Assignment 2

ArrayList:

1. Create an ArrayList of Strings and add some names to it. Use a loop to iterate over the list and print each name.
2. Create an ArrayList of integers and fill it with some random numbers. Use the Collections.sort() method to sort the list in ascending order.
3. Create a program that reads a list of words from the user, adds them to an ArrayList, and then sorts them in alphabetical order. Finally, print the sorted list of words.
4. Create a program that reads a list of integers from the user, adds them to an ArrayList, and then finds and prints the maximum and minimum values in the list.
5. Create a program that reads a list of numbers from a file, adds them to an ArrayList, and then calculates and prints the sum and average of the numbers in the list.
6. Create a program that reads a list of strings from a file, adds them to an ArrayList, and then removes any duplicates from the list.
7. Create a program that reads a list of student names and their grades from a file, adds them to an ArrayList, and then sorts the list by grade in descending order.
8. Create a program that reads a list of books from a file, adds them to an ArrayList, and then searches for a book by title. If the book is found, print its details; otherwise, print a message saying the book was not found.
9. Create a program that reads a list of employee names and salaries from a file, adds them to an ArrayList, and then calculates and prints the total salary of all employees.
10. Create a program that reads a list of dates from a file, adds them to an ArrayList, and then sorts the list in chronological order.

LinkedList:

1. Given the head of a singly linked list, return the middle node of the linked list.
   If there are two middle nodes, return the second middle node.

   Sample Example1:

   Input: head = [1,2,3,4,5]
   Output: [3,4,5]
   Explanation: The middle node of the list is node 3.

   Sample Example2:

   Input: head = [1,2,3,4,5,6]
   Output: [4,5,6]
   Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

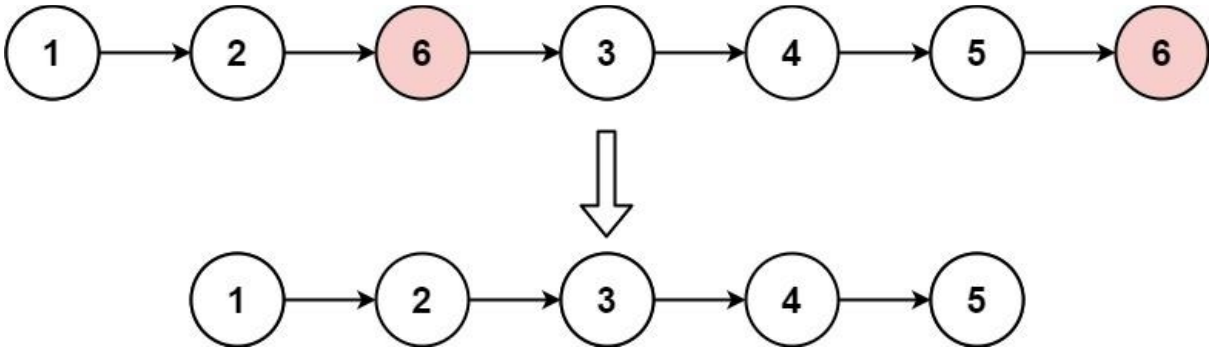
2. Given the head of a singly linked list, reverse the list, and return the reversed list by using iteration (loops).

Sample Example1:
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]

3. Solve the task 2 by using recursion.

4. Given the head of a linked list and an integer *val*, remove all the nodes of the linked list that has *Node.val == val*, and return the new head.

Sample Example1:



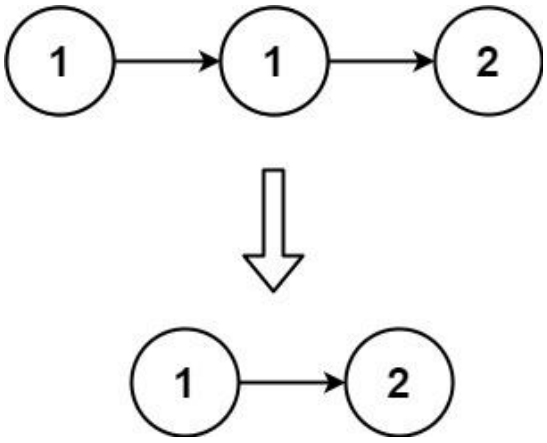
Input: head = [1,2,6,3,4,5,6], val = 6
Output: [1,2,3,4,5]

Sample Example2:

Input: head = [7,7,7,7], val = 7
Output: []

5. Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.

Sample Example1:



Input: head = [1,1,2]
Output: [1,2]

Sample Example2:
Input: head = [1,1,2,3,3]
Output: [1,2,3]

6. Design HashMap without using any built-in hash table libraries.

   Implement the MyHashMap class:

   - MyHashMap() initializes the object with an empty map.
     void put(int key, int value) inserts a (key, value) pair into the HashMap. If the key already exists in the map, update the corresponding value.
   - int get(int key) returns the value to which the specified key is mapped, or -1 if this map contains no mapping for the key.
   - void remove(key) removes the key and its corresponding value if the map contains the mapping for the key.

   Sample Input1:

   ["MyHashMap", "put", "put", "get", "get", "put", "get", "remove", "get"]
   [[], [1, 1], [2, 2], [1], [3], [2, 1], [2], [2], [2]]
   Output
   [null, null, null, 1, -1, null, 1, null, -1]

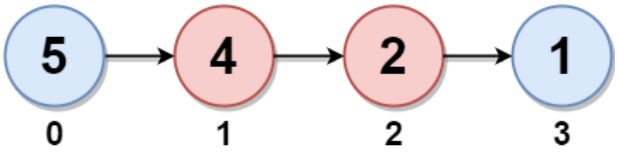
   Explanation
   MyHashMap myHashMap = new MyHashMap();
   myHashMap.put(1, 1); // The map is now [[1,1]]
   myHashMap.put(2, 2); // The map is now [[1,1], [2,2]]
   myHashMap.get(1);   // return 1, The map is now [[1,1], [2,2]]
   myHashMap.get(3);   // return -1 (i.e., not found), The map is now [[1,1], [2,2]]
   myHashMap.put(2, 1); // The map is now [[1,1], [2,1]] (i.e., update the existing value)
   myHashMap.get(2);   // return 1, The map is now [[1,1], [2,1]]
   myHashMap.remove(2); // remove the mapping for 2, The map is now [[1,1]]
   myHashMap.get(2);   // return -1 (i.e., not found), The map is now [[1,1]]

7. In a linked list of size n, where n is even, the i^{th} node (0-indexed) of the linked list is known as the twin of the $(n-1-i)^{th}$ node, if $0 <= i <= (n / 2) - 1$.

   For example, if n = 4, then node 0 is the twin of node 3, and node 1 is the twin of node 2. These are the only nodes with twins for n = 4.
   The twin sum is defined as the sum of a node and its twin.

Given the head of a linked list with even length, return the **maximum twin sum** of the linked list.

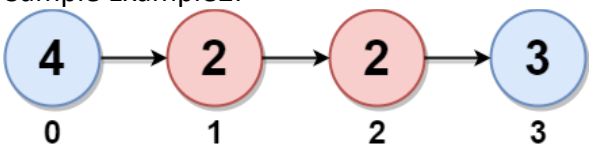
Sample Example1:



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

Explanation:
Nodes 0 and 1 are the twins of nodes 3 and 2, respectively. All have twin sum = 6.
There are no other nodes with twins in the linked list.
Thus, the maximum twin sum of the linked list is 6.

Sample Example2:



Input: head = [4,2,2,3]
Output: 7
Explanation:
The nodes with twins present in this linked list are:
- Node 0 is the twin of node 3 having a twin sum of 4 + 3 = 7.
- Node 1 is the twin of node 2 having a twin sum of 2 + 2 = 4.
Thus, the maximum twin sum of the linked list is max(7, 4) = 7.

8. Find the n^{th} node from the end: Given a linked list and a positive integer n, find the n^{th} node from the end of the list. This can be done by using two pointers, one that starts at the beginning of the list and another that starts n nodes ahead. The two pointers are then moved simultaneously until the second pointer reaches the end of the list, at which point the first pointer will be pointing to the nth node from the end.