Welcome to class!

Turn on your camera

Rename yourself to include your pod # at the beginning of your Zoom name - *ie 6 - Emily Garvey (she/her)*

How are you feeling today? Emojis in chat!

Community Agreements |

⏰ We will be here and be present ⏰

❣️ Actively build community & create an inclusive space ❣️

🧠 We will be curious and have a learning mindset 🧠

❓ Ask for help when you need it ❓

📹 We will all have our cameras on 📹

Agenda

Check In	5 mins
Linked Lists	15 mins
Problem Walkthrough	20 mins
Breakout Sessions	60 mins
Q&A and Wrap Up	20 mins

CODEPATH*ORG

Goals

- ❑ We'll be covering Linked Lists today!
- ❑ In our breakout sessions, we will work with our mentors to solve problems using UMPIRE
- ❑ We will meet back as a group to wrap up

CODE PATH *ORG

Survey Feedback | What we heard

Time | [3:00 mn]

Glows

“Meeting podmates”
“UMPIRE”
“Learning about career center”

Grows

“Not enough practice with UMPIRE”
“Post slides before class”
“Too much administrative info”

Linked Lists | 15 mins

Activity | Check for Understanding

Time | [1:00 mn]

What is the **time complexity** to count the number of elements in the linked list?

a) O(1)
b) O(n)
c) O(logn)
d) None of the above

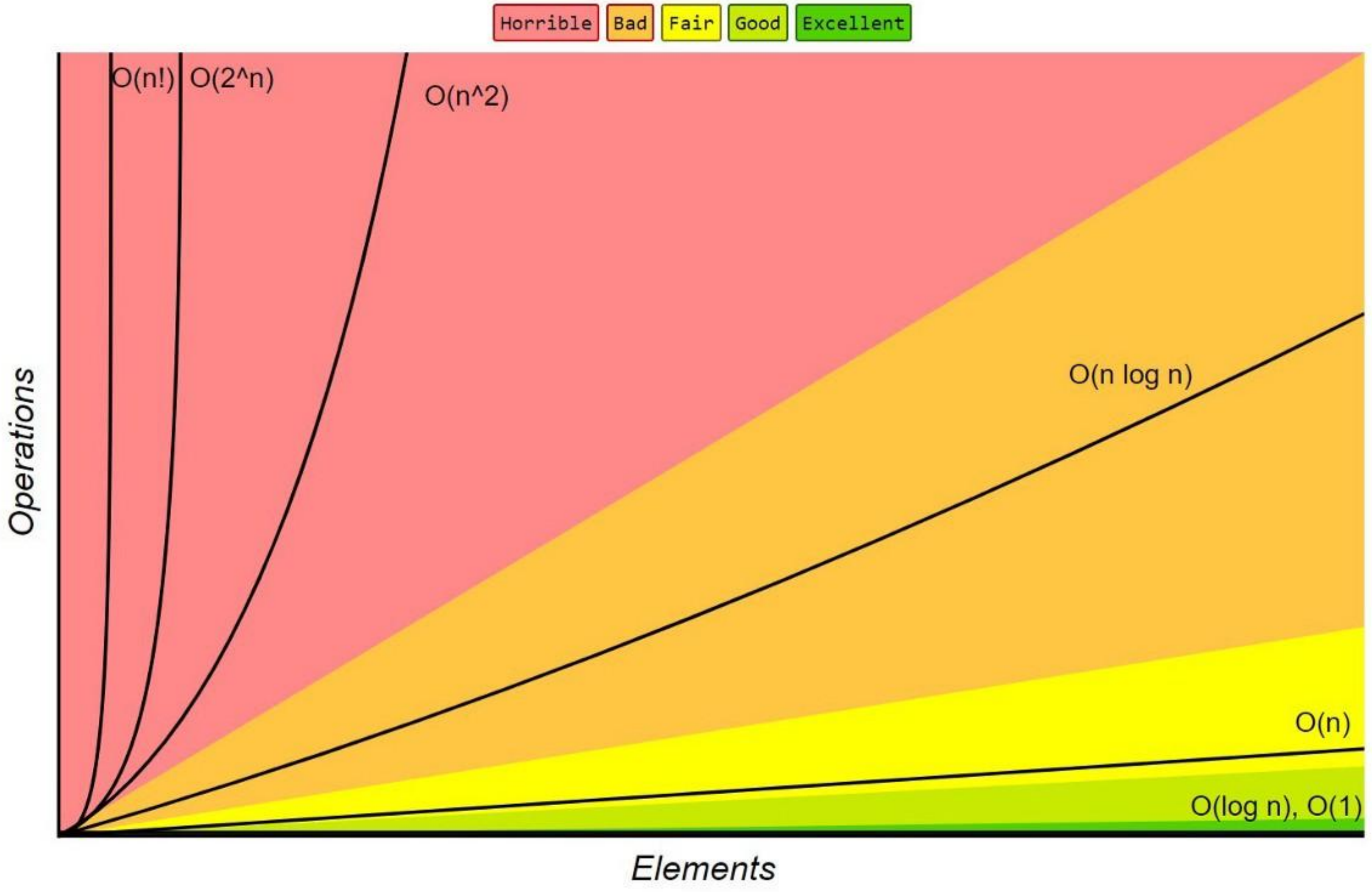
Activity | Check for Understanding

Time | [1:00 mn]

Answer: B

To count the number of elements, you have to traverse through the entire list, hence complexity is linear, or O(n)

Big-O Complexity Chart

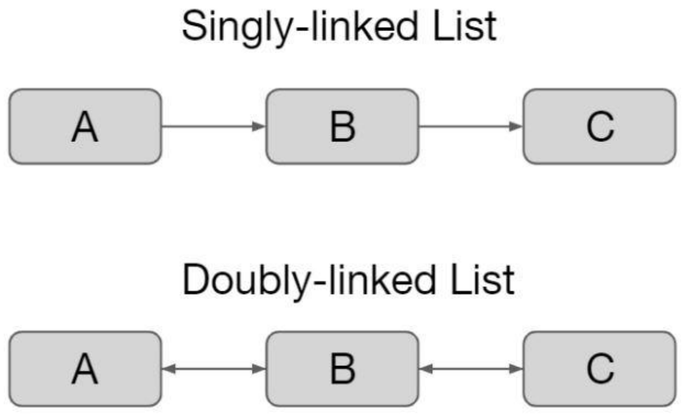


https://guides.codepath.com/compsci/Big-O-Complexity-Analysis

CODE PATH *ORG

Linked Lists

- Comprised of nodes that store an arbitrary value (usually a string or a number)
- Different types of linked lists
    - Singly-linked list: each node only has a next pointer
    - Doubly-linked list: each node has a next and previous pointer

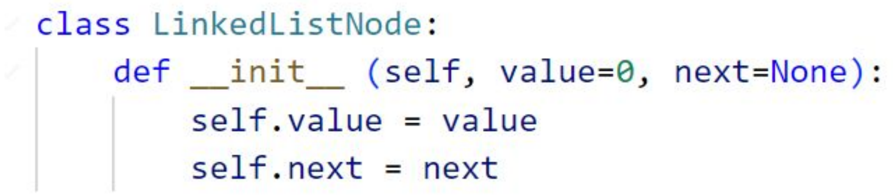


CODE PATH ORG

Linked Lists Class

- You only need a value in order to initialize a node
  - next
  - prev (if doubly linked list)

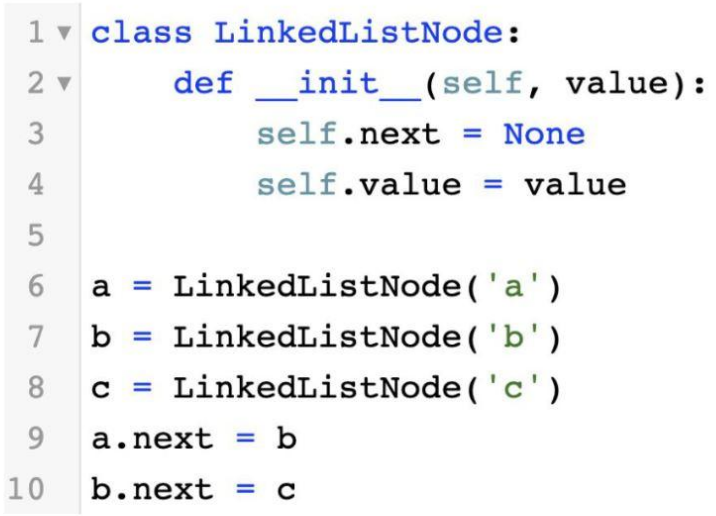
```java
public class LinkedListNode {
    Object value;
    LinkedList next;

    public LinkedListNode(Object value) {
        this.next = null;
        this.value = value;
    }
}
```



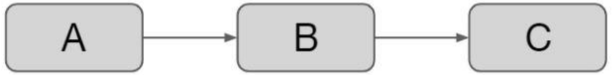
```python
class LinkedListNode:
    def __init__ (self, value=0, next=None):
        self.value = value
        self.next = next
```

Creating Linked Lists (python)

- Initialize each node and assign the next pointers properly

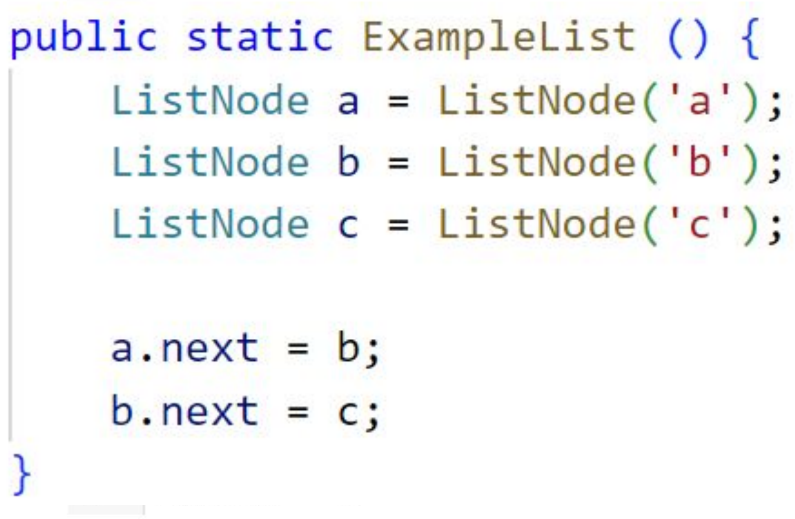


```python
class LinkedListNode:
    def __init__(self, value):
        self.next = None
        self.value = value

a = LinkedListNode('a')
b = LinkedListNode('b')
c = LinkedListNode('c')
a.next = b
b.next = c
```

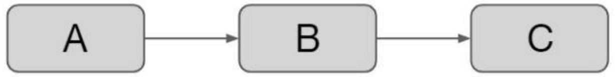


Creating Linked Lists (Java)

- Initialize each node and assign the next pointers properly



```
public static ExampleList () {
    ListNode a = ListNode('a');
    ListNode b = ListNode('b');
    ListNode c = ListNode('c');

    a.next = b;
    b.next = c;
}
```



CODE PATH *ORG

Activity | Check for Understanding

Time | [1:00 mn]

What would be the asymptotic time complexity to add a node at the end of singly linked list, if the pointer is initially pointing to the head of the list?

a. O(1)
b. O(n)
c. O(n^2)
d. O(1)

Activity | Check for Understanding

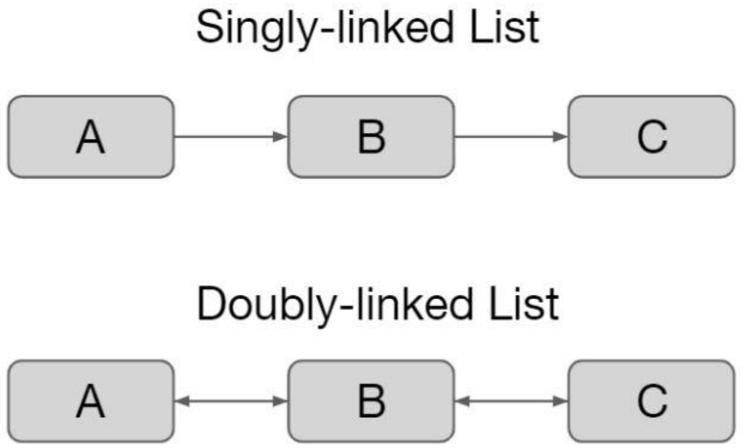
Time | [1:00 mn]

b) O(n)

Time complexity of append is O(n) where n is the number of nodes in the linked list. Since there is a loop from head to end, the function does O(n) work.

The **Match** Step Of U**M**PIRE: Linked Lists

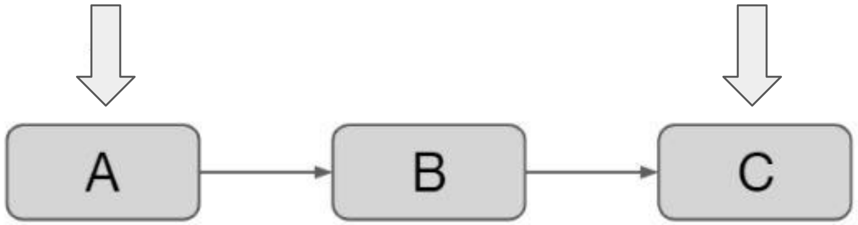
- All problems deal with pointer-manipulation and traversing the list
- Most problems require you to use O(1) space
- Common patterns in solving these problems:
  - Dummy head
  - Two pointer
  - multi-pass



CODE PATH *ORG

Match: Two Pointer

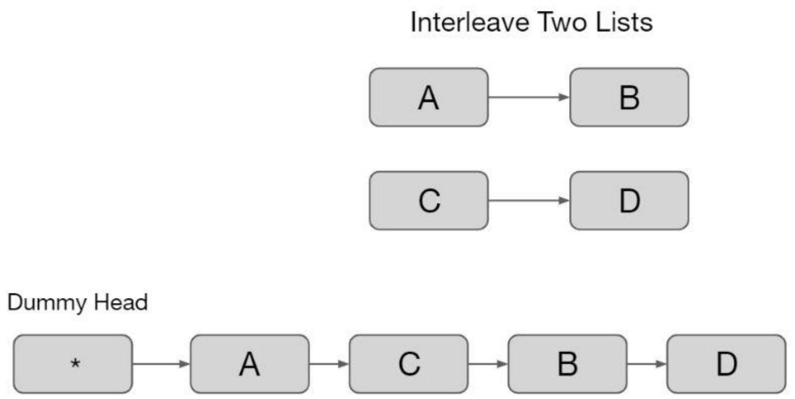
- Use two/multiple pointers to manipulate references to nodes (eg. reversing a linked list)
- Also used to get/infer certain information about list (eg. detect cycle, get N nodes apart)



CODE PATH *ORG

Match: Dummy Head

- Create a dummy head node and construct a list using its next pointer
- Return dummyHead.next to return the newly constructed list
- Very useful when you need to manipulate pointers and create a list with the same nodes but different ordering



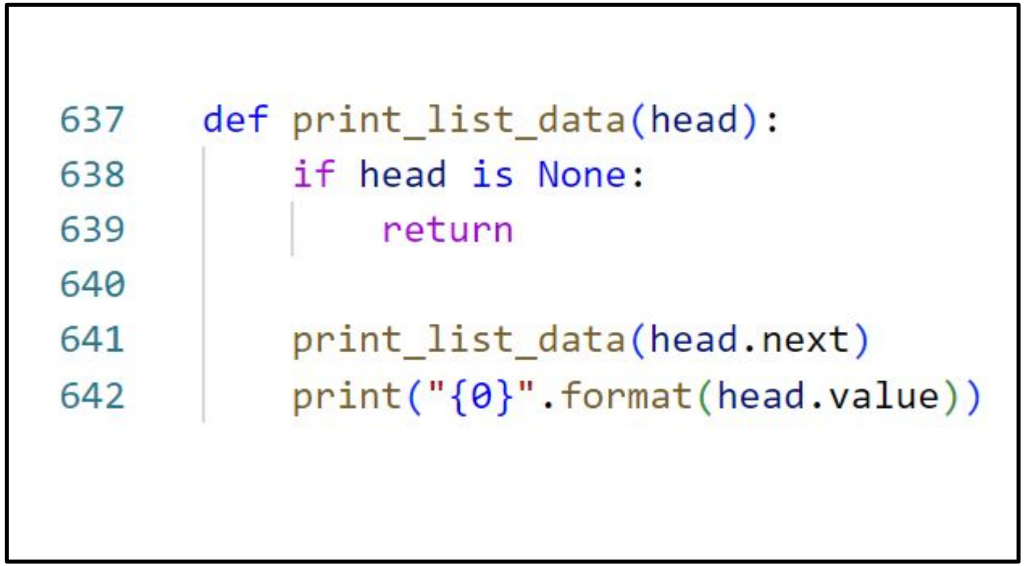
CODE PATH *ORG

Other Match options

- Multi-pass
  - If you can guarantee constant amount of passes or if you need to know the length of the list
- LL Reverse
  - Adding two lists together
  - LL palindrome

Activity | Check for Understanding

Time | [1:00 mn]



```
def print_list_data(head):
    if head is None:
        return

    print_list_data(head.next)
    print("{0}".format(head.value))
```

Post on the chat: In detail, what does the function do?

Activity | Check for Understanding

Time | [1:00 mn]

print_list_data is iterating through the linked list recursively and thus printing all the values in the linked list backwards. Why backwards?

UMPIRE Review & | 20 mins
Problem Walkthrough

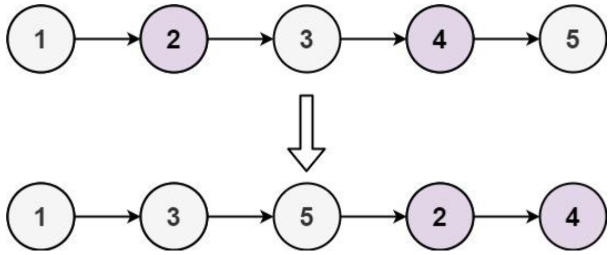
Odd Even Linked List

Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The first node is considered odd, and the second node is even, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

You must solve the problem in O(1) extra space complexity and O(n) time complexity.

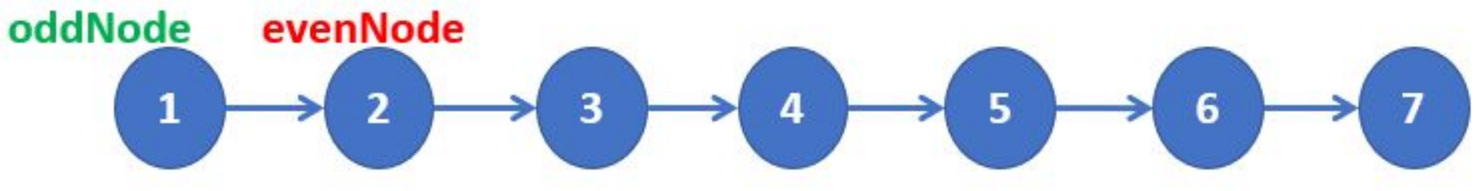


Input: head = [1,2,3,4,5]
Output: [1,3,5,2,4]

https://leetcode.com/problems/odd-even-linked-list/

U-nderstand

- How do we control what is our current node in the odd list and even list?
- Could the input node be null?
- Is the list guaranteed to have an even or odd length?



CODE PATH *ORG

M-atch

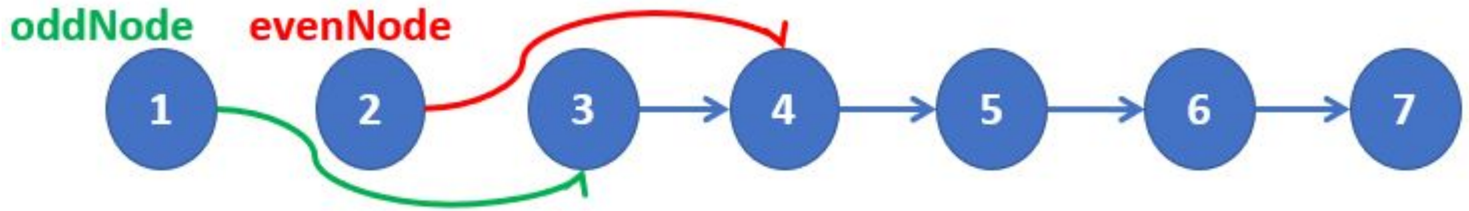
- Can we solve this problem using…
  - Multiple passes over the linked list
  - Dummy Head
  - Two Pointers

P-lan

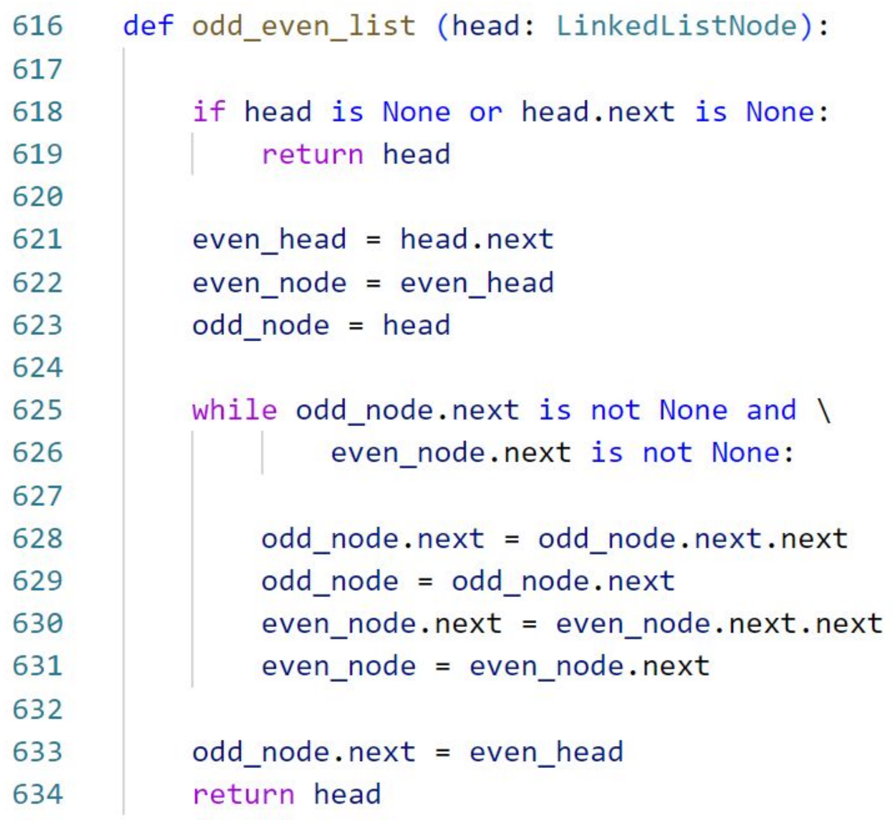
Let **oddNode** point to the node after the current next node.

Let **evenNode** point to the node after the current next node.

After assigning the nodes to the “correct” next node, we will let them move forward and continue the process until one of them touches null. (i.e., their next node points to null)

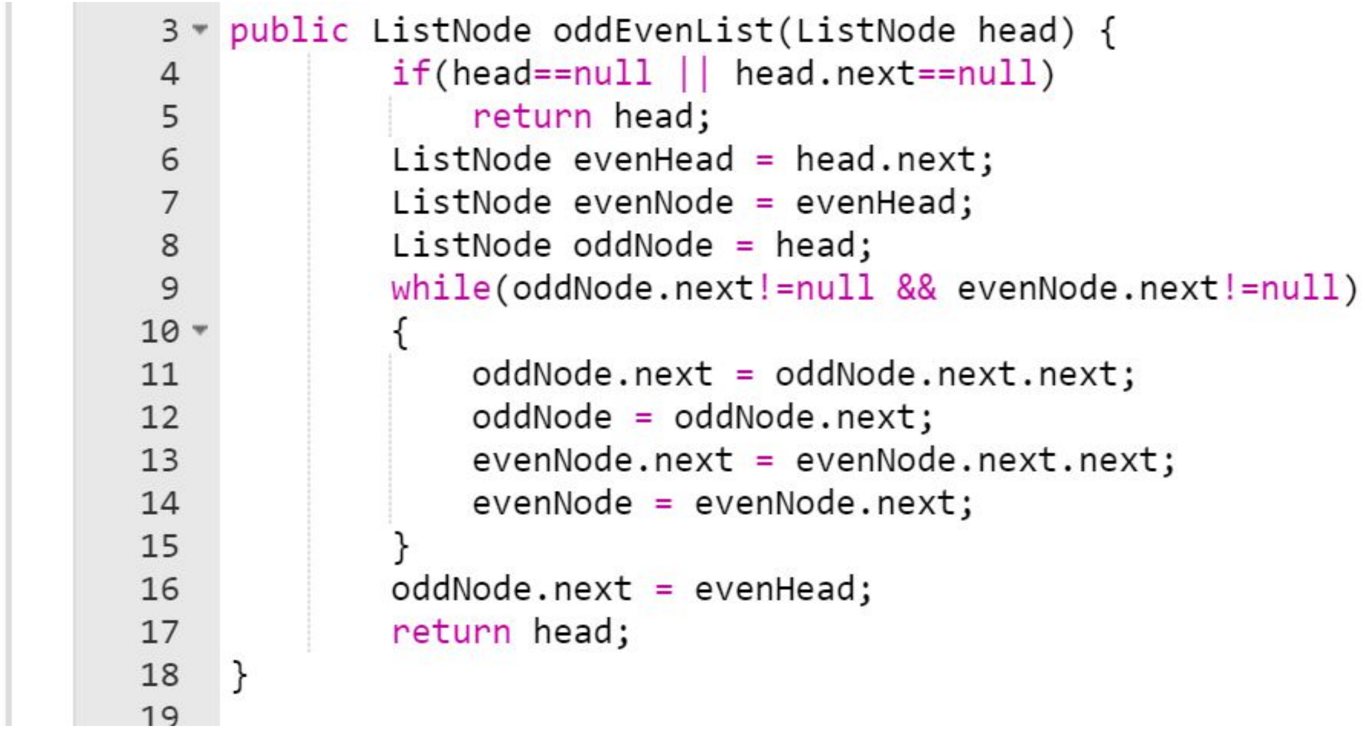


I-mplement (python)



```python
def odd_even_list (head: LinkedListNode):

    if head is None or head.next is None:
        return head

    even_head = head.next
    even_node = even_head
    odd_node = head

    while odd_node.next is not None and \
            even_node.next is not None:

        odd_node.next = odd_node.next.next
        odd_node = odd_node.next
        even_node.next = even_node.next.next
        even_node = even_node.next

    odd_node.next = even_head
    return head
```

I-mplement (Java)



```java
public ListNode oddEvenList(ListNode head) {
        if(head==null || head.next==null)
            return head;
        ListNode evenHead = head.next;
        ListNode evenNode = evenHead;
        ListNode oddNode = head;
        while(oddNode.next!=null && evenNode.next!=null)
        {
            oddNode.next = oddNode.next.next;
            oddNode = oddNode.next;
            evenNode.next = evenNode.next.next;
            evenNode = evenNode.next;
        }
        oddNode.next = evenHead;
        return head;
}
```

R-eview

- Trace through your code with an input to check for the expected output
- Catch possible edge cases and off-by-one errors

CODE PATH *ORG

E-valuate

- Time Complexity: O(n)
  where n is the number of nodes

- Space Complexity: O(1)
  We used an odd list and even list at the same time. We used two variables *odd_node* and *even_node* to control what is our current node in the odd list and even list.

CODE
PATH
*ORG

Breakout Sessions | 60 mins

Activity | Breakout Groups

60 MINS

Directions

1. **Introduce yourself!** Answer one of the icebreaker questions
   a. Are your a morning or evening person?
   b. What is a guilty pleasure of yours?

2. Work together through the UMPIRE steps of **Problem 1** in the Course Portal under Week 1, Session 2

Reminders

Don’t forget to turn on your cameras!

CODE
PATH
*ORG

Breakout Problem Review | 15 mins

Wrap Up | 5 mins

Activity | Exit Ticket

3 MINS

Answer the 3 multiple choice in the Zoom Poll.

Exit tickets are a great way for us to check your understanding of the topics we discussed in class and identify for you what you should review after class.

Topics: Linked lists

*Answers and explanations are available on the slides in the **Appendix** section

Exit Ticket: Question #1

Topic: Linked Lists

Question:

Which of the following operations is performed more efficiently by doubly linked list than by singly linked list?

a) Deleting a node whose location in given
b) Searching of an unsorted list for a given item
c) Finding an element given its index in the list
d) Traversing a list to process each node

Exit Ticket: Question #2

Topic: Linked Lists

Question:

What is the advantage of using linked list over an array?

a) Faster access to data
b) Not of a fixed size
c) Easier to use
d) Smaller Size

Exit Ticket: Question #3

Topic: Linked Lists

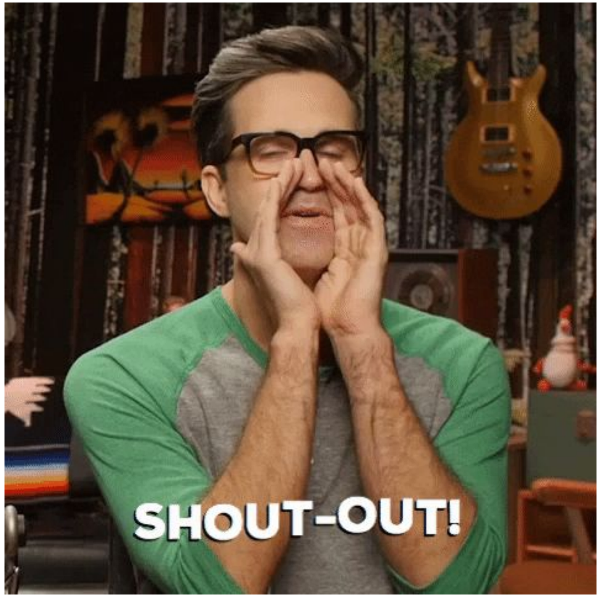
Question:

In doubly linked lists, traversal can be performed?

a) Only in forward direction
b) Only in reverse direction
c) In both directions
d) None of the above

CODE
PATH
*ORG

Shout Outs!



Take a moment to shoutout someone today who helped you out.

Alternatively, drop in the chat something new that you were excited to learn about today!

HackerRank Reminders

- ❏ You must use the **same** email you provided CodePath during application for your submission to be tracked
- ❏ HRs are due the day before Session #1 of the following week
  - ❏ Tues/Thurs class: Mondays at 11:59pm PDT
  - ❏ Wed/Sat class: Tuesdays at 11:59pm PDT
- ❏ You are allowed 2 missing/submitted late HRs, no questions asked

At CodePath we collect your scores as a way to track your progress and provide you feedback throughout the program. You are not graded in this course, these assignments are made solely for practice only.

All information is on the **assignment tab** on the course portal.

CODE PATH *ORG

Before you Leave |

- ❑ Complete the Session Survey [5 min]
- ❑ Next session is 06/07/22 5pm pst
- ❑ Complete your HackerRank assessment by **1 day before the next session** at 11:59pm PDT
- ❑ Explore the Week 1 Resources tab on the Course Portal
- ❑ Check out the Week 1 Career tab on: **Software Engineering Skills**

Appendix

Exit Ticket: Question #1

Topic: Linked Lists

Question:

Which of the following operations is performed more efficiently by doubly linked list than by singly linked list?

a) Deleting a node whose location in given
b) Searching of an unsorted list for a given item
c) Finding an element given its index in the list
d) Traversing a list to process each node

Answer:

a) Deleting a node whose location in given

Explanation:

In order to delete a node and connect the previous and the next node together, you need to know their pointers. In a doubly-linked list, both pointers are available in the node that is to be deleted. The time complexity is constant in this case, i.e., O(1). Whereas in a singly-linked list, the pointer to the previous node is unknown and can be found only by traversing the list from head until it reaches the node that has a next node pointer to the node that is to be deleted. The time complexity in this case is O(n).

Exit Ticket: Question #2

Topic: Linked Lists

Question:

What is the advantage of using linked list over an array?

a) Faster access to data
b) Not of a fixed size
c) Easier to use
d) Smaller Size

Answer: B

Explanation:

B) Linked lists are not of a fixed size. The memory to store nodes are allocated dynamically. Arrays are fixed in size and hold contiguous memory.

Exit Ticket: Question #3

Topic: Linked Lists

Question:

In doubly linked lists, traversal can be performed?

a) Only in forward direction
b) Only in reverse direction
c) In both directions
d) None of the above

Answer: c

Explanation:

```
As the doubly linked list
contains two pointers i.e.
previous and next, we can
traverse it into the
directions forward and
backward.
```