Q1. Given an array of 0's and 1's, in O(n) time make all the 0's in one side and all the 1's in other side.

```
#include <stdio.h>
void Arrange(int a[], int size)
{
    int l = 0, r = size - 1;

    while (l < r)
    {
        while (a[l] == 0 && l < r)
            l++;
        while (a[r] == 1 && l < r)
            r--;
        if (l < r)
        {
            a[l] = 0;
            a[r] = 1;
            l++;
            r--;
        }
    }
}

int main()
{
    int arr[] = {0, 1, 0, 1, 1, 0};
    int a_s = 6, i = 0;

    Arrange(arr, a_s);
    printf("Arranged array is ");
    for (i = 0; i < 6; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

OUTPUT

```
Arranged array is 0 0 0 1 1 1
```

Q2. You are given an array of n+2 elements of the array occur once only between 1 to n, except two nos. which occur twice. find the two repeating nos. in O(n) time complexity.

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

```c
void prtRpting(int a[], int size)
{
    int i, j;
    printf("Repeating elements are ");
    for (i = 0; i < size; i++)
        for (j = i + 1; j < size; j++)
            if (a[i] == a[j])
                printf(" %d ", a[i]);
}

int main()
{
    int a[] = {6, 4, 6, 7, 4, 5, 3};
    int a_s = sizeof(a) / sizeof(a[0]);
    prtRpting(a, a_s);

    return 0;
}
```

OUTPUT

```
Repeating elements are  6  4
```

Q3. Given an array, write a program that split even and odd nos.The program should put even nos. first and then odd nos.

```c
#include <stdio.h>
void swap(int *a, int *b);

void seg(int arr[], int size)
{
    int l = 0, r = size - 1;
    while (l < r)
    {
        while (arr[l] % 2 == 0 && l < r)
            l++;
        while (arr[r] % 2 == 1 && l < r)
            r--;
        if (l < r)
        {
            swap(&arr[l], &arr[r]);
            l++;
            r--;
        }
    }
}
```

```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
int main()
{
    int arr[] = {2, 3, 4, 5, 6};
    int a_s = sizeof(arr) / sizeof(arr[0]);
    int i = 0;
    seg(arr, a_s);
    printf("Array after Rearanging ");
    for (i = 0; i < a_s; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

OUTPUT

Array after Rearanging 2 6 4 5 3

Q4. You are given the pointer to the head nodes of two link lists. Compare the data in the nodes of the link do check if they are equal or not. If they are equal return 1 else 0.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
struct Node
{
    int data;
    struct Node *next;
};
bool areIdentical(struct Node *a, struct Node *b)
{

    if (a == NULL && b == NULL)
        return true;
    if (a != NULL && b != NULL)
        return (a->data == b->data) && areIdentical(a->next, b->next);
    return false;
}
void push(struct Node **head_ref, int new_data)
{
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;
```

```
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
int main()
{
    struct Node *a = NULL;
    struct Node *b = NULL;
    push(&a, 1);
    push(&a, 2);
    push(&a, 3);
    push(&b, 1);
    push(&b, 2);
    push(&b, 3);
    // push(&b, 4);

    areIdentical(a, b) ? printf("Identical - 1") : printf("Not identical - 0");

    return 0;
}
```

OUTPUT

1. Identical – 1
2. Not identical - 0

Q5. Return the lowest common ancestor(LCA) if node values are given for a tree.

```
#include <stdio.h>
#include <stdlib.h>
#define COUNT 7
struct node
{
    int data;
    struct node *left, *right;
};
struct node *lca(struct node *root, int n1, int n2)
{
    while (root != NULL)
    {
        if (root->data > n1 && root->data > n2)
            root = root->left;
        else if (root->data < n1 && root->data < n2)
            root = root->right;

        else
            break;
```

```
    }
    return root;
}
struct node *newNode(int data)
{
    struct node *node = (struct node *)malloc(sizeof(struct node));
    node->data = data;
    node->left = node->right = NULL;
    return (node);
}
void print2DUtil(struct node *root, int space)
{
    if (root == NULL)
        return;
    space += COUNT;
    print2DUtil(root->right, space);
    printf("");
    for (int i = COUNT; i < space; i++)
        printf(" ");
    printf("%d\n", root->data);
    print2DUtil(root->left, space);
};
void print2D(struct node *root)
{
    print2DUtil(root, 0);
}

int main()
{
    struct node *root = newNode(20);
    root->left = newNode(8);
    root->right = newNode(22);
    root->left->left = newNode(4);
    root->left->right = newNode(12);
    root->left->right->left = newNode(10);
    root->left->right->right = newNode(14);
    print2D(root);
    int n1 = 10, n2 = 14;
    struct node *t = lca(root, n1, n2);
    printf("LCA of %d and %d is %d \n", n1, n2, t->data);
    n1 = 10, n2 = 12;
    t = lca(root, n1, n2);
    printf("LCA of %d and %d is %d \n", n1, n2, t->data);
    n1 = 10, n2 = 22;
    t = lca(root, n1, n2);
```

```
    printf("LCA of %d and %d is %d \n", n1, n2, t->data);
    return 0;
}
```

OUTPUT

```
        22
20
                    14
            12
                    10
        8
            4
LCA of 10 and 14 is 12
LCA of 10 and 12 is 12
LCA of 10 and 22 is 20
```

Q6. Given a Binary tree check if its BST or not

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

struct node
{
    int data;
    struct node *left;
    struct node *right;
};

int isBSTUtil(struct node *node, int min, int max);

int isBST(struct node *node)
{
    return (isBSTUtil(node, INT_MIN, INT_MAX));
}

int isBSTUtil(struct node *node, int min, int max)
{

    if (node == NULL)
        return 1;

    if (node->data < min || node->data > max)
        return 0;
    return isBSTUtil(node->left, min, node->data - 1) && isBSTUtil(node->right, node->data + 1, max);
```

```
}

struct node *newNode(int data)
{
    struct node *node = (struct node *) malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return (node);
}

int main()
{
    struct node *root = newNode(4);
    root->left = newNode(2);
    root->right = newNode(5);
    root->left->left = newNode(1);
    root->left->right = newNode(3);
    // root->left->right = newNode(6);

    if (isBST(root))
        printf("Is BST");
    else
        printf("Not a BST");
    return 0;
}
```

OUTPUT

1. Is BST
2. Not a BST
