PSO 3

Induction

justin-zhang.com/teaching/CS251

Question 1

For this problem we will consider the following algorithm which computes n^x

```
function POWER(n : Z_{>0}, x : Z_{≥0})
    if x = 0 then
        return 1
    end if
    if x = 1 then
        return n
    end if
    temp ← 1
    if x is odd then
        temp ← n
        temp ← temp × POWER(n, (x − 1)/2)
        return temp × POWER(n, (x − 1)/2)
    end if
    temp ← temp × POWER(n, x/2)
    return temp × POWER(n, x/2)
end function
```

(a) Use induction to prove that temp always outputs n^x for any integers $x \geq 0$ and $n > 0$. **Hint:** Do we want to induct on x or do we want to induct on n?

Recursion and Induction are *strongly linked!*

Proposition: `fib`(n) = nth fib. number

```
fun fib(n) =
    if n = 0: return 0
    if n = 1: return 1
    return fib(n - 1) + fib (n - 2)
```

My function structure should mirror my proof structure

Recursion and Induction are *strongly linked!*

```
fun fib(n) =
    if n = 0: return 0      (Base case)
    if n = 1: return 1
    return fib(n - 1) + fib (n - 2)
```

Proposition: `fib`(n) = nth fib. number
Base case: (n = 0,1)

...

My function structure should mirror my proof structure

Recursion and Induction are *strongly linked!*

```
fun fib(n) =
    if n = 0: return 0
    if n = 1: return 1
    return fib(n - 1) + fib (n - 2)
```

Base case

Recursive case

Proposition: `fib`(n) = nth fib. number

Base case: (n = 0,1)

...

Inductive step: Suppose n > 1. We want to show `fib`(n) = nth fib. Number

IH?

My function structure should mirror my proof structure

Recursion and Induction are *strongly linked!*

Base case

Recursive case

```
fun fib(n) =
    if n = 0: return 0
    if n = 1: return 1
    return fib(n - 1) + fib (n - 2)
```

We need an IH for both (n - 1) AND (n - 2)

Proposition: `fib`(n) = nth fib. number

Base case: (n = 0,1)

...

Inductive step: Suppose n > 1. We want to show `fib`(n) = nth fib. Number

IH: Assume that

1. **`fib(n-1)` = (n-1)fib**
2. **`fib(n-2)` = (n - 2)th fib.**

Note: we could have used *strong* induction e.g ______________________

Question 1

For this problem we will consider the following algorithm which computes n^x

```
function POWER(n : Z_{>0}, x : Z_{≥0})
    if x = 0 then
        return 1
    end if
    if x = 1 then
        return n
    end if
    temp ← 1
    if x is odd then
        temp ← n
        temp ← temp × POWER(n, (x − 1)/2)
        return temp × POWER(n, (x − 1)/2)
    end if
    temp ← temp × POWER(n, x/2)
    return temp × POWER(n, x/2)
end function
```

(a) Use induction to prove that temp always outputs n^x for any integers $x \geq 0$ and $n > 0$. **Hint:** Do we want to induct on x or do we want to induct on n?

n or x? (hint: function structure motivates induction)

Inducting on x (fix the other variable n)

```
function POWER(n : Z>0, x : Z≥0)
    if x = 0 then
        return 1
    end if
    if x = 1 then
        return n
    end if
    temp ← 1
    if x is odd then
        temp ← n
        temp ← temp × POWER(n, (x − 1)/2)
        return temp × POWER(n, (x − 1)/2)
    end if
    temp ← temp × POWER(n, x/2)
    return temp × POWER(n, x/2)
end function
```

Proposition: `Power`(n) = n^x

Base case:

Inductive step:

IH:

Let's first label the base case and recursive case

Inducting on x

```
function POWER(n : Z_{>0}, x : Z_{≥0})
    if x = 0 then
        return 1
    end if
    if x = 1 then
        return n
    end if
    temp ← 1
    if x is odd then
        temp ← n
        temp ← temp × POWER(n, (x − 1)/2)
        return temp × POWER(n, (x − 1)/2)
    end if
    temp ← temp × POWER(n, x/2)
    return temp × POWER(n, x/2)
end function
```

Base case

Recursive case

Proposition: `Power`(n) = n^x

Base case: $n^0 = 1$, $n^1 = n$

Inductive step:

IH:

Inducting on x

```
function POWER(n : Z_{>0}, x : Z_{≥0})
    if x = 0 then
        return 1
    end if
    if x = 1 then
        return n
    end if
    temp ← 1
    if x is odd then
        temp ← n
        temp ← temp × POWER(n, (x − 1)/2)
        return temp × POWER(n, (x − 1)/2)
    end if
    temp ← temp × POWER(n, x/2)
    return temp × POWER(n, x/2)
end function
```

Base case

Recursive case

Proposition: `Power`(n) = n^x

Base case: $n^0 = 1$, $n^1 = n$

Inductive step: Suppose x > 1. We want to show `Power`(n) = n^x

IH:

(IH here is a bit more tricky..)

Inducting on x

```
function Power(n : Z_{>0}, x : Z_{≥0})
    if x = 0 then
        return 1
    end if
    if x = 1 then
        return n
    end if
    temp ← 1
    if x is odd then
        temp ← n
        temp ← temp × Power(n, (x − 1)/2)
        return temp × Power(n, (x − 1)/2)
    end if
    temp ← temp × Power(n, x/2)
    return temp × Power(n, x/2)
end function
```

Base case

Recursive case

Proposition: `Power`(n,x) = n^x

Base case: $n^0 = 1$, $n^1 = n$

Inductive step: Suppose x > 1. We want to show `Power`(n) = n^x

IH: Assume `Power`(n,x') = $n^{x'}$ **for all x' < x**

How do I proceed with my proof?

Inducting on x

```
function POWER(n : Z_{>0}, x : Z_{≥0})
    if x = 0 then
        return 1
    end if
    if x = 1 then
        return n
    end if
    temp ← 1
    if x is odd then
        temp ← n
        temp ← temp × POWER(n, (x − 1)/2)
        return temp × POWER(n, (x − 1)/2)
    end if
    temp ← temp × POWER(n, x/2)
    return temp × POWER(n, x/2)
end function
```

Base case

Recursive case

Proposition: `Power`(n,x) = n^x

Base case: $n^0 = 1$, $n^1 = n$

Inductive step: Suppose x > 1. We want to show `Power`(n) = n^x

IH: Assume `Power`(n,x') = $n^{x'}$ for all x' < x

PROOF STRUCTURE == CODE STRUCTURE

Case 1: x odd

Case 2: x even

(b) Let $T(x)$ denote the total number of multiplication operations when we compute POWER(n, x) and $n \neq 0$. Write down a recurrence relationship for $T(x)$.

```
function POWER(n : Z>0, x : Z≥0)
    if x = 0 then
        return 1
    end if
    if x = 1 then
        return n
    end if
    temp ← 1
    if x is odd then
        temp ← n
        temp ← temp × POWER(n, (x − 1)/2)
        return temp × POWER(n, (x − 1)/2)
    end if
    temp ← temp × POWER(n, x/2)
    return temp × POWER(n, x/2)
end function
```

Base case

Recursive case

RECURRENCE
~~PROOF~~ STRUCTURE == CODE STRUCTURE

Case 1: x odd

Case 2: x even

(c) Solve your recurrence relationship to find $T(x)$. Express your answer using big Θ notation.

$T(n) = 2T(n/2) + 2$

(d) Modify the recursive algorithm POWER so that it is more efficient. What is the new recurrence relationship for $T(x)$? What does it solve to?

```
function POWER(n : Z_{>0}, x : Z_{≥0})
    if x = 0 then
        return 1
    end if
    if x = 1 then
        return n
    end if
    temp ← 1
    if x is odd then
        temp ← n
        temp ← temp × POWER(n, (x − 1)/2)
        return temp × POWER(n, (x − 1)/2)
    end if
    temp ← temp × POWER(n, x/2)
    return temp × POWER(n, x/2)
end function
```

Question 1

(Linked List) Consider a sorted circular doubly linked list of N numbers where the head element points to the smallest element in the list. Provide the asymptotic complexity in big-Θ with a brief explanation (including assumptions and analysis for each case, if there is more than one) for the following operations.

1. Inserting an element in its sorted position.

Question 1

(Linked List) Consider a sorted circular doubly linked list of N numbers where the head element points to the smallest element in the list. Provide the asymptotic complexity in big-Θ with a brief explanation (including assumptions and analysis for each case, if there is more than one) for the following operations.

2. Finding the smallest element in the list.

Question 1

(Linked List) Consider a sorted circular doubly linked list of N numbers where the head element points to the smallest element in the list. Provide the asymptotic complexity in big-Θ with a brief explanation (including assumptions and analysis for each case, if there is more than one) for the following operations.

3. Finding the 3^{rd} - largest element in the list.

Question 1

(Linked List) Consider a sorted circular doubly linked list of N numbers where the head element points to the smallest element in the list. Provide the asymptotic complexity in big-Θ with a brief explanation (including assumptions and analysis for each case, if there is more than one) for the following operations.

4. Finding the median in the list.

Question 1

(Linked List) Consider a sorted circular doubly linked list of N numbers where the head element points to the smallest element in the list. Provide the asymptotic complexity in big-Θ with a brief explanation (including assumptions and analysis for each case, if there is more than one) for the following operations.

1. Inserting an element in its sorted position.
2. Finding the smallest element in the list.
3. Finding the 3^{rd} - largest element in the list.
4. Finding the median in the list.

Question 3

The Josephus Problem is a theoretical puzzle based on a historical account from the Jewish historian Flavius Josephus during the Jewish-Roman war. According to the story, Josephus and his 40 soldiers were trapped in a cave, with enemy soldiers outside. Preferring suicide to capture, they decided to form a circle and, proceeding around it, to kill every kth person until no one was left. Josephus, preferring to surrender to the Romans rather than die, figured out where he needed to sit to be the last survivor. This problem asks you to compute the position Josephus should choose to avoid being killed, given the number of people in the circle (n) and the step rate (k).

Input: The total number of people n in the circle and a number k which indicates that every kth person will be killed.

Output: The position in which Josephus should sit to be the last survivor.

First, create a Circular Linked List: Represent the people in a circle using a circular linked list where each node represents a person. The last person's next pointer points back to the first person, forming a circle.

Next, simulate the Elimination Process:

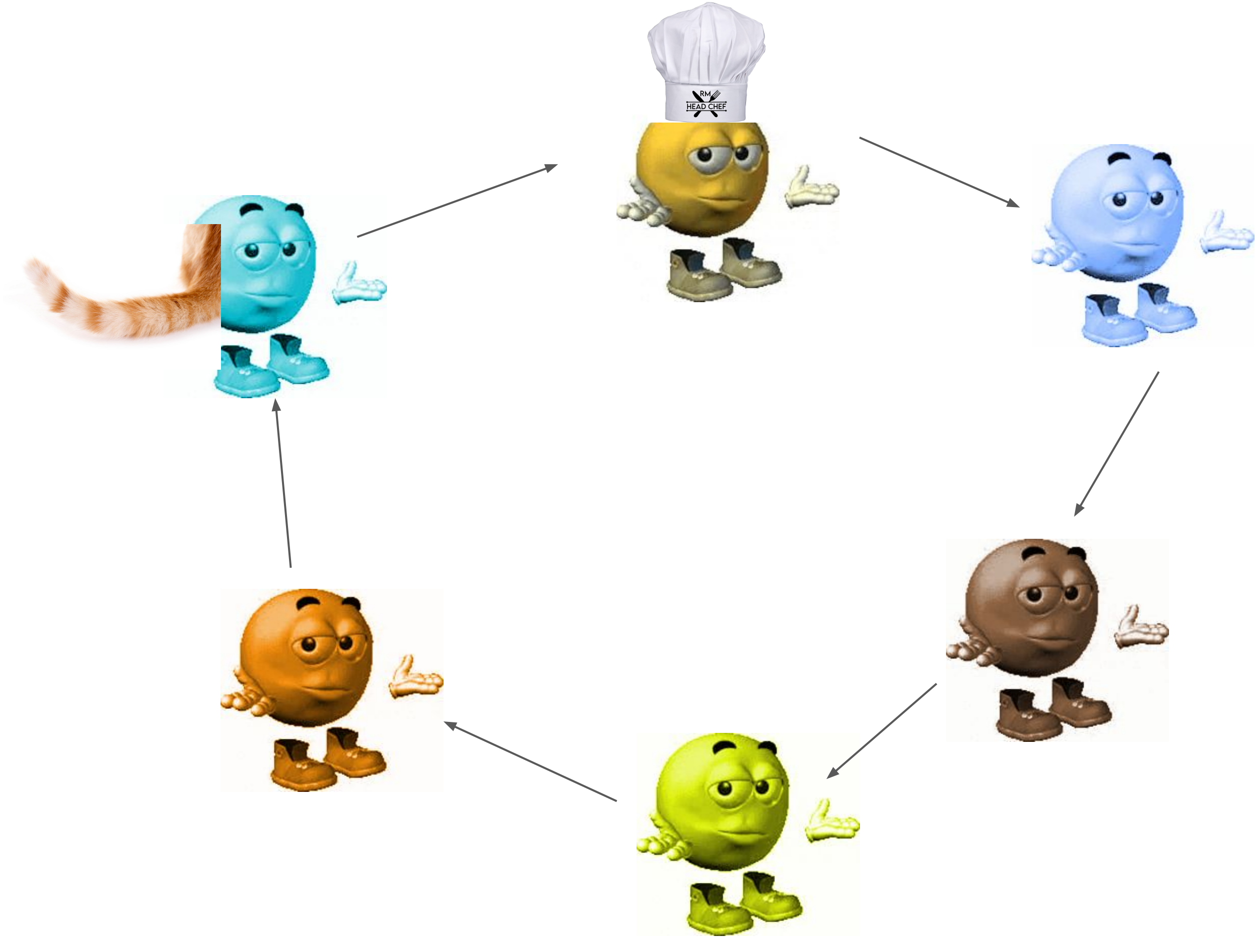
- Start with the first person (head of the list) and proceed to the kth person by traversing the list.
- Remove the kth person from the circle. This involves changing the next pointer of the $(k-1)$th person to point to the $(k+1)$th person.
- Continue the process, starting from the next person in the circle, until only one person remains.

Finally, identify the Last Survivor: The last remaining node in the list represents the position Josephus should choose. Return this position.

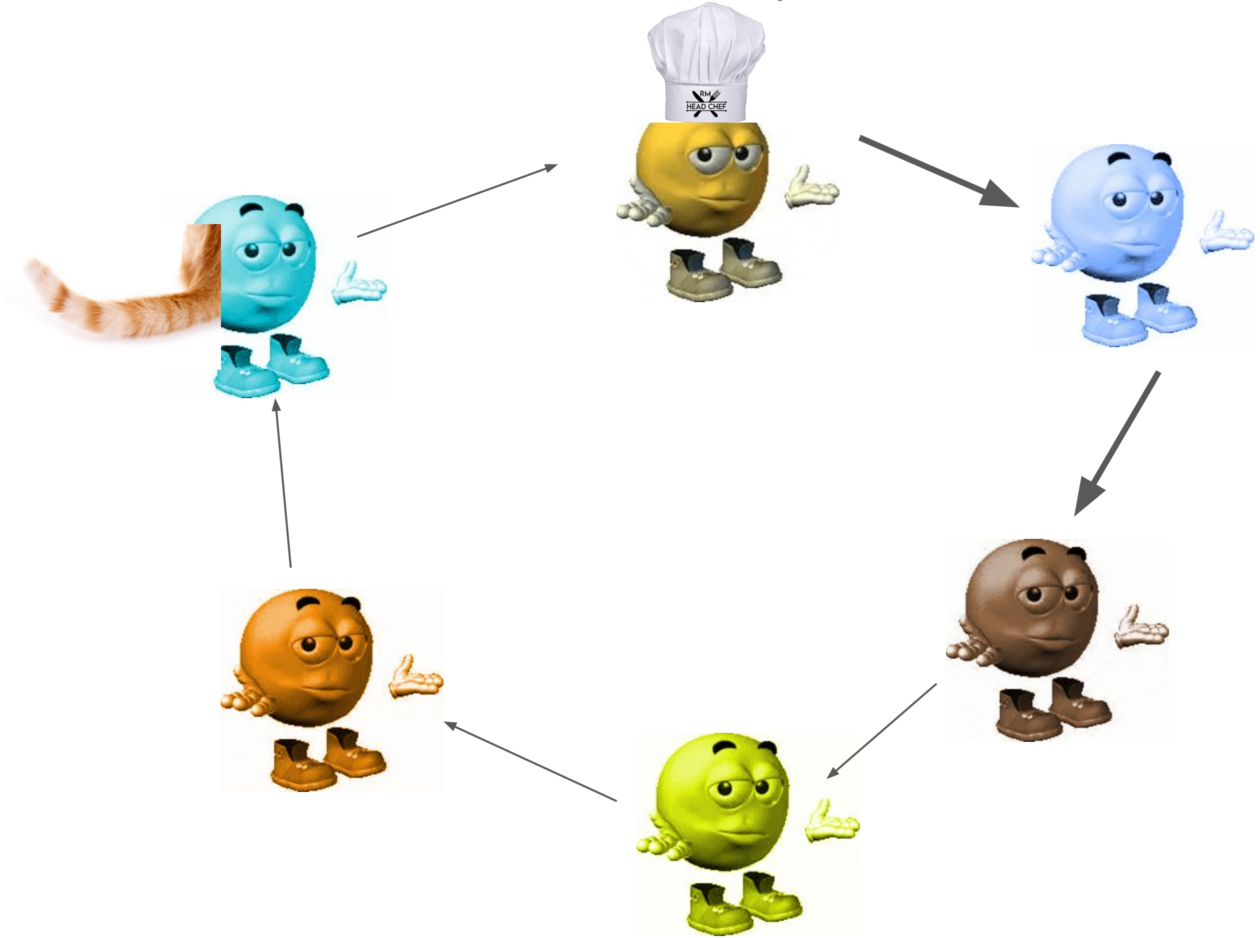
Provide the time complexity and space complexity of the solution.

Me telling you guys about Josephus lore

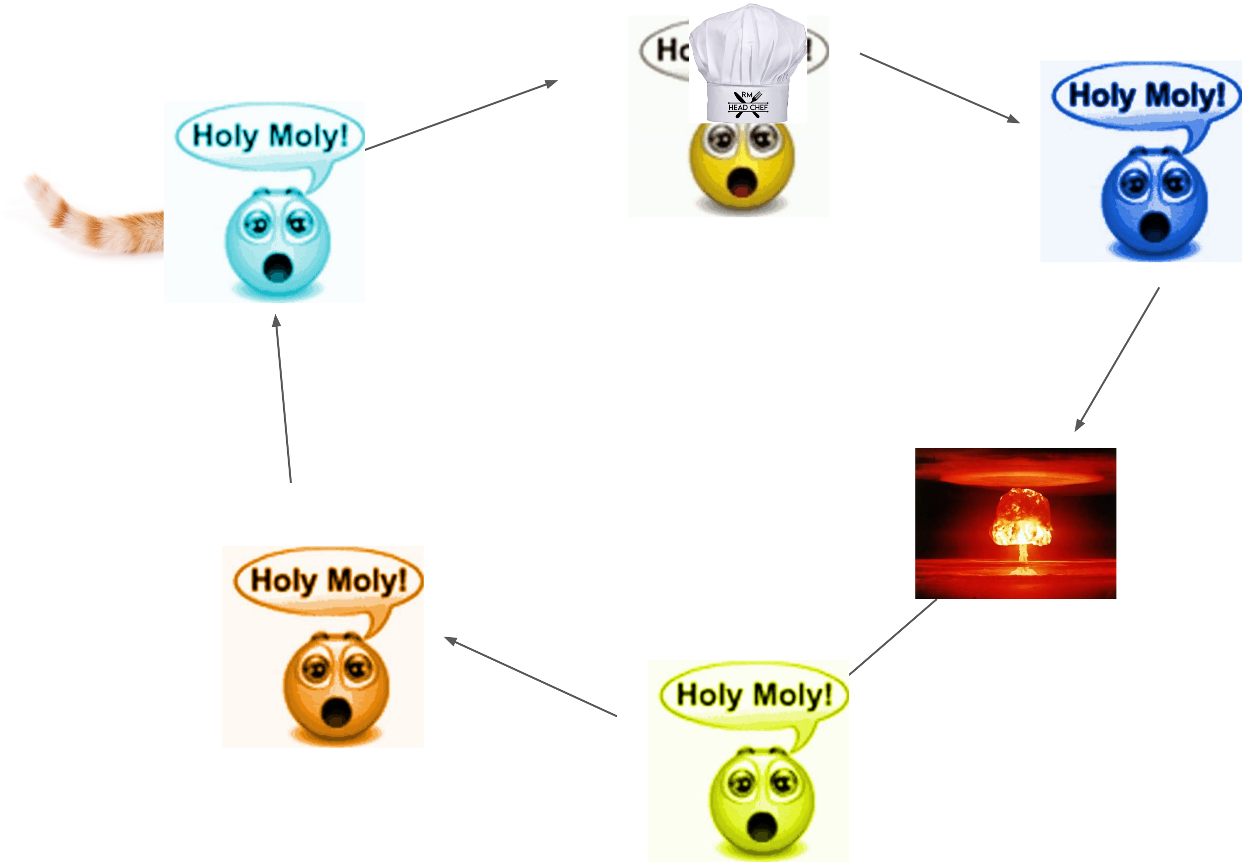
Step 1: Create a circular linked list of size n



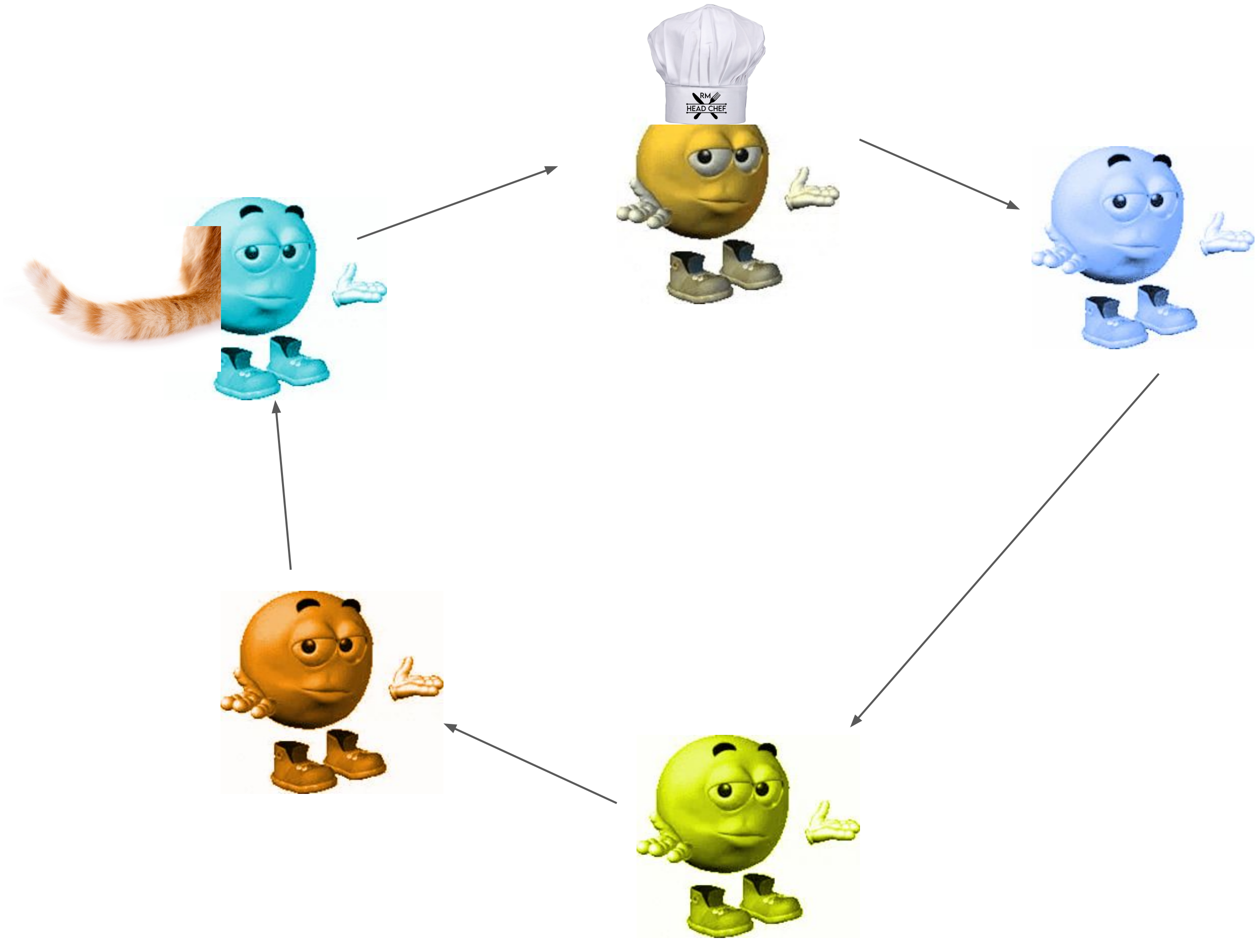
Step 2: From head node, traverse k next pointers



Step 3: Remove kth node



Step 3: Remove kth node



Step 3: Repeat until 1 node left

Step 3: Repeat until 1 node left

I am haunted by ghosts.
How can I repent?

Provide the time and space complexity of the algorithm

Step 3: Repeat until 1 node left

No

Ok, ghosts. I did what you asked. May I be free of y'all?