Problem Set 4 Solution

Hyungmo Gu

April 9, 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

$$\begin{aligned} 3^k &\geq i && (1) \\ k &\geq \log_3 i && (2) \end{aligned}$$

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

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

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

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

Then,

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

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

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

Then,

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

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 $\ge e$.

Since $n!$ is also eventually $\ge e$, 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

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

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 $\Omega(n^2)$

<u>Correct Solution:</u>

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

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

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 $nums = [a_1, a_2, \ldots, a_n]$ 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

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

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

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

- 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 $n, k \in \mathbb{N}$ 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 $\Theta(\log n)$. $\square$

Question 4