Coding interview

Array DS

1. How do you find the missing number in a given integer array of 1 to 100?

Based on the sum of the numbers −

- The sum of n sequential numbers will be [n*(n+1)]/2. Using this get the sum of the numbers the n numbers.
- Add all the elements in the array.
- Subtract the sum of the numbers in the array from the sum of the n numbers.

Example

```
import java.util.Scanner;

public class MissingNumber {

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);

      System.out.println("Enter the n value: ");

      int n = sc.nextInt();

      int inpuArray[] = new int[n];

      System.out.println("Enter (n-1) numbers: ");

      for(int i=0; i<=n-2; i++) {

         inpuArray[i] = sc.nextInt();

      }

      //Finding the missing number

      int sumOfAll = (n*(n+1))/2;

      int sumOfArray = 0;

      for(int i=0; i<=n-2; i++) {

         sumOfArray = sumOfArray+inpuArray[i];

      }

      int missingNumber = sumOfAll-sumOfArray;

      System.out.println("Missing number is: "+missingNumber);

   }
```

```
}
```

Output

```
Enter the n value:
5
Enter (n-1) numbers:
1
2
4
5
Missing number is: 3
```

1. 2. How do you find the duplicate number on a given integer array?

Use two loops. In the outer loop, pick elements one by one and count the number of occurrences of the picked element in the inner loop.

This method doesn't use the other useful data provided in questions like range of numbers is between 1 to n and there are only two repeating elements.

```
class RepeatElement
{
    void printRepeating(int arr[], int size)
    {
        int i, j;
        System.out.println("Repeated Elements are :");
        for (i = 0; i < size; i++)
        {
            for (j = i + 1; j < size; j++)
            {
                if (arr[i] == arr[j])
                    System.out.print(arr[i] + " ");
            }
        }
    }

    public static void main(String[] args)
    {
        RepeatElement repeat = new RepeatElement();
        int arr[] = {4, 2, 4, 5, 2, 3, 1};
        int arr_size = arr.length;
        repeat.printRepeating(arr, arr_size);
    }
}
```

Time Complexity: O(n*n)
Auxiliary Space: O(1)

Method 2 (Use Count array)
Traverse the array once. While traversing, keep track of count of all elements in the array using a temp array count[] of size n, when you see an element whose count is already set, print it as duplicate.
This method uses the range given in the question to restrict the size of count[], but doesn't use the data that there are only two repeating elements.

Time Complexity: O(n)
Auxiliary Space: O(n)

3.How do you find the largest and smallest number in an unsorted integer array?

Method 1: Traverse the array iteratively and keep track of the smallest and largest element until the end of the array.

```
// Java program to find the smallest and largest element in an array

import java.util.*;

class Main
{
public static void main(String args[])
{
int large,small,i;
int a[] = new int[]{1, 2, 3, 4, 5};
int n = a.length;
large=small=a[0];
for(i=1;i<n;++i)
{
if(a[i]>large)
large=a[i];

if(a[i]<small)
small=a[i];
}

System.out.print("\nThe smallest element is " + small );
System.out.print("\nThe largest element is " + large );
}
```

}
O(n)

4.How do you find all pairs of an integer array whose sum is equal to a given number?

A **simple solution** is to traverse each element and check if there's another number in the array which can be added to it to give sum.

```
class GFG {

    // Returns number of pairs
    // in arr[0..n-1] with sum
    // equal to 'sum'
    static void printPairs(int arr[],
                           int n, int sum)
    {
        // int count = 0;

        // Consider all possible pairs
        // and check their sums
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                if (arr[i] + arr[j] == sum)
                    System.out.println("(" + arr[i] + ", " + arr[j] +
")");
    }

    // Driver Code
    public static void main(String[] arg)
    {
        int arr[] = { 1, 5, 7, -1, 5 };
        int n = arr.length;
        int sum = 6;
        printPairs(arr, n, sum);
    }
}

O(n^2)
```

5.How do you find duplicate numbers in an array if it contains multiple duplicates?

Simple Approach: The idea is to use nested loop and for each element check if the element is present in the array more than once or not. If present, then store it in a Hash-map. Otherwise, continue checking other elements.

```
 static void findDuplicates(
      int arr[], int len)
    {
```

```
        // initialize ifPresent as false
        boolean ifPresent = false;

        // ArrayList to store the output
        ArrayList<Integer> al = new ArrayList<Integer>();

        for (int i = 0; i < len - 1; i++) {
            for (int j = i + 1; j < len; j++) {
                if (arr[i] == arr[j]) {
                    // checking if element is
                    // present in the ArrayList
                    // or not if present then break
                    if (al.contains(arr[i])) {
                        break;
                    }

                    // if element is not present in the
                    // ArrayList then add it to ArrayList
                    // and make ifPresent at true
                    else {
                        al.add(arr[i]);
                        ifPresent = true;
                    }
                }
            }
        }

        // if duplicates is present
        // then print ArrayList
        if (ifPresent == true) {

            System.out.print(al + " ");
        }
        else {
            System.out.print(
                "No duplicates present in arrays");
        }
    }

    // Driver Code
    public static void main(String[] args)
    {

        int arr[] = { 12, 11, 40, 12, 5, 6, 5, 12, 11 };
        int n = arr.length;

        findDuplicates(arr, n);
    }
}
```

6.How are duplicates removed from a given array in Java?

Method 1: (Using extra space)

1. Create an auxiliary array temp[] to store unique elements.

2. Traverse input array and one by one copy unique elements of arr[] to temp[]. Also keep track of count of unique elements. Let this count be **j**.
3. Copy **j** elements from temp[] to arr[] and return j

```
class Main
{
    // Function to remove duplicate elements
    // This function returns new size of modified
    // array.
    static int removeDuplicates(int arr[], int n)
    {
        // Return, if array is empty
        // or contains a single element
        if (n==0 || n==1)
            return n;

        int[] temp = new int[n];

        // Start traversing elements
        int j = 0;
        for (int i=0; i<n-1; i++)
            // If current element is not equal
            // to next element then store that
            // current element
            if (arr[i] != arr[i+1])
                temp[j++] = arr[i];

        // Store the last element as whether
        // it is unique or repeated, it hasn't
        // stored previously
        temp[j++] = arr[n-1];

        // Modify original array
        for (int i=0; i<j; i++)
            arr[i] = temp[i];

        return j;
    }

    public static void main (String[] args)
    {
        int arr[] = {1, 2, 2, 3, 4, 4, 4, 5, 5};
        int n = arr.length;

        n = removeDuplicates(arr, n);

        // Print updated array
        for (int i=0; i<n; i++)
           System.out.print(arr[i]+" ");
    }
```

Time Complexity : O(n)

Auxiliary Space : O(n)

7.How do you reverse an array in place in Java?

The **second method** uses the similar code for the inputting and printing of the array. However, we don't create a new array like the above method. Instead, we reverse the original array itself. In this method we swap the elements of the array. The first element is swapped with the last element. The second element id swapped with the last but one element and so on. For instance, consider array [1, 2, 3, …., n-2, n-1, n]. We swap 1 with n, 2 with n-1, 3 with n-2 and further.

```
public class arrayReverse {

    /*function swaps the array's first element with last element,
      second element with last second element and so on*/
    static void reverse(int a[], int n)
    {
        int i, k, t;
        for (i = 0; i < n / 2; i++) {
            t = a[i];
            a[i] = a[n - i - 1];
            a[n - i - 1] = t;
        }

        /*printing the reversed array*/
        System.out.println("Reversed array is: \n");
        for (k = 0; k < n; k++) {
            System.out.println(a[k]);
        }
    }

    public static void main(String[] args)
    {
        int [] arr = {10, 20, 30, 40, 50};
        reverse(arr, arr.length);
    }
}
```

The **third method** is to use the function java.util.Collections.reverse(List list) method. This method reverses the elements in the specified list. Hence, we convert the array into a list first by using java.util.Arrays.asList(array) and then reverse the list.

```
// Reversing an array using Java collections.

import java.util.*;
```

```
public class reversingArray {



    /*function reverses the elements of the array*/

    static void reverse(Integer a[])

    {

        Collections.reverse(Arrays.asList(a));

        System.out.println(Arrays.asList(a));

    }



    public static void main(String[] args)

    {

        Integer [] arr = {10, 20, 30, 40, 50};

        reverse(arr);

    }

}
```

The **first method** is as follows:
(i) Take input the size of array and the elements of array.
(ii) Consider a function reverse which takes the parameters-the array(say arr) and the size of the array(say n).
(iii) Inside the function, a new array (with the array size of the first array, arr) is initialized. The array arr[] is iterated from the first element and each element of array arr[] is placed in the new array from the back, i.e, the new array is iterated from its last element.
(iv) In this way, all the elements of the array arr[] are placed reversely in the new array.

(v) Further, we can iterate through the new array from the beginning and print the elements of the array.

```
public class reverseArray {

    /* function that reverses array and stores it
       in another array*/
    static void reverse(int a[], int n)
    {
        int[] b = new int[n];
        int j = n;
        for (int i = 0; i < n; i++) {
            b[j - 1] = a[i];
            j = j - 1;
        }

        /*printing the reversed array*/
        System.out.println("Reversed array is: \n");
        for (int k = 0; k < n; k++) {
            System.out.println(b[k]);
        }
    }

    public static void main(String[] args)
    {
        int [] arr = {10, 20, 30, 40, 50};
        reverse(arr, arr.length);
    }
}
```

Linked List

11. How do you find the middle element of a singly linked list in one pass?

Given a singly linked list, find the middle of the linked list. For example, if the given linked list is 1->2->3->4->5 then the output should be 3.
If there are even nodes, then there would be two middle nodes, we need to print the second middle element. For example, if given linked list is 1->2->3->4->5->6 then output should be 4.

Method 1:
Traverse the whole linked list and count the no. of nodes. Now traverse the list again till count/2 and return the node at count/2.
Method 2:
Traverse linked list using two pointers. Move one pointer by one and the other pointers by two. When the fast pointer reaches the end slow pointer will reach the middle of the linked list.

```
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

/* Function to get the middle of the linked list*/
void printMiddle(struct Node *head)
{
    struct Node *slow_ptr = head;
    struct Node *fast_ptr = head;

    if (head!=NULL)
    {
        while (fast_ptr != NULL && fast_ptr->next != NULL)
        {
            fast_ptr = fast_ptr->next->next;
            slow_ptr = slow_ptr->next;
        }
        printf("The middle element is [%d]\n\n", slow_ptr->data);
    }
}

void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
        (struct Node*) malloc(sizeof(struct Node));
```

```
    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

// A utility function to print a given linked list
void printList(struct Node *ptr)
{
    while (ptr != NULL)
    {
        printf("%d->", ptr->data);
        ptr = ptr->next;
    }
    printf("NULL\n");
}

/* Driver program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;
    int i;

    for (i=5; i>0; i--)
    {
        push(&head, i);
        printList(head);
        printMiddle(head);
    }

    return 0;
}
```

11. How do you check if a given linked list contains a cycle? How do you find the starting node of the cycle?

<u>Solution 2</u>: This problem can be solved without hashmap by modifying the linked list data-structure.
Approach: This solution requires modifications to the basic linked list data structure.

- Have a visited flag with each node.
- Traverse the linked list and keep marking visited nodes.
- If you see a visited node again then there is a loop. This solution works in O(n) but requires additional information with each node.
- A variation of this solution that doesn't require modification to basic data structure can be implemented using a hash, just store the

addresses of visited nodes in a hash and if you see an address that already exists in hash then there is a loop.

```
// C++ program to detect loop in a linked list

#include <bits/stdc++.h>

using namespace std;



/* Link list node */

struct Node {

    int data;

    struct Node* next;

    int flag;

};



void push(struct Node** head_ref, int new_data)

{

    /* allocate node */

    struct Node* new_node = new Node;



    /* put in the data */

    new_node->data = new_data;
```

```
    new_node->flag = 0;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

// Returns true if there is a loop in linked list
// else returns false.
bool detectLoop(struct Node* h)
{
    while (h != NULL) {
        // If this node is already traverse
        // it means there is a cycle
        // (Because you we encountering the
        // node for the second time).
        if (h->flag == 1)
            return true;

        // If we are seeing the node for
```

```
            // the first time, mark its flag as 1
            h->flag = 1;

            h = h->next;
        }

        return false;
}

/* Driver program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    push(&head, 20);
    push(&head, 4);
    push(&head, 15);
    push(&head, 10);

    /* Create a loop for testing */
```

```
    head->next->next->next->next = head;



    if (detectLoop(head))

        cout << "Loop found";

    else

        cout << "No Loop";



    return 0;

}

// This code is contributed by Geetanjali
```

Output:

```
Loop Found
```

Complexity Analysis:

- **Time complexity:**O(n).
  Only one traversal of the loop is needed.
- **Auxiliary Space:**O(1).
  No extra space is needed.

13.How do you reverse a linked list?

To reverse the given linked list we will use three extra pointers that will be in the process. The pointers will be previous, after, current.

We will initialize previous and after to NULL initially and current to head of the linked list.

After this, we will iterate until we reach the NULL of the initial (non-reversed linked list). And do the following −

```
after = current ->
next current ->
next = previous
previous = current
```

```
current = after
```

```
#include <stdio.h>
struct Node {
   int data;
   struct Node* next;
   Node(int data){
      this->data = data;
      next = NULL;
   }
};
struct LinkedList {
   Node* head;
   LinkedList(){
      head = NULL;
   }
   void interReverseLL(){
      Node* current = head;
      Node *prev = NULL, *after = NULL;
      while (current != NULL) {
         after = current->next;
         current->next = prev;
         prev = current;
         current = after;
      }
      head = prev;
   }
   void print() {
```

```
      struct Node* temp = head;
      while (temp != NULL) {
         printf("%d ", temp-> data);
         temp = temp->next;
      }
      printf("\n");
   }
   void push(int data){
      Node* temp = new Node(data);
      temp->next = head;
      head = temp;
   }
};
int main() {
   LinkedList linkedlist;
   linkedlist.push(85);
   linkedlist.push(10);
   linkedlist.push(65);
   linkedlist.push(32);
   linkedlist.push(9);
   printf("Linked List : \t");
   linkedlist.print();
   linkedlist.interReverseLL();
   printf("Reverse Linked List : \t");
   linkedlist.print();
   return 0;
```

14.How do you find the length of a singly linked list?

```
int getCount(Node* head)
```

```
{
    int count = 0; // Initialize count
    Node* current = head; // Initialize current
    while (current != NULL)
    {
        count++;
        current = current->next;
    }
    return count;
}
```

15.How do you find the third node from the end in a singly linked list?