $k = 1 \dots n - 1$ ):

In this case, we need to show  $|S_k| = 2^{k-1}$  for  $k = 1 \dots n - 1$ .

The question 4.b tells us

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, (\text{convert\_to\_binary}(n) \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \quad (47)$$

Because we know no elements are missing in  $S_k$ , by using this fact, we can calculate that

$$|S_k| = 2^k - 1 - 2^{k-1} + 1 \quad (48)$$

$$= 2^k - 2^{k-1} \quad (49)$$

$$= 2^{k-1} \quad (50)$$

**Case 2** ( $k = n$ ):

In this case, we need to show  $|S_k| = 2^{k-1}$  for  $k = n$ .

The question 4.b tells us

$$\forall n \in \mathbb{Z}^+, \forall k \in \mathbb{N}, (\text{convert\_to\_binary}(n) \text{ takes exactly } k \text{ loop iterations}) \Leftrightarrow 2^{k-1} \leq n \leq 2^k - 1 \quad (51)$$

Using this fact, we know the value of last element in  $S_{n-1}$  is  $2^{n-1} - 1$ .

Because we know elements in  $\mathcal{I}_n$  increases by 1, we can conclude the value of first element in  $S_n$  is

$$2^{n-1} - 1 + 1 = 2^{n-1} \quad (52)$$

Now, because we know the first element in  $S_n$  is  $2^{n-1}$  and the last element in  $S_n$  is  $2^n - 1$ , using these fact, we can conclude

5. Evaluate the running time of **convert\_to\_binary(n)** for all elements in  $S_k$ .

Fifth, we need to evaluate the running time of **convert\_to\_binary(n)** for all elements in  $S_k$ .

- State that elements in  $S_k$  result in loop with  $k$  iterations, and that each loop in **convert\_to\_binary(n)** takes a constant time (1 step)

The header tells us that elements in  $S_k$  result in loop with  $k$  iterations, and the code tells us each loop takes a constant time (1 step).

- Show the loop has total time of  $k$  steps

Using these facts, we can calculate the loop has total time of

$$k \cdot 1 = k \quad (57)$$

steps.

- Show **convert\_to\_binary(n)** has running time of  $k$  steps.

Since we are ignoring the time of constant operations outside of the loop, we can conclude **convert\_to\_binary(n)** has running time of  $k$  steps.

Fifth, we need to evaluate the running time of **convert\_to\_binary(n)** for all elements in  $S_k$ .

The header tells us that elements in  $S_k$  result in loop with  $k$  iterations, and the code tells us each loop takes constant time (1 step).

Using these facts, we can calculate the loop has total time of

$$k \cdot 1 = k \quad (58)$$

steps.

Since we are ignoring the time of constant operations outside of the loop, we can conclude **convert\_to\_binary(n)** has running time of  $k$  steps.



6. Re-express the average-case running time as sum over  $S_k$ .

Sixth, we need to re-express the average-case running time as sum over  $S_k$ .

Sixth, we need to re-express the average-case running time as sum over  $S_k$ .

Because we know the sets  $S_k$  over  $k = 1 \dots n$  are partitions of  $\mathcal{I}_n$ , we can conclude  $\sum_{i \in \mathcal{I}_n}$  is the same as  $\sum_{k=1}^n \sum_{i \in S_k}$ .

Using this fact, we can write

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{|\mathcal{I}_n|} \cdot \sum_{k=1}^n \sum_{i \in S_k} \text{Runtime of convert\_to\_binary} \quad (59)$$

Then, because we know all values in  $|S_k|$  has the same running time, we can write

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{|\mathcal{I}_n|} \cdot \sum_{k=1}^n |S_k| \cdot \text{Runtime of convert\_to\_binary} \quad (60)$$

Then, since we know  $|\mathcal{I}_n| = 2^n - 1$ , and  $|S_k| = 2^{k-1}$ , we can write

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{2^n - 1} \cdot \sum_{k=1}^n 2^{k-1} \cdot \text{Runtime of convert\_to\_binary} \quad (61)$$

Then, because we know all elements in  $S_k$  has runtime of  $k$ , we can conclude

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{2^n - 1} \cdot \sum_{k=1}^n 2^{k-1} \cdot k \quad (62)$$

7. Evaluate the average case running time.

Finally, we need to evaluate the average-case running time of **convert\_to\_binary(n)**.

- State the hint, and the average-case running time

The hint tells us

$$\forall m \in \mathbb{N}, \forall r \in \mathbb{R}, \sum_{i=1}^m ir^{i-1} = \frac{1-r^{m+1}}{(1-r)^2} - \frac{(m+1)r^m}{1-r} \text{ where } r \neq 1 \quad (63)$$

- Evaluate the expression using the hint

Using the hint, we can conclude

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{2^n - 1} \cdot \sum_{k=1}^n 2^{k-1} \cdot k \quad (64)$$

$$= \frac{1}{2^n - 1} \cdot \left[ \frac{1 - 2^{n+1}}{(1-2)^2} - \frac{(n+1)2^n}{1-2} \right] \quad (65)$$

$$= \frac{1}{2^n - 1} \cdot [1 - 2^{n+1} + (n+1)2^n] \quad (66)$$

$$= \frac{1}{2^n - 1} \cdot [1 - 2 \cdot 2^n + (n+1)2^n] \quad (67)$$

$$= \frac{1}{2^n - 1} \cdot [1 - 2 \cdot 2^n + (n+1)2^n] \quad (68)$$

$$= \frac{1}{2^n - 1} \cdot [1 + 2^n(n+1-2)] \quad (69)$$

$$= \frac{1}{2^n - 1} \cdot [1 + 2^n(n-1)] \quad (70)$$

Finally, we need to evaluate the average-case running time of **convert\_to\_binary(n)**.

The hint tells us

$$\forall m \in \mathbb{N}, \forall r \in \mathbb{R}, \sum_{i=1}^m ir^{i-1} = \frac{1-r^{m+1}}{(1-r)^2} - \frac{(m+1)r^m}{1-r} \text{ where } r \neq 1 \quad (71)$$

Using the hint, we can conclude

$$AVG_{\text{convert\_to\_binary}}(n) = \frac{1}{2^n - 1} \cdot \sum_{k=1}^n 2^{k-1} \cdot k \quad (72)$$

$$= \frac{1}{2^n - 1} \cdot \left[ \frac{1 - 2^{n+1}}{(1-2)^2} - \frac{(n+1)2^n}{1-2} \right] \quad (73)$$

$$= \frac{1}{2^n - 1} \cdot [1 - 2^{n+1} + (n+1)2^n] \quad (74)$$

$$= \frac{1}{2^n - 1} \cdot [1 - 2 \cdot 2^n + (n+1)2^n] \quad (75)$$

$$= \frac{1}{2^n - 1} \cdot [1 - 2 \cdot 2^n + (n+1)2^n] \quad (76)$$

$$= \frac{1}{2^n - 1} \cdot [1 + 2^n(n+1-2)] \quad (77)$$

$$= \frac{1}{2^n - 1} \cdot [1 + 2^n(n-1)] \quad (78)$$

## Notes:

- Realized the solution to this problem feels something like this

$$A(x_1) \wedge B(x_2) \wedge \dots \wedge P(x_n) \Rightarrow Q(x_m) \Rightarrow R(x_o)$$

- Realized each large part of proof (i.e  $P(x_n)$ ) must be provable and logically structured (i.e.  $P(x_n) : X(x_n) \Rightarrow Y(x_n)$ ).

For example, the part about finding the first and last value of  $k$  in  $S_k$  should have been the proof of showing the first value of  $k$  is 1, and showing the last value of  $k$  is  $n$ .

Another example is work on finding the number of input elements  $|\mathcal{I}_n|$ . this should have been the proof of showing the number of input elements  $|\mathcal{I}_n|$  is  $2^n - 1$ .

- Realized I should have used sub-header like **Part 1**, **Part 2**, ... to make proof more readable.
- 따뜻한 내 여보 향해 한걸음 더!!!!
- 울지마 형모야.
- 주저 앞지마 형모야.
- 할 수 있어.
- 고마워요