Fibonacci numbers:

\[ F_0 = 0 \]
\[ F_1 = 1 \]
\[ F_2 = 1 \]
\[ F_3 = 2 \]
\[ F_4 = 3 \]

Given \( n \), compute \( F_n \):

\[
\begin{align*}
\text{Init:} & \quad k = 1 \\
& \quad a = 1 \\
& \quad b = 0 \\
\text{While } k \leq n: & \quad \text{Y: } k < n \\
& \quad \text{X: } 1 \leq k \leq n \\
& \quad a = F_k \\
& \quad b = F_{k-1} \\
& \quad a = a + b \\
& \quad b = a - b \\
& \quad k = k + 1 \\
\text{Return } a
\end{align*}
\]

- Init: listone after init.
- Maintain acc: \( \mathbf{\text{acc}} \rightarrow \text{listone after one execution} \)

\[
\begin{align*}
\text{acc:} & \quad a, b, k \rightarrow a = F_k \left\{ \\
& \quad b = F_{k-1}.
\end{align*}
\]

- Getter one exec:

\[
\begin{align*}
a' &= a + b \\
b' &= a - b \\
k' &= k + 1
\end{align*}
\]

\[
a' = a + b = F_k + F_{k-1} = F_{k+1} = F_{1.1}
\]
Insertion Sort:

While
for i = 2 to n:
    key = A[i]
    j = i - 1
    while j ≥ 0 and A[j] ≥ key:
        j = j - 1
    A[j+1] = key

- worst case time: \( O(n^2) \) \( \xrightarrow{\text{Reversely}} \) Reversely sorted
  \( \Omega(h(n)) \) \( \Theta(n) \) = n \( \xrightarrow{\text{Sortet}} \) Sequence

\[
\begin{align*}
1 & \quad 2 & \quad 3 & \quad 4 & \quad 5 \\
\text{key} & \quad 1 & \quad -1 & \quad & \\
\overbrace{\begin{array}{c}
5 & \quad 4 & \quad 3 & \quad 2 & \quad 1 \\
1 & \quad 5 & \quad 3 & \quad 1 & \quad 2 \\
\text{key}
\end{array}}^{i}
\end{align*}
\]
Arrays

\[ A[0, 1, 2, 3, 4, 5, 6, 7] \]

+ \[ A[0], A[1] \] \( O(1) \) quick access

Linked list:

- Value
- Pointer

Good for dynamic list insertion and removal

\[ O(n) \]
1. How to reverse the list?
   - Maintain 3 pointers: previous, current, and next.
   - Assign appropriately when traversing the list.

2. Find the middle element.
   - If \( n \) is even, return \( \frac{n}{2} \).
   - If \( n \) is odd, return \( \frac{n+1}{2} \).

3. Two ways
   - First scan
   - Second scan

4. Having two pointers, slow and fast.
   - Fast is 2 times faster than the slow.
   - When traversed, once the fast reaches the end, the slow will be in the middle.
3. detect loop in linked list.

- First version:
  - With a hash table to check if an element is visited twice.
  - (Waste memory for the hash table)

- Second version:
  - With 2 pointers
  - slow = head, fast = head
  - While (slow != null and fast != null)
    - slow = slow.next
    - fast = fast.next.next
  - if (slow == fast):
    - return true. (there is a loop)
  - return false

Proof:
- Why this always happen if there is a loop?
- need to show: ∃i: i = k + 2i (mod n).
- Choose: i = n - k
- n-k = k + 2(n-k) (mod n)
  - = 2n - k (mod n) - n - k
  - 0 ≤ k ≤ n