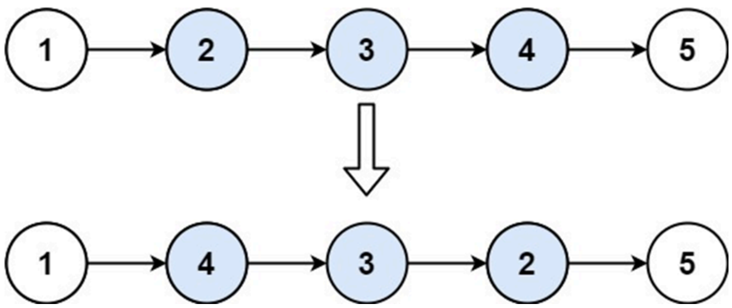
1. **Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return *the reversed list*.**

Example 1:



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

Example 2:

Input: head = [5], left = 1, right = 1

Output: [5]

Constraints:

- The number of nodes in the list is n.
- <= n <= 500
- -500 <= Node.val <= 500
- 1 <= left <= right <= n

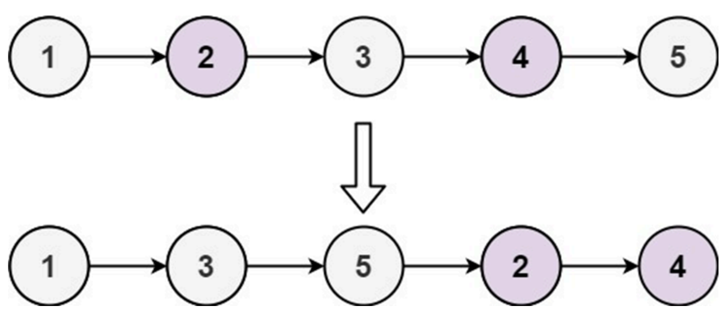
2 **Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.**

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.

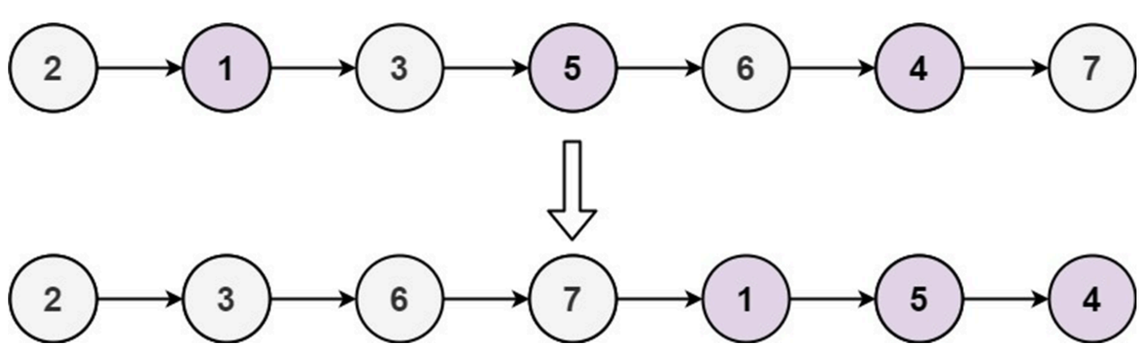
You must solve the problem in O(1) extra space complexity and O(n) time complexity.

Example 1:



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

Example 2:



Input: head = [2,1,3,5,6,4,7]
Output: [2,3,6,7,1,5,4]

Constraints:

- The number of nodes in the linked list is in the range [0, 10^4].
- -10^6 <= Node.val <= 10^6

3. Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return *its area.*

Example 1:

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0

Input:
matrix =
[["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]
Output: 6
Explanation: The maximal rectangle is shown in the above picture.

Example 2:

Input: matrix = [["0"]]

Output: 0

Example 3:

Input: matrix = [["1"]]
Output: 1

Constraints:

- rows == matrix.length
- cols == matrix[i].length
- 1 <= row, cols <= 200
- matrix[i][j] is '0' or '1'.

4. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

- **MinStack() initialises the stack object.**
- **void push(int val) pushes the element val onto the stack.**
- **void pop() removes the element on the top of the stack.**
- **int top() gets the top element of the stack.**
- **int getMin() retrieves the minimum element in the stack.**

You must implement a solution with O(1) time complexity for each function.

Example 1:

Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output

[null,null,null,null,-3,null,0,-2]

Explanation

MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2

Constraints:

- -2^{31} <= val <= 2^{31} – 1
- Methods pop, top and getMin operations will always be called on non-empty stacks.
- At most 3 * 10^4 calls will be made to push, pop, top, and getMin.

5. **Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).**

Implement the MyStack class:

- **void push(int x) Pushes element x to the top of the stack.**
- **int pop() Removes the element on the top of the stack and returns it.**
- **int top() Returns the element on the top of the stack.**
- **boolean empty() Returns true if the stack is empty, false otherwise**

Notes:

- **You must use only standard operations of a queue, which means that only push to back, peek/pop from front, size and is empty operations are valid.**
- **Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.**

Example 1:

Input
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 2, 2, false]

Explanation

MyStack myStack = new MyStack();
myStack.push(1);
myStack.push(2);
myStack.top(); // return 2
myStack.pop(); // return 2
myStack.empty(); // return False

Constraints:

- 1 <= x <= 9
- At most 100 calls will be made to push, pop, top, and empty.
- All the calls to pop and top are valid.

6. **Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).**

Implement the MyQueue class:

- **void push(int x) Pushes element x to the back of the queue.**
- **int pop() Removes the element from the front of the queue and returns it.**
- **int peek() Returns the element at the front of the queue.**
- **boolean empty() Returns true if the queue is empty, false otherwise.**

Notes:

- **You must use only standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.**
- **Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.**

Example 1:

Input
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
Output
[null, null, null, 1, 1, false]

Explanation

MyQueue myQueue = new MyQueue();
myQueue.push(1); // queue is: [1]
myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)

myQueue.peek(); // return 1
myQueue.pop(); // return 1, queue is [2]
myQueue.empty(); // return false

Constraints:

- 1 <= x <= 9
- At most 100 calls will be made to push, pop, peek, and empty.
- All the calls to pop and peek are valid.

All the best!!