香港中文大學（深圳）
The Chinese University of Hong Kong

CSC3100 Data Structures Midterm exam questions

Yixiang Fang
School of Data Science (SDS)
The Chinese University of Hong Kong, Shenzhen

Q 1

1. [15 marks] State and prove whether the following two statements are correct or not:

(1) $\frac{n^3}{\log n} = \Omega(n^3)$; (2) $f(n) + g(n) = \Theta(\max(f(n), g(n)))$, where $f(n) > 0$ and $g(n) > 0$

$g(n) = O(f(n))$ if and only if there exist positive constants c and n_0 such that $g(n) \le c \cdot f(n)$ for all $n \ge n_0$

$g(n) = \Omega(f(n))$ if and only if there exist a positive constant c and n_0 such that $g(n) \ge c \cdot f(n)$ for all $n \ge n_0$

(1) Not correct.

Assume that

$$\frac{n^3}{\log n} = \Omega\left(n^3\right).$$

Then there exist positive constants c and n_0 such that

$$0 \le cn^3 \le \frac{n^3}{\log n} \quad \text{for all } n > n_0.$$

Since $n \ge 2$, we have

$$n \le 2^{\frac{1}{c}}.$$

Here n is bounded, which contradicts the assumption. Thus, the statement is not correct.

Q 1

1. [15 marks] State and prove whether the following two statements are correct or not:

(1) $\frac{n^3}{\log n} = \Omega(n^3)$; (2) $f(n)+g(n)=\Theta(\max(f(n), g(n)))$, where $f(n) > 0$ and $g(n) > 0$

(2) Correct.

Since $f(n) \le \max(f(n), g(n))$, $g(n) \le \max(f(n), g(n))$, we have

$$f(n) + g(n) \le 2\max(f(n), g(n)).$$

Since $f(n) \ge 0$, $g(n) \ge 0$, we have

$$f(n) + g(n) = \max(f(n), g(n)) + \min(f(n), g(n)) \ge \max(f(n), g(n)).$$

Thus,

$$\max(f(n), g(n)) \le f(n) + g(n) \le 2\max(f(n), g(n)).$$

We can take $c_1 = 1$, $c_2 = 2$ and $n_0 = 0$ such that

$$0 \le c_1 \max(f(n), g(n)) \le f(n) + g(n) \le c_2 \max(f(n), g(n)) \quad \text{for all } n > n_0.$$

Thus,

$$f(n) + g(n) = \Theta(\max(f(n), g(n))).$$

Q 2

2. [20 marks] Suppose two singly linked lists, *list1* and *list2*, satisfies: (a) their head nodes can be accessed by *list1.head* and *list2.head* respectively (they do not store data); (b) each of them has *n* nodes; (c) for each node *x* in *list1* and *list2*, it has a next pointer and a data element, namely *x.next* and *x.data*; (d) all the data elements in each list are sorted in an ascending order.

(1) [8 marks] Design an algorithm with pseudocodes to put all the data elements of these two linked lists into one stack *S* (initially empty), such that all the data elements in the stack are organized in ascending order, and its bottom element is the **smallest** one.

(2) [7 marks] Design an algorithm with pseudocodes to put all the data elements of these two linked lists into one queue *Q* (initially empty), such that all the data elements in the queue are organized in desending order, and its first data element is the **largest** one.

(3) [5 marks] Can we count the frequency of each data element of these two linked lists in *O(n)* time (e.g., if a data element appears only once in *list1* and *list2* respectively, then its frequency is 2)? If yes, explain your algorithm steps; if no, explain why.

Q 2

(1) The algorithm is as follows.

```
PUT-INTO-STACK(list1, list2)
let S be an empty stack.
p1 = list1.head.next
p2 = list2.head.next
while p1 ≠ NIL and p2 ≠ NIL
    if p1.data < p2.data
        PUSH(S, p1.data)
        p1 = p1.next
    else
        PUSH(S, p2.data)
        p2 = p2.next
while p1 ≠ NIL
    PUSH(S, p1.data)
    p1 = p1.next
while p2 ≠ NIL
    PUSH(S, p2.data)
    p2 = p2.next
return S
```

(2) The algorithm is as follows.

```
PUT-INTO-QUEUE(list1, list2)
S = PUT-INTO-STACK(list1, list2
let Q be an empty queue.
while not EMPTY(S)
    data = POP(S)
    ENQUEUE(Q, data)
return Q
```

(3) The algorithm is as follows.

```
COUNT-FREQUENCY(list1, list2)
S = PUT-INTO-STACK(list1, list2)
T = ∅
last = NIL
freq = −1
while not EMPTY(S)
    data = POP(S)
    if freq == −1
        last = data
        freq = 1
    elseif last ≠ data
        T = T ∪ {last ↦ freq}
        last = data
        freq = 1
    else
        freq = freq + 1
return T
```

Q 3

3. [20 marks] Consider a singly linked list, in which each node *x* has a next pointer and a data element, namely *x.next* and *x.data*, where *x.data* is an integer value in [0, 10000].
(1) [10 marks] Design an algorithm with pseudocodes to count the total number of nodes in the linked list. Notice that the head node should not be counted.
(2) [10 marks] Assume that the above linked list has *n* nodes. Now given *n* arbitrary integer values, design an algorithm to find out the integer values of them that are in the linked list above in *O(n)* time cost.

(1) The algorithm is as follows.

COUNT(*list*)

$n = 0$
$p = list.head.next$
while $p \neq$ NIL
    $n = n + 1$
    $p = p.next$
return n

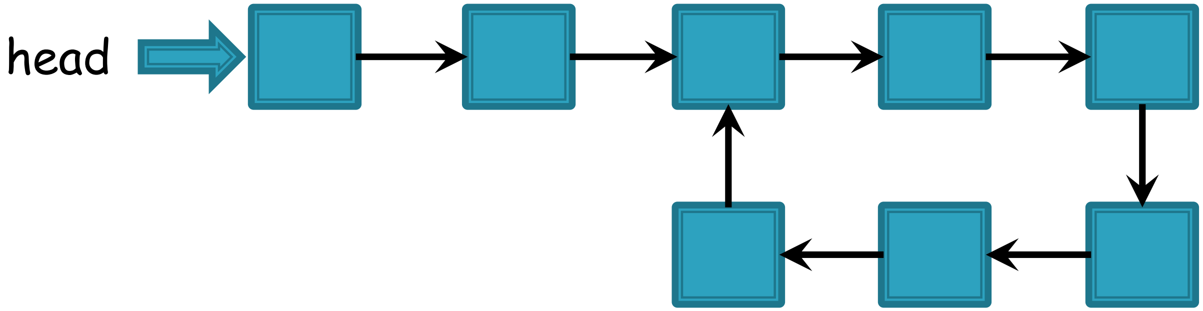
You need to check whether the list has a cycle or not !!!

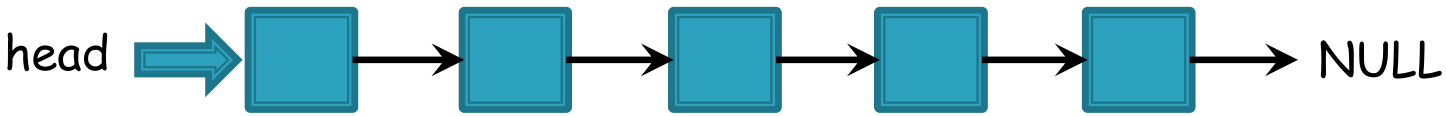
FIND-VALUES-IN-LIST(*list*, *values*)

let $C[0\,..\,10000]$ be a new array
for $i = 0$ **to** 10000
    $C[i] =$ FALSE
$p = list.head.next$
while $p \neq$ NIL
    $C[p.data] =$ TRUE
    $p = p.next$
$S = \varnothing$
for $data \in values$
    if $0 \leq data \leq 10000$ and $C[data]$
        $S = S \cup \{data\}$
return S

Exercise 3: cycle detection

Given the head of a singly linked list L, decide if L has a cycle





Method #1

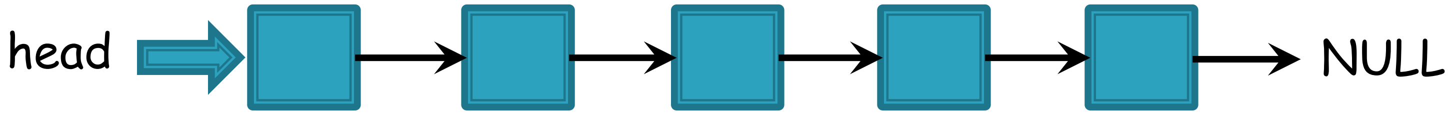
- If L is acyclic, we ultimately arrive at NULL by continuously following the next pointer:

```
p = L.head
for i = 1 upto M
        if p == NULL
                return "acyclic"
        else   p = p.next
return "cyclic"
```

M is some big number

- M must be sufficiently large to guarantee correctness, but it is hard to decide M

Method #2

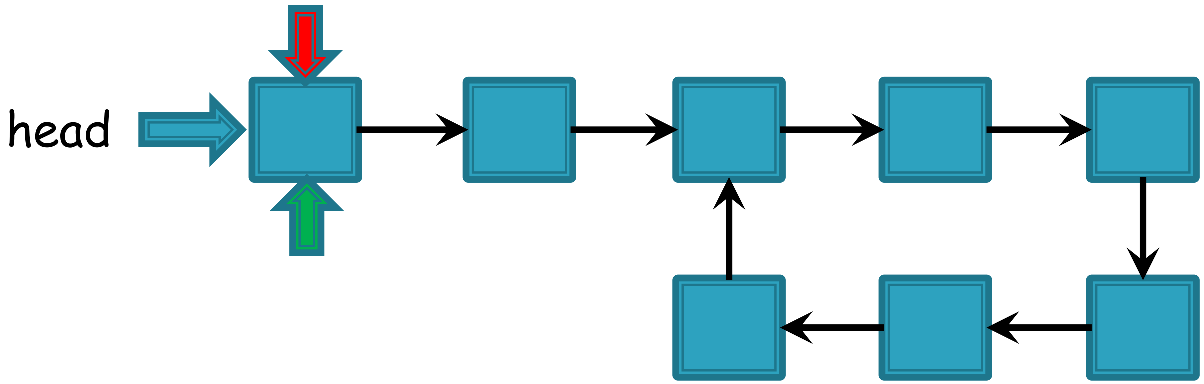


- Store all revisited noded in a new list L':

```
p = L.head
while p != NULL
        if search(L', p) == NULL
                insert(L',p)
                p = p.next
        else return "cyclic"
return "acyclic"
```

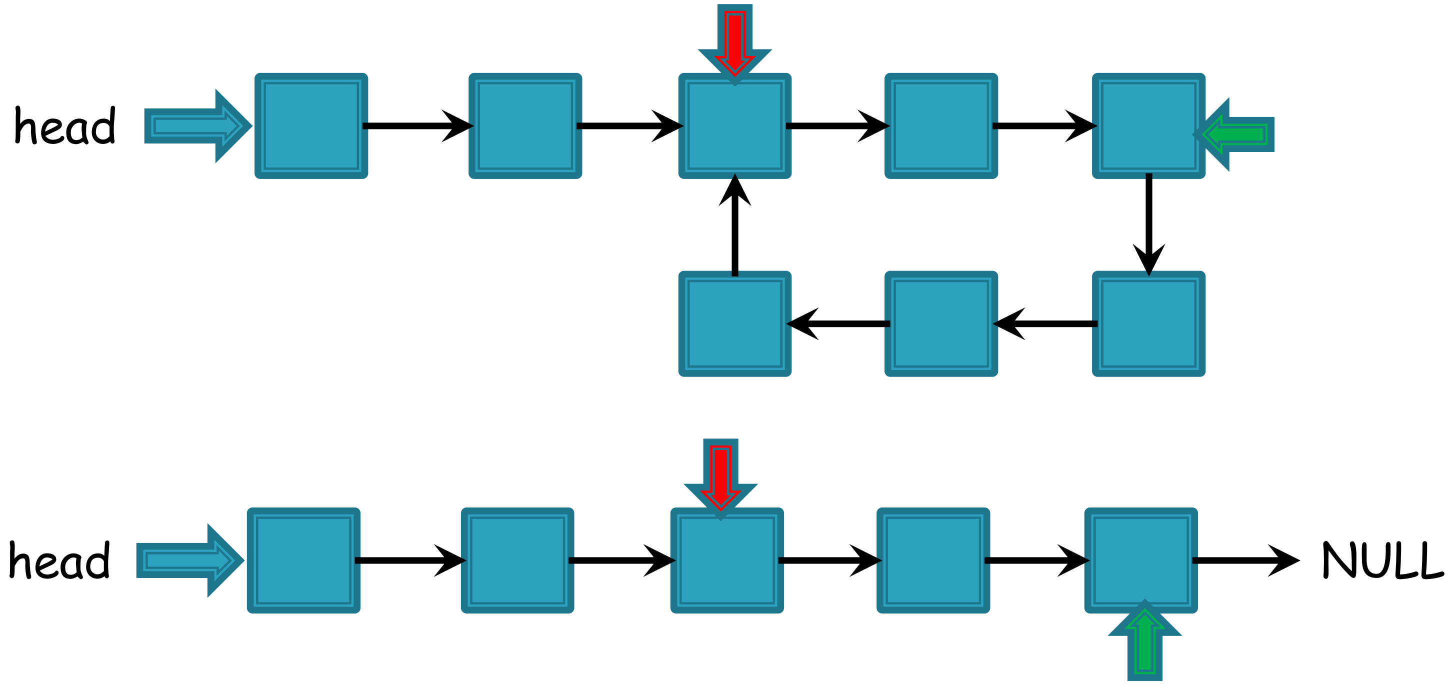
- Search on L' is expensive; use a Hash table

Method #3



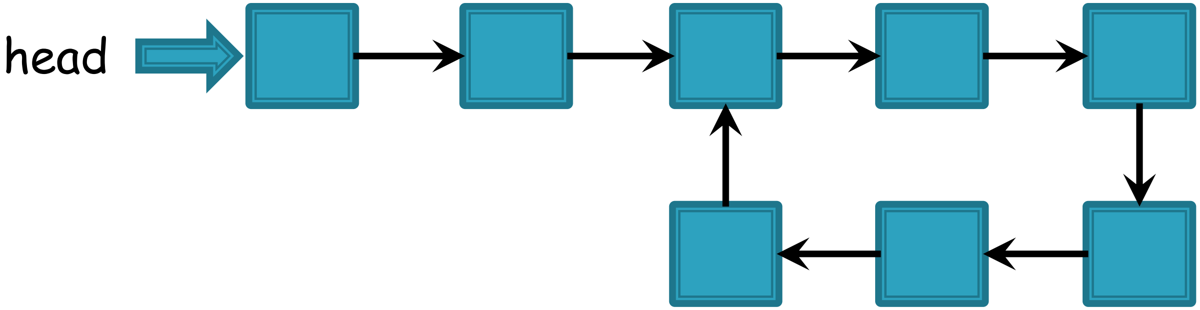
- Use two pointers A and B, both initialized to head
- Every time A=A.next while B=B.next.next

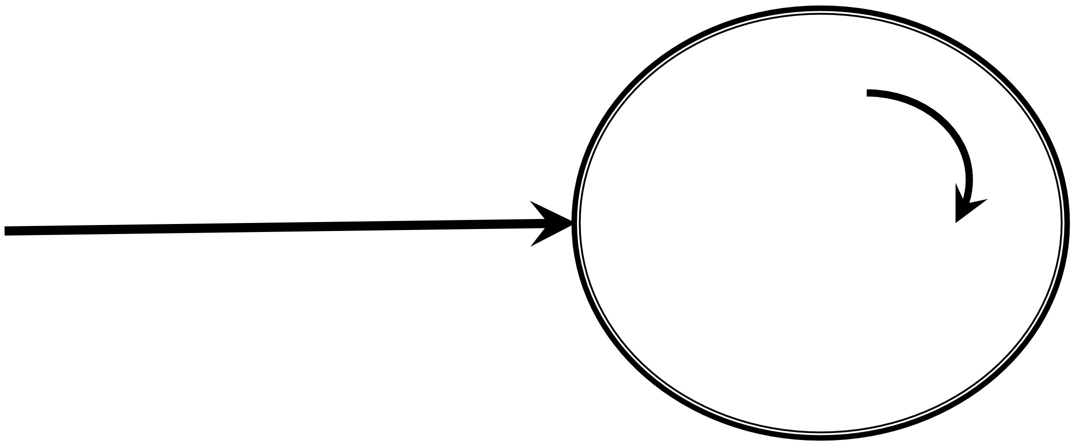
Method #3



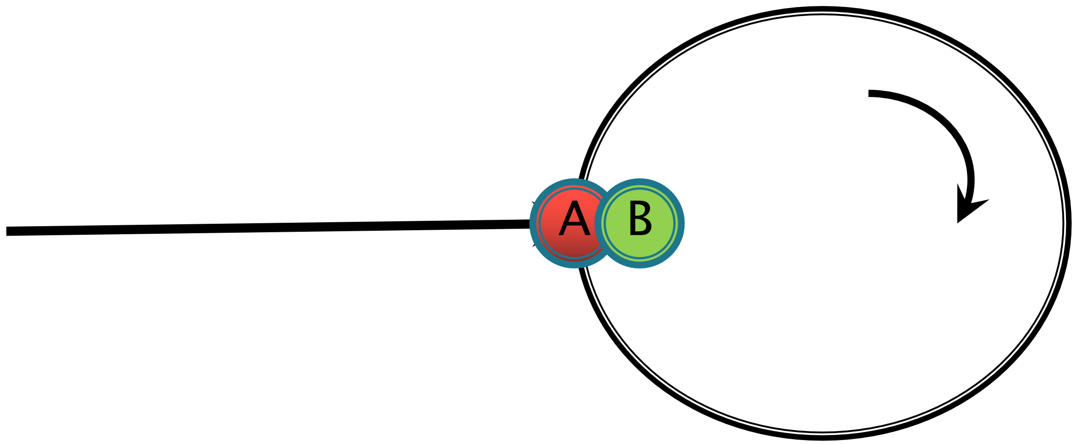
- If L is acyclic, either B or B.next be NULL
- If L is cyclic, B enters the cycle earlier than A

Method #3



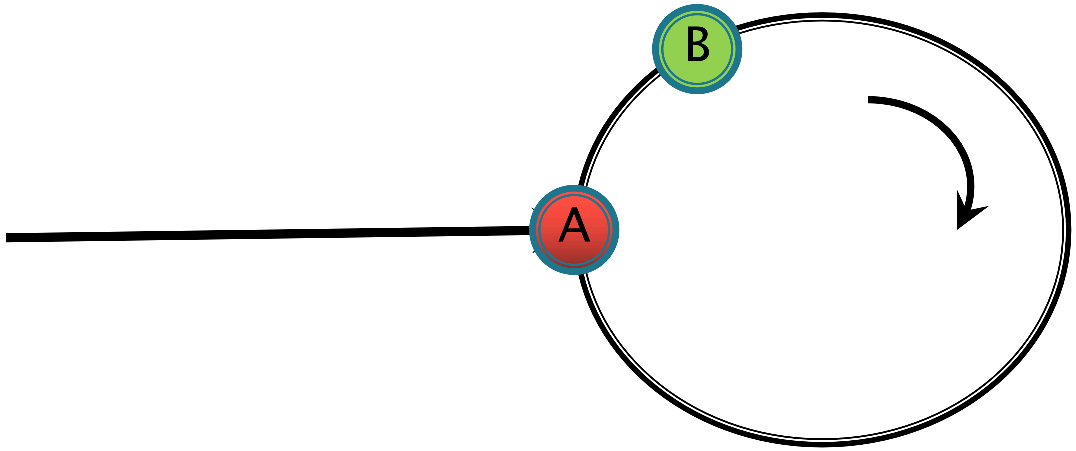


Method #3



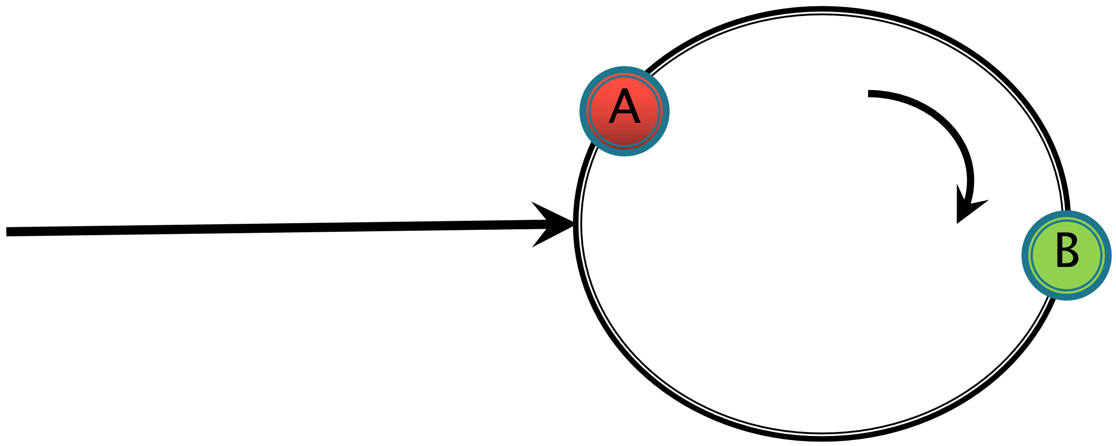
- Case I: B is exactly at entrance when A arrives at the cycle
- So A and B meet at entrance

Method #3



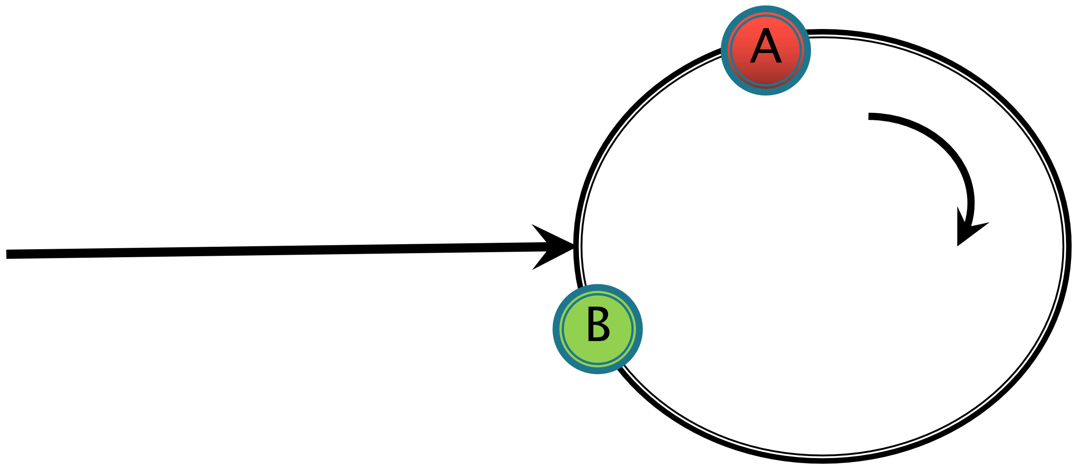
- Case II: B is somewhere else in the cycle when A arrives at entrance

Method #3



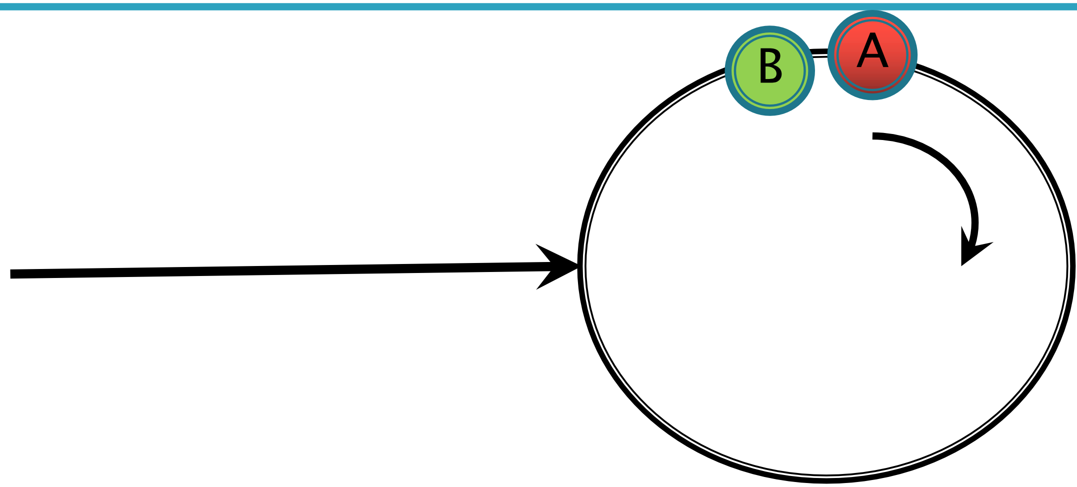
- Since B is moving faster, it must overtake A at a certain point in time
- After t timestamps, the distance gap is (2-1)t = t
- The distance of a circle is x, which is a constant
- So the distances between A and B are 0,1,2,..., t-1

Method #3



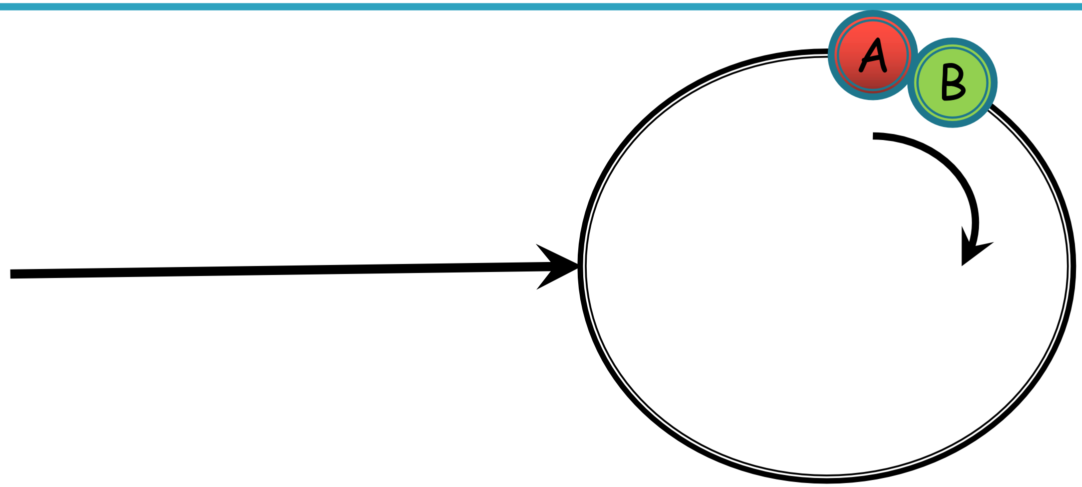
- Since B is moving faster, it must overtake A at a certain point in time

Method #3



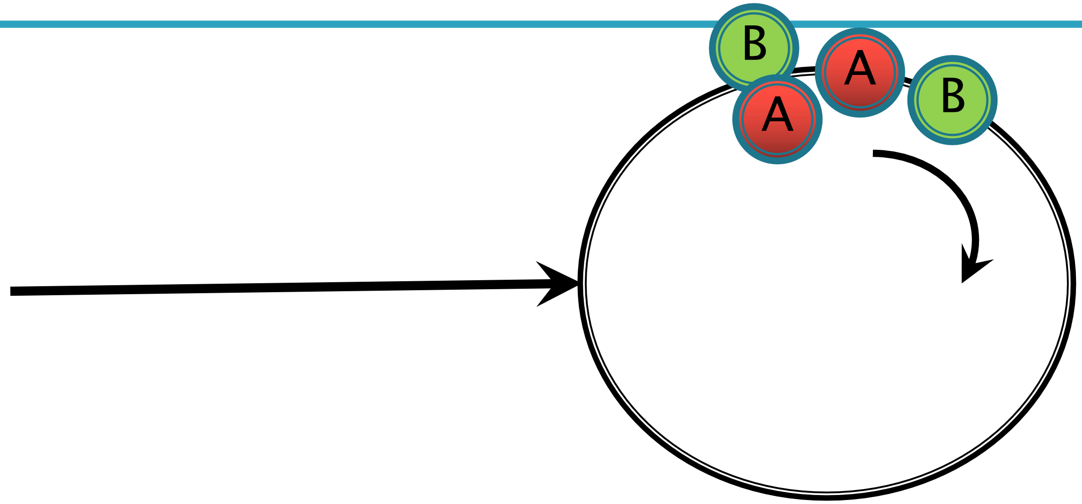
- Since B is moving faster, it must overtake A at a certain point in time

Method #3



- Since B is moving faster, it must overtake A at a certain point in time

Method #3



- And right before B overtakes A, the two nodes meet

Method #3

- Thus, A and B are guaranteed to meet if list is cyclic

```
A = L.head; B = L.head
while B != NULL and B.next != NULL
        if A == B
                return "cyclic"
        A = A.next
        B = B.next.next
return "acyclic"
```

Q 4

4. [15 marks] Suppose that we have an array A with n ($n \geq 3$) numbers, and all the numbers first increase and then decrease, i.e., $A[1] < A[2] < \cdots < A[i]$ and $A[i] > A[i+1] > \cdots A[n-1] > A[n]$ for some $i \in [2, n-1]$, but the value of i is unknown in advance.

(1) [10 marks] Given an arbitrary number k, can we check whether it is contained by A in *O(log n)* time cost? If yes, explain your algorithm steps; if no, explain why.

(3) [5 marks] Design a sorting algorithm with pseudocodes to sort the above array, such that all the numbers are in ascending order and the overall time cost is bounded by *O(n)*.

```
FIND-I(A)
l = 1
r = n
while l ≤ r
    m = (l + r)/2
    if A[m − 1] < A[m] < A[m + 1]
        l = m
    elseif A[m − 1] > A[m] > A[m + 1]
        r = m
    else
        return m
```

```
CONTAINS(A, k)
i = FIND-I(A)
l = 1
r = i
while l ≤ r
    m = (l + r)/2
    if A[m] < k
        l = m + 1
    elseif A[m] > k
        r = m − 1
    else
        return TRUE
l = i + 1
r = n
while l ≤ r
    m = (l + r)/2
    if A[m] > k
        l = m + 1
    elseif A[m] < k
        r = m − 1
    else
        return TRUE
return FALSE
```

Recall MergeSort

1 13 24 26 2 15 27 38

↑ ↑ ↑

Aptr *Bptr* *Cptr*

1 13 24 26 2 15 27 38 1

↑ ↑ ↑

Aptr *Bptr* *Cptr*

1 13 24 26 2 15 27 38 1 2

↑ ↑ ↑

Aptr *Bptr* *Cptr*

Recall MergeSort

1 13 24 26 2 15 27 38 1 2 13

↑ ↑ ↑

Aptr *Bptr* *Cptr*

1 13 24 26 2 15 27 38 1 2 13 15

↑ ↑ ↑

Aptr *Bptr* *Cptr*

1 13 24 26 2 15 27 38 1 2 13 15 24

↑ ↑ ↑

Aptr *Bptr* *Cptr*

Q 4

```
SORT(A)
i = FIND-I(A)
let B[1..n] be a new array
l = 1
r = n
k = 1
while l ≤ i and r ≥ i + 1
    if A[l] < A[r]
        B[k] = A[l]
        l = l + 1
        k = k + 1
    else
        B[k] = A[r]
        r = r - 1
        k = k + 1
while l ≤ i
    B[k] = A[l]
    l = l + 1
    k = k + 1
while r ≥ i + 1
    B[k] = A[r]
    r = r - 1
    k = k + 1
```

Q 5

5. [15 marks] Consider a binary search tree shown in Figure 1.
(1) [9 marks] Write the preorder, inorder, and postorder sequences of traversing the three by using preorder, inorder, and postorder traversal strategies, respectively.
(2) [6 marks] Perform two sequentially operations: first delete key 15, and then insert a new key 8 into the updated tree. Please draw the two updated binary search trees after these operations respectively.

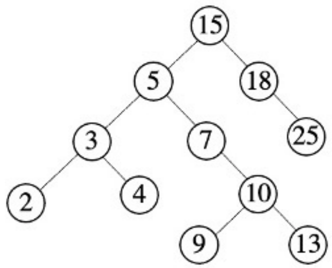


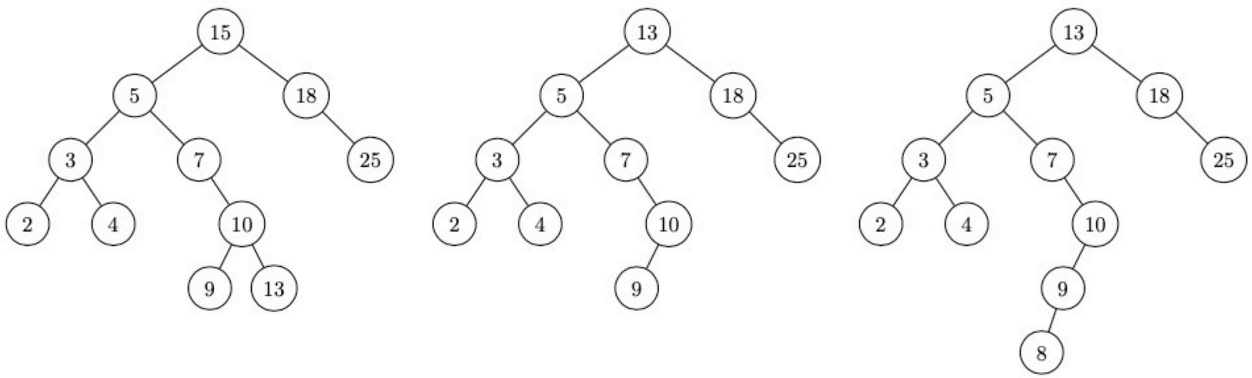
Figure 1

- Preorder: $[15, 5, 3, 2, 4, 7, 10, 9, 13, 18, 25]$.
- Inorder: $[2, 3, 4, 5, 7, 9, 10, 13, 15, 18, 25]$.
- Postorder: $[2, 4, 3, 9, 13, 10, 7, 5, 25, 18, 15]$.

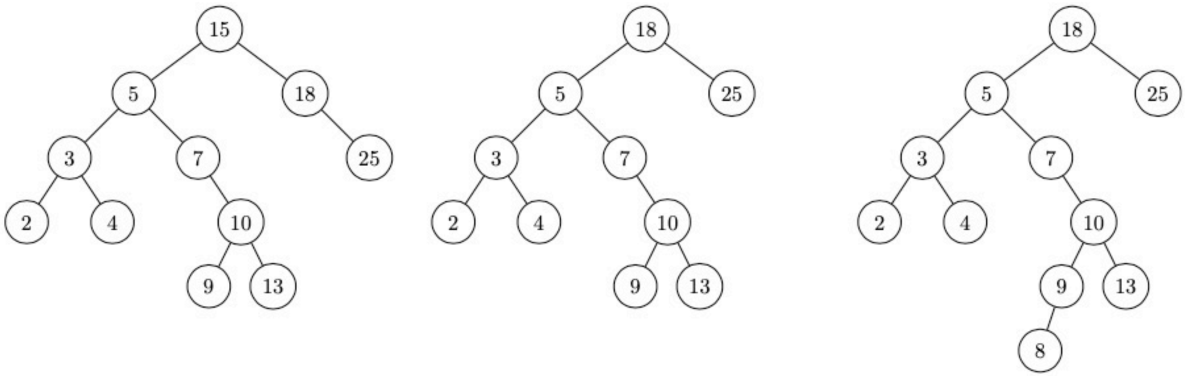
Q 5

5. [15 marks] Consider a binary search tree shown in Figure 1.
(1) [9 marks] Write the preorder, inorder, and postorder sequences of traversing the three by using preorder, inorder, and postorder traversal strategies, respectively.
(2) [6 marks] Perform two sequentially operations: first delete key 15, and then insert a new key 8 into the updated tree. Please draw the two updated binary search trees after these operations respectively.

Solution 1:



Solution 2:



Q 6

6. [15 marks] Consider the two binary search trees in Figures 1 and 2.
(1) [6 marks] Are these two trees AVL trees? Show your answers and reasons.
(2) [9 marks] Consider the tree in Figure 2. <u>Sequentially</u> insert two new keys 1 and 4 into it, and after each insertion, please update the tree as an AVL tree. Please draw the two updated AVL trees.

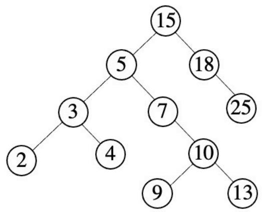


Figure 1

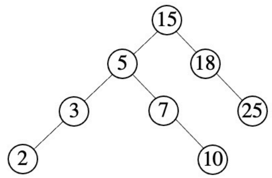


Figure 2

(1)

- Figure 1 is not an AVL since the height of the left subtree of node 15 is 4 and the height of right subtree of node 15 is 2.
- Figure 2 is an AVL since for every node in the tree, the height of the left subtree differs from the height of the right subtree by at most 1.

(2)