1. Reverse a Number (Recursive)

C
```c
int reverse(int num) {
    if (num == 0) {
        return 0;
    }
    return (num % 10) * 10 + reverse(num / 10);
}

int main() {
    int number;
    printf("Enter a number: ");
    scanf("%d", &number);

    int reversed = reverse(number);
    printf("Reversed number: %d\n", reversed);

    return 0;
}
```

2. Find Perfect Number

C
```c
int isPerfect(int num) {
    int sum = 0;
    for (int i = 1; i < num; i++) {
        if (num % i == 0) {
            sum += i;
        }
    }
    return (sum == num);
}

int main() {
    int number;
    printf("Enter a number: ");
    scanf("%d", &number);

    if (isPerfect(number)) {
        printf("%d is a perfect number.\n", number);
    } else {
        printf("%d is not a perfect number.\n", number);
    }

    return 0;
}
```

3. Time Complexity Analysis (Example)

Example Algorithm: Linear search

C
```c
int linearSearch(int arr[], int n, int x) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == x) {
            return i;
        }
```

```
    }
    return -1;
}
```
Use code with caution.
content_copy

Time Complexity: O(n)

4. Non-Recursive vs. Recursive Algorithm Analysis (Example)

Example: Factorial

Non-Recursive

C
```
int factorial(int n) {
    int fact = 1;
    for (int i = 2; i <= n; i++) {
        fact *= i;
    }
    return fact;
}
```

Time Complexity: O(n)

Recursive

C
```
int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Time Complexity: O(n!)

6. Intersection of Two Arrays (Unique Elements)

C
```
#include <stdbool.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int removeDuplicates(int nums[], int *size) {
    int i = 0;
    for (int j = 1; j < *size; j++) {
        if (nums[i] != nums[j]) {
            i++;
            nums[i] = nums[j];
        }
```

```
    }
    return i + 1;
}

int *intersection(int *nums1, int nums1Size, int *nums2, int nums2Size, int
*returnSize) {
    int *result = (int *)malloc(sizeof(int) * (nums1Size + nums2Size)); //
Allocate memory for intersection
    int k = 0; // Index for result array

    // Sort both arrays for efficient comparison
    for (int i = 0; i < nums1Size - 1; i++) {
        for (int j = 0; j < nums1Size - i - 1; j++) {
            if (nums1[j] > nums1[j + 1]) {
                swap(&nums1[j], &nums1[j + 1]);
            }
        }
    }
    for (int i = 0; i < nums2Size
```

7. Intersection of Two Arrays (All Occurrences)

C

```
#include <stdbool.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int removeDuplicates(int nums[], int *size) {
    int i = 0;
    for (int j = 1; j < *size; j++) {
        if (nums[i] != nums[j]) {
            i++;
            nums[i] = nums[j];
        }
    }
    return i + 1;
}

int *intersection(int *nums1, int nums1Size, int *nums2, int nums2Size, int
*returnSize) {
    int *result = (int *)malloc(sizeof(int) * (nums1Size + nums2Size)); //
Allocate memory for intersection
    int k = 0; // Index for result array

    // Sort both arrays for efficient comparison
    for (int i = 0; i < nums1Size - 1; i++) {
        for (int j = 0; j < nums1Size - i - 1; j++) {
            if (nums1[j] > nums1[j + 1]) {
                swap(&nums1[j], &nums1[j + 1]);
            }
        }
    }
    for (int i = 0; i < nums2Size - 1; i++) {
        for (int j = 0; j < nums2Size - i - 1; j++) {
            if (nums2[j] > nums2[j + 1]) {
                swap(&nums2[j], &nums2[j + 1]);
```

```
            }
        }
    }

    // Find intersection elements (counting occurrences)
    int i = 0, j = 0;
    int count = 0; // Count occurrences of current intersection element
    while (i < nums1Size && j < nums2Size) {
        if (nums1[i] < nums2[j]) {
            i++;
        } else if (nums1[i] > nums2[j]) {
            j++;
        } else { // nums1[i] == nums2[j] (intersection element)
            result[k++] = nums1[i];
            count++;  // Increment count for current element
            i++;
            j++;
        }
    }

    // Handle remaining elements in the larger array (if any)
    while (i < nums1Size) {
        result[k++] = nums1[i++];
    }
    while (j < nums2Size) {
        result[k++] = nums2[j++];
    }

    // Reduce result size if necessary (remove duplicates due to counting)
    *returnSize = k;
    for (i = 1; i < *returnSize; i++) {
        if (result[i] == result[i - 1]) {
            // Shift elements to remove duplicate
            for (int j = i; j < *returnSize - 1; j++) {
                result[j] = result[j + 1];
            }
            (*returnSize)--; // Decrement size for removed element
            i--; // Decrement i to check the shifted element
        }
    }

    return result;
}

int main() {
    int nums1[] = {1, 2, 2, 1};
    int nums2[] = {2, 2};
    int nums1Size = sizeof(nums1) / sizeof(nums1[0]);
    int nums2Size = sizeof(nums2) / sizeof(nums2[0]);
    int returnSize = 0;

    int *intersectionResult = intersection(nums1, nums1Size, nums2,
nums2Size, &returnSize);

    printf("Intersection (all occurrences): ");
    for (int i = 0; i < returnSize; i++) {
        printf("%d ", intersectionResult[i]);
    }
    printf("\n");

    free(intersectionResult); // Free allocated memory
```

```
    return 0;
}
```

8. Sort Array (No Built-in Functions, O(n log n)) - Merge Sort (continued)

C

```
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Create temporary arrays
    int leftArr[n1], rightArr[n2];

    // Copy data to temporary arrays
    for (int i = 0; i < n1; i++) {
        leftArr[i] = arr[left + i];
    }
    for (int j = 0; j < n2; j++) {
        rightArr[j] = arr[mid + 1 + j];
    }

    // Merge the temporary arrays back into arr[left..right]
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        } else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements (if any)
    while (i < n1) {
        arr[k] = leftArr[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    int arr[] = {6, 5, 3, 1, 8, 7, 2, 4};
    int n = sizeof(arr) / sizeof(arr[0]);

    mergeSort(arr, 0, n - 1);
```

```
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

9. Sort Array by Odd-Even Positions

C

```
void sortOddEven(int arr[], int n) {
    int i = 0, j = 1;
    while (i < n && j < n) {
        // Find the first even number (if any)
        while (i < n && arr[i] % 2 != 0) {
            i += 2;
        }

        // Find the first odd number (if any) after the even number
        while (j < n && arr[j] % 2 == 0) {
            j += 2;
        }

        // Swap if both even and odd numbers are found
        if (i < n && j < n) {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
}

int main() {
    int arr[] = {4, 1, 3, 2, 16, 9};
    int n = sizeof(arr) / sizeof(arr[0]);

    sortOddEven(arr, n);

    printf("Sorted array (odd-even positions): ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

10. SORTING AN ARRAY

```
int removeDuplicates(int nums[], int n) {
    if (n == 0 || n == 1) {
        return n;
    }
```

```
    int j = 1;
    for (int i = 1; i < n; i++) {
        if (nums[i] != nums[i - 1]) {
            nums[j] = nums[i];
            j++;
        }
    }

    return j;
}

int main() {
    int nums[] = {1, 1, 2, 3, 5, 5};
    int n = sizeof(nums) / sizeof(nums[0]);

    int newSize = removeDuplicates(nums, n);

    printf("Array after removing duplicates: ");
    for (int i = 0; i < newSize; i++) {
        printf("%d ", nums[i]);
    }
    printf("\n");

    return 0;
}
```