1. Find the Missing Number

Given an array of size `n-1` with elements from 1 to `n`, find the missing number in the array.
Input: `arr = {1, 2, 4, 6, 3, 7, 8}, n = 8`
Output: `5`

Hint: Use the formula for the sum of the first `n` natural numbers.

```
#include <stdio.h>

int findMissingNumber(int arr[], int n) {
    int total = n * (n + 1) / 2;
    int sum = 0;
    for (int i = 0; i < n - 1; i++) {
        sum += arr[i];
    }
    return total - sum;
}

int main() {
    int arr[] = {1, 2, 4, 6, 3, 7, 8};
    int n = 8;
    printf("Missing Number: %d\n", findMissingNumber(arr, n));
    return 0;
}
```

---

2. Rearrange Array Alternately

Rearrange a sorted array such that elements are arranged in alternating max and min form.
Input: `arr = {1, 2, 3, 4, 5, 6, 7}`
Output: `{7, 1, 6, 2, 5, 3, 4}`

Hint: Use two pointers to place elements from both ends.

```
#include <stdio.h>

void rearrangeArray(int arr[], int n) {
    int temp[n];
    int i = 0, j = n - 1, k = 0;

    while (i <= j) {
        if (k % 2 == 0) {
            temp[k++] = arr[j--];
        } else {
            temp[k++] = arr[i++];
        }
    }

    for (int i = 0; i < n; i++) {
        arr[i] = temp[i];
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    rearrangeArray(arr, n);
    printf("Rearranged Array: ");
    printArray(arr, n);

    return 0;
}
```

---

3. Find the Largest Sum Contiguous Subarray

Find the contiguous subarray within an array (containing at least one number) that has the largest sum.
Input: `arr = {-2, -3, 4, -1, -2, 1, 5, -3}`
Output: `7` (Subarray: `{4, -1, -2, 1, 5}`)

Hint: Use **Kadane's**

```
#include <stdio.h>

#include <limits.h>



int maxSubArraySum(int arr[], int n) {

    int max_sum = INT_MIN, current_sum = 0;
    for (int i = 0; i < n; i++) {
        current_sum += arr[i];
        if (current_sum > max_sum) {
            max_sum = current_sum;
        }
        if (current_sum < 0) {
            current_sum = 0;
        }
    }
    return max_sum;
}


int main() {
    int arr[] = {-2, -3, 4, -1, -2, 1, 5, -3};
    int n = sizeof(arr) / sizeof(arr[0]);
   printf("Maximum Sum: %d\n", maxSubArraySum(arr, n));
    return 0;
}}
```

---

4. Array Rotation

Write a function to rotate an array to the left by `d` positions.
Input: `arr = {1, 2, 3, 4, 5, 6, 7}`, `d = 2`
Output: `{3, 4, 5, 6, 7, 1, 2}`

Hint: Use slicing or a temporary array.

```
#include <stdio.h>

void rotateArray(int arr[], int n, int d) {
    int temp[d];
    for (int i = 0; i < d; i++) {
        temp[i] = arr[i];
    }
    for (int i = d; i < n; i++) {
        arr[i - d] = arr[i];
    }
    for (int i = 0; i < d; i++) {
        arr[n - d + i] = temp[i];
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);once
    int d = 2;

    rotateArray(arr, n, d);
    printf("Rotated Array: ");
    printArray(arr, n);

    return 0;
}
```

---

5. Find the Element that Appears Once

All elements in an array appear twice except for one. Find the element that appears only once.
Input: `arr = {2, 3, 7, 3, 2}`
Output: `7`

Hint: Use XOR properties.

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

```
int findSingle(int arr[], int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        result ^= arr[i];
    }
    return result;
}

int main() {
    int arr[] = {2, 3, 7, 3, 2};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Element Appearing Once: %d\n", findSingle(arr, n));
    return 0;
}
```

---

6. Check if Array is a Palindrome

Check if the elements of an array form a palindrome.
Input: `arr = {1, 2, 3, 2, 1}`
Output: `Yes`

Hint: Compare elements from start and end.

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

bool isPalindrome(int arr[], int n) {
    for (int i = 0; i < n / 2; i++) {
        if (arr[i] != arr[n - i - 1]) {
            return false;
        }
    }
    return true;
}

int main() {
    int arr[] = {1, 2, 3, 2, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    if (isPalindrome(arr, n)) {
        printf("Array is a Palindrome\n");
    } else {
        printf("Array is not a Palindrome\n");
    }
    return 0;
}
```

---

7. Merge Two Sorted Arrays

Merge two sorted arrays into one sorted array without using extra space.
Input: `arr1 = {1, 3, 5}`, `arr2 = {2, 4, 6}`
Output: `{1, 2, 3, 4, 5, 6}`

Hint: Use the merge step from merge sort.

```
#include <stdio.h>

void mergeSortedArrays(int arr1[], int n1, int arr2[], int n2, int merged[]) {
    int i = 0, j = 0, k = 0;

    while (i < n1 && j < n2) {
        if (arr1[i] < arr2[j]) {
            merged[k++] = arr1[i++];
        } else {
            merged[k++] = arr2[j++];
        }
    }

    while (i < n1) {
        merged[k++] = arr1[i++];
    }
    while (j < n2) {
        merged[k++] = arr2[j++];
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr1[] = {1, 3, 5};
    int arr2[] = {2, 4, 6};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    int merged[n1 + n2];

    mergeSortedArrays(arr1, n1, arr2, n2, merged);

    printf("Merged Array: ");
    printArray(merged, n1 + n2);

    return 0;
}
```

---

8. Find the Maximum Product Subarray

Find the subarray with the maximum product in a given array.
Input: `arr = {2, 3, -2, 4}`
Output: `6` (Subarray: `{2, 3}`)

Hint: Track maximum and minimum products at each step.

```
#include <stdio.h>

int maxProductSubarray(int arr[], int n) {
    int max_product = arr[0], min_product = arr[0], result = arr[0];

    for (int i = 1; i < n; i++) {
        if (arr[i] < 0) {
            int temp = max_product;
            max_product = min_product;
            min_product = temp;
        }

        max_product = (arr[i] > arr[i] * max_product) ? arr[i] : arr[i] * max_product;
        min_product = (arr[i] < arr[i] * min_product) ? arr[i] : arr[i] * min_product;

        if (max_product > result) {
            result = max_product;
        }
    }

    return result;
}

int main() {
    int arr[] = {2, 3, -2, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Maximum Product Subarray: %d\n", maxProductSubarray(arr, n));
    return 0;
}
```

---

9. Find the Majority Element

Find the element that appears more than `n/2` times in the array.
Input: `arr = {2, 2, 1, 1, 2, 2}`
Output: `2`

Hint: Use the **Moore Voting Algorithm**.

```
#include <stdio.h>

int findMajorityElement(int arr[], int n) {
    int candidate = -1, count = 0;

    // Find the candidate
    for (int i = 0; i < n; i++) {
        if (count == 0) {
            candidate = arr[i];
            count = 1;
        } else if (arr[i] == candidate) {
            count++;
        } else {
            count--;
        }
    }

    // Verify the candidate
    count = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] == candidate) {
            count++;
        }
    }

    return (count > n / 2) ? candidate : -1;
}

int main() {
    int arr[] = {2, 2, 1, 1, 2, 2};
    int n = sizeof(arr) / sizeof(arr[0]);

    int result = findMajorityElement(arr, n);
    if (result != -1) {
        printf("Majority Element: %d\n", result);
    } else {
        printf("No Majority Element\n");
    }

    return 0;
}
```

---

10. Segregate 0s and 1s

Given a binary array, segregate 0s and 1s in linear time.
Input: `arr = {0, 1, 0, 1, 0, 1, 1}`
Output: `{0, 0, 0, 1, 1, 1, 1}`

Hint: Count 0s and 1s or use two pointers.

```
#include <stdio.h>


void segregate0s1s(int arr[], int n) {

    int left = 0, right = n - 1;

    while (left < right) {

        while (arr[left] == 0 && left < right) {

            left++;

        }

        while (arr[right] == 1 && left < right) {

            right--;

        }

        if (left < right) {

            arr[left] = 0;

            arr[right] = 1;

            left++;

            right--;

        }

    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {0, 1, 0, 1, 0, 1, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    segregate0s1s(arr, n);
    printf("Segregated Array: ");
    printArray(arr, n);

    return 0;
}
```