

# Table of Contents

**Question 1:**

---

**Question 2:**

---

- (a)
- (b)
- (c)
- (d)
- (e)

**Question 3:**

---

- (a)
- (b)
- (c)

## Question 1.

Let  $n$  = num of elements in array,

$m$  = size of array

Let  $\phi(h) = 4n - 2m$

$\phi(h_0) = 4(0) - 2(0) = 0$  since we start off with no elements and an empty array

$\phi(h_i) \geq 0$

This is because the array is resized after there is more than  $m$  elements. So after the resize, there will be  $n + 1$  elements, with the array of size  $\lceil 1.5n \rceil$ .  $\lceil 4(n + 1) \rceil \geq 2 \lceil 1.5n \rceil$ . For any operation after that, the number of elements increases, but the size of array stays the same.

Appending an element:

Case 1:  $n < m$

The actual cost is 1, since only one element is added to the array.

$$\begin{aligned}\Delta\phi(h) &= 4(n + 1) - 2(m) - (4n - 2m) \\ &= 4\end{aligned}$$

Therefore amortized time =  $1 + 4 \in \mathcal{O}(1)$

Case 2:  $n = m$

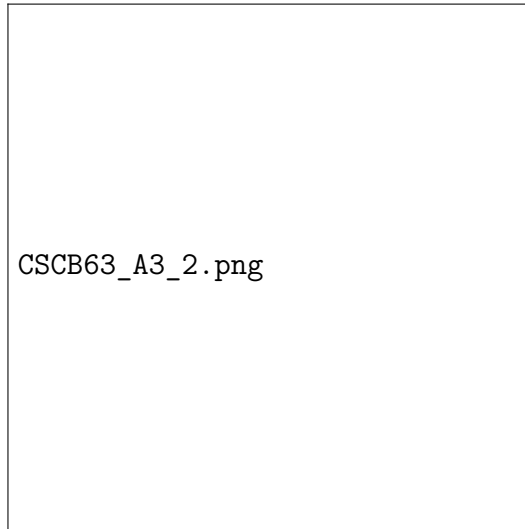
The actual cost is  $n + 1$ , since the entire array is copied plus the new element.

$$\begin{aligned}\Delta\phi(h) &= 4(n + 1) - 2 \lceil 1.5n \rceil - (4n - 2n) \\ &= 4 - 2 \lceil 1.5n \rceil + 2n \\ &= \begin{cases} 4 - 3n + 2n & \text{if } n \text{ is even} \\ 4 - (3n + 1) + 2n & \text{if } n \text{ is odd} \end{cases} \\ &= \begin{cases} 4 - n & \text{if } n \text{ is even} \\ 3 - n & \text{if } n \text{ is odd} \end{cases}\end{aligned}$$

$$\begin{aligned}\text{Amortized Time} &= \begin{cases} n + 4 - n & \text{if } n \text{ is even} \\ n + 3 - n & \text{if } n \text{ is odd} \end{cases} \\ &= \begin{cases} 4 & \text{if } n \text{ is even} \\ 3 & \text{if } n \text{ is odd} \end{cases} \\ &\in \mathcal{O}(1)\end{aligned}$$

## Question 2.

(a)



(b)

Let  $I_n$  denote the set of bit positions in the binary representation of  $n$  that contain the value of 1. Then in the data structure, those positions in  $I_n$  will also be where the elements are in the data structure.

Let  $I_n = \{j_1, j_2, \dots, j_p\}$

The worst case for **SEARCH(k)** is if  $\mathbf{k}$  is not in the data structure. So the total time complexity is:

$$\sum_{i=1}^p (\log_2(2^{j_i}) + 1) = \left( \sum_{i=1}^p j_i \right) + p$$

(c)

The worst case for **INSERT(k)** is if all  $j_i \in I_n$  needs to be merged. So the total time complexity is:

$$\sum_{i=1}^p 2^{j_i} = n$$

(d)

\*Assuming creating an array of size 1, and inserting it to the beginning of the linked list costs 1. Merging 2 arrays costs the sum of the 2 array sizes.

In the following table, we will have the insertion vs total costs.

Number of insertions	Cost of Current Operation	Total Cost	$\text{floor}(n * \log_2(n) + 2*n)$
1	2	2	2
2	4	6	6
3	2	8	10
4	8	16	16
5	2	18	21
6	4	22	27
7	2	24	33
8	16	40	40
9	2	42	46
10	4	46	53
11	2	48	60
12	8	56	67
13	2	58	74
14	4	62	81
15	2	64	88
16	32	96	96

It is clear that the total cost is always  $\leq n \log_2(n) + 2n$

$$\begin{aligned}
 \text{Total Amortized Cost} &= \frac{n \log_2(n) + 2n}{n} \\
 &= \log_2(n) + 2 \\
 &\in \mathcal{O}(\log_2 n)
 \end{aligned}$$

(e)

Lets charge for each operation  $\lfloor \log_2 n \rfloor + 4$ .

Proving total credits  $\geq 0$ :

Insertions	Cost	Credits Charged	Credits Left
1	2	4	2
2	4	5	3
3	2	5	6
4	8	6	4
5	2	6	8
6	4	6	10
7	2	6	14
8	16	7	5
9	2	7	10
10	4	7	13
11	2	7	18
12	8	7	17
13	2	7	22
14	4	7	25
15	2	7	30
16	32	8	6

For all credits left, it is always  $\geq \log_2(n) + 2$

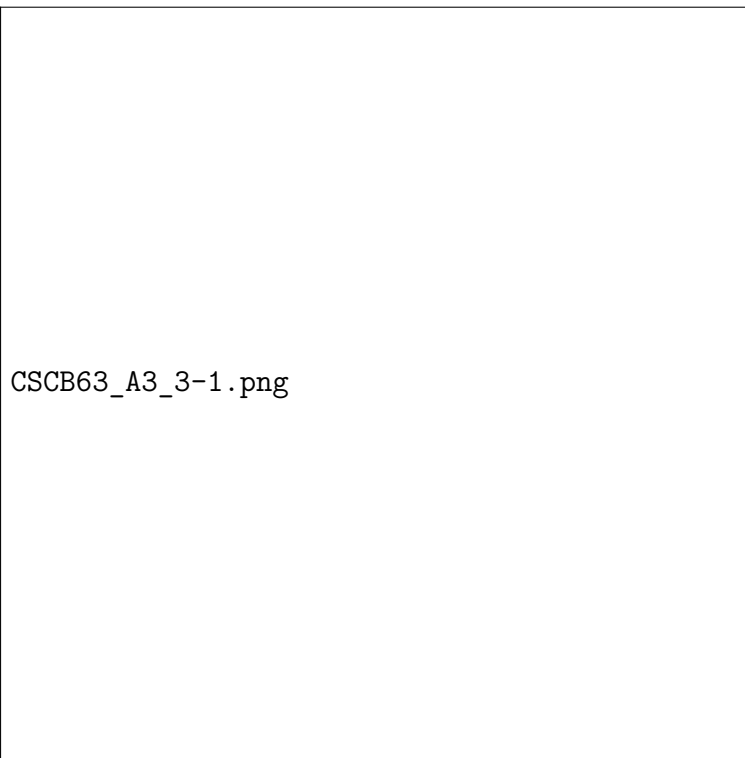
Which is always  $\geq 0$  for all  $n \in \mathbb{N}^+$

Since we charge  $\lfloor \log_2 n \rfloor + 4$  for each operation, therefore the Total Amortized Cost  $\in \mathcal{O}(\log_2 n)$

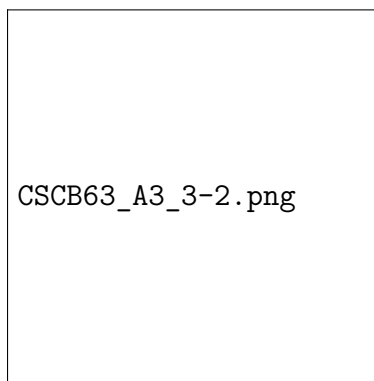
## Question 3.

(a)

The forest after performing operations:  $\text{UNION}(5, 2)$ ,  $\text{MAKESET}(3)$ ,  $\text{MAKESET}(4)$ ,  $\text{UNION}(3, 4)$ ,  
 $\text{UNION}(5, 3)$ ,  $\text{MAKESET}(1)$ ,  $\text{MAKESET}(8)$ ,  $\text{UNION}(1, 8)$ ,  $\text{FINDSET}(8)$ ,



The forest after performing operation:  $\text{UNION}(4, 1)$ .



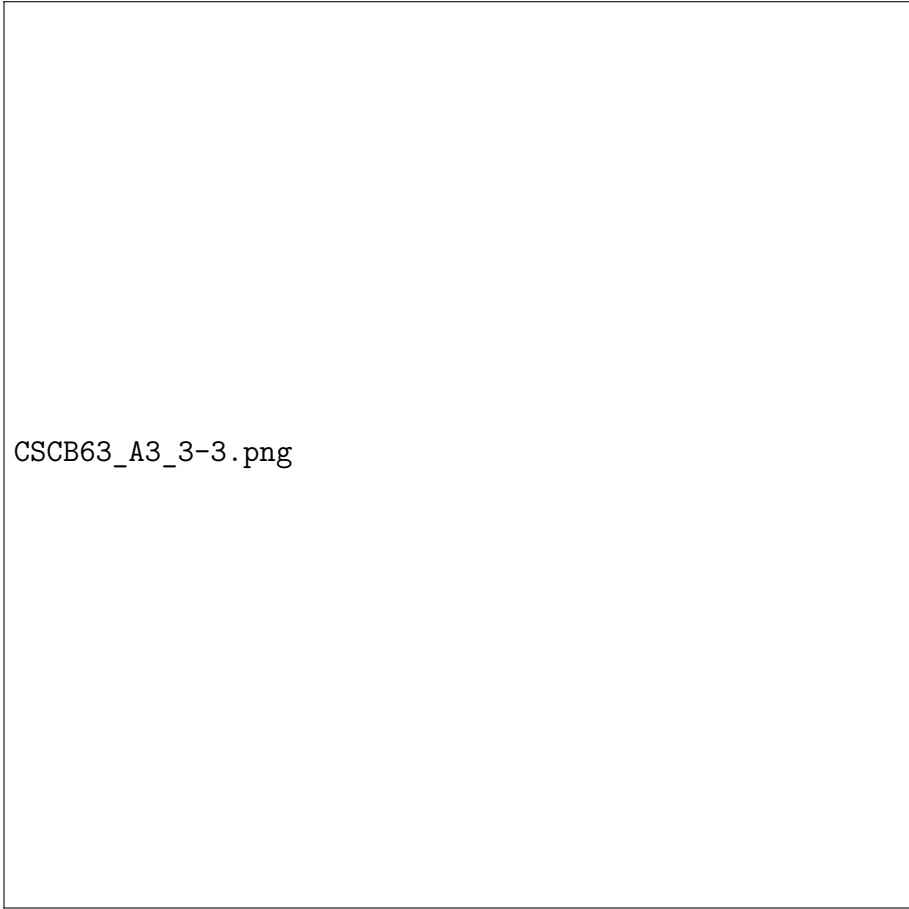
(b)

Yes.

Let  $P = \text{MAKESET}(1), \text{MAKESET}(2), \text{UNION}(1,2), \text{MAKESET}(3), \text{UNION}(1,3)$

Up until  $\text{MAKESET}(3)$ , The same number of nodes are accessed.

For  $\text{UNION}(1,3)$ , the linked list must traverse through the node (1) and (2) to insert the node (3). For the tree, node (3) is directly pointed to node (1). This makes it so that the linked list accesses 1 extra time, which is node (2).

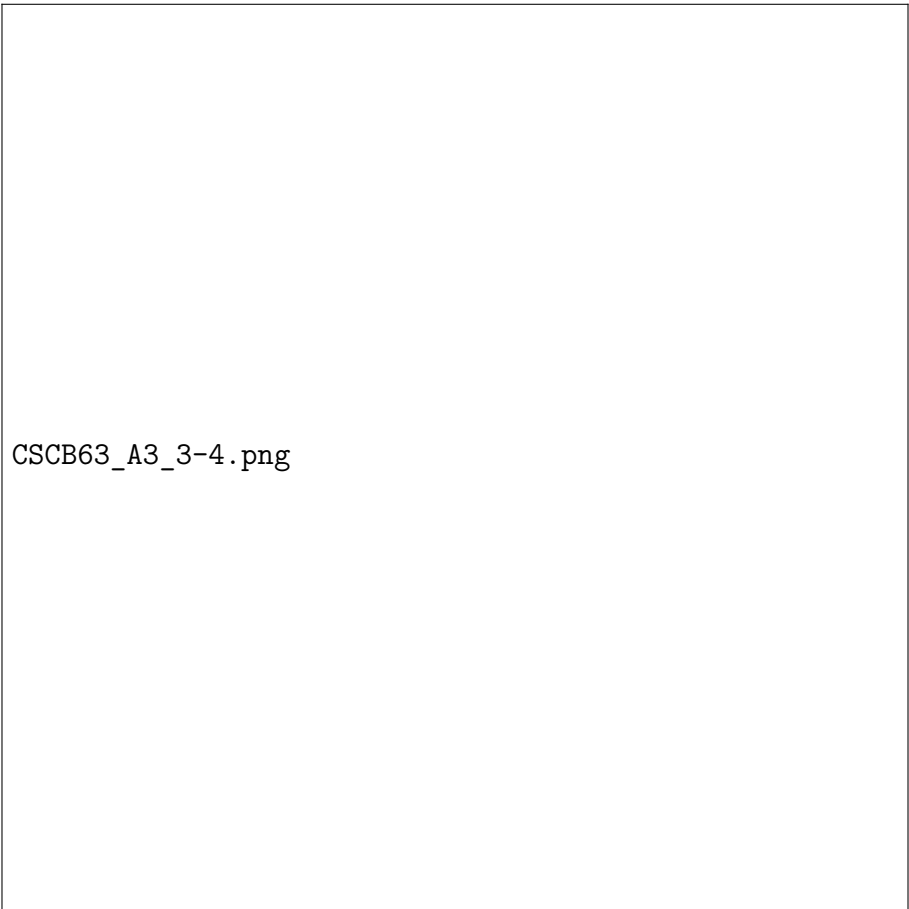


CSCB63\_A3\_3-3.png

(c)

Yes.

Let  $P = \text{MAKESET}(5), \text{MAKESET}(2), \text{UNION}(5, 2), \text{MAKESET}(3), \text{MAKESET}(4), \text{UNION}(3, 4), \text{UNION}(2, 3), \text{FIND}(4)$ .



CSCB63\_A3\_3-4.png

Up until  $\text{UNION}(2, 3)$ , The tree has 1 less access than the linked list. This is because the linked list needs to update nodes (2), (3), and (4). For the tree, it just needs to access node (5) and (3).

After  $\text{FIND}(4)$ , the Tree needs to perform path compression, which requires access to nodes (4), (3), and (5). However for the linked list, It only needs to access node (4). Thus, the tree will have accessed 1 more node than the linked list.