# Topic - Rotation Algorithms / Techniques

Rotation is some of the basic operations that we perform on array, There are multiple ways to perform rotation on array.

- One by One Rotation
- Reversal Algorithm
- Juggling Algotihm

## One By One Rotation Technique

### Introduction

The method for rotating the elements of an array in a circular fashion. It shifts the elements of the array one position to the left or right, depending on the direction of rotation.

### Rotation Direction

The rotation can be performed in two directions:

- **Left Rotation:** In this mode, the elements are shifted to the left. The last element of the array becomes the first, and all other elements are moved one position to the left.
- **Right Rotation:** In this mode, the elements are shifted to the right. The first element of the array becomes the last, and all other elements are moved one position to the right.

### Implementation

**Left Rotation** To perform a left rotation on an array, follow these steps:

1. Store the first element in a temporary variable.
2. Shift all other elements one position to the left.
3. Set the last element of the array to the value stored in the temporary variable.

**Right Rotation** To perform a right rotation on an array, follow these steps:

1. Store the last element in a temporary variable.
2. Shift all other elements one position to the right.
3. Set the first element of the array to the value stored in the temporary variable.

```
void LeftRotation_by_one(vector<int> & nums, int k, string direction){
    int len = nums.size()-1;

    if(direction == "left"){
        int temp = arr[0]
        for (int i=1 ; i<len; i++){
            arr[i-1] =arr[i];
        }
        arr[size-1] =temp;
    }
    else{
        int temp = arr[len]
        for (int i=len ; i>=0; i--){
            arr[i] =arr[i-1];
        }
        arr[0] =temp;
    }
}
```

## Reversal Algorithm Technique

### Introduction

The reversal algorithm is a simple and efficient technique used to reverse the elements of an array or a list. It is a fundamental operation in data manipulation and can be applied to various scenarios, including reversing a string, array, or linked list.

## Algorithm Overview

The reversal algorithm for array rotation involves two main steps:

1. Divide the array into two subarrays, where the division point represents the rotation point.
2. Reverse both subarrays.
3. Reverse the entire array to obtain the desired rotated array.

### Implementation

To perform rotation using reversal algorithm, follow these steps:

1. Divide the array into two subarrays at the rotation point.
2. Reverse both subarrays.
3. Reverse the entire array.

```
void reverse(vector<int>& nums, int low, int high){
    while(low<high){
        int temp = nums[low];
        nums[low] = nums[high];
        nums[high] = temp;
        low++;
        high--;
    }
}


void LeftRotation_by_one(vector<int> & nums, int k, string direction){
    int len = nums.size()-1;
    if (len <= 1 || k== 0) {
        return;
    }

    k = k % len;

    if (k == 0) {
        return;
    }

    if(direction == "left"){
        reverse(nums, 0, k-1);
        reverse(nums,k,len);
        reverse(nums,0,len);
    }
    else{
        reverse(nums, len-k+1, len);
        reverse(nums,0,len-k);
        reverse(nums,0,len);
    }
}
```

## Juggling Algorithm

### Introduction

- The Juggling Algorithm is an efficient method for rotating the elements of an array.
- It is particularly useful when you need to perform left or right rotations on an array by a fixed number of positions.
- This algorithm is known for its simplicity and speed in achieving array rotations.

## Algorithm Overview

The Juggling Algorithm for array rotation is based on the concept of GCD (Greatest Common Divisor) / HCF ( Highest Common Factor ). It involves three main steps:

1. Calculate the GCD of the array size `n` and the number of positions to be

rotated `d`.
2. Divide the array into `gcd(n, d)` sets, where each set contains elements that need to be cycled within itself.
3. Perform cyclic swaps within each set, moving the elements to their new positions.

## Implementation

To perform Juggling Algorithm, follow these steps:

1. Calculate the GCD of `n` and `d` (denoted as `gcd`).
2. Divide the array into `gcd` sets.
3. Iterate over each set and perform cyclic swaps within the set.

```c
int findGCD(int a, int b) {
    if (b == 0) {
        return a;
    }
    return findGCD(b, a % b);
}

void jugglingRotate(int arr[], int n, int d) {

    d = d % n;  // Ensures that d is within the given array size.

    int gcd = findGCD(n, d); // Calculates gcd for size by rotation.

    for (int i = 0; i < gcd; i++) {
        // Stores the first element to temproary varaible.
        int temp = arr[i];
        int j = i;
        // Looping to do swap elemnts from every block or we can say swap element after ever
        while (1) {
            int k = (j + d) % n;
            if (k == i) {
                break;
            }
            arr[j] = arr[k];
            j = k;
        }
        // First element again gets stored at last swapped position.
        arr[j] = temp;
    }
}
```

### Lets Visualize juggling

- Suppose we have an array of size 6 and we have top perform 3 rotation ( n = 6 , k= 3).

# Topic - Rotation Algorithms / Techniques

Rotation is some of the basic operations that we perform on array, There are multiple ways to perform rotation on array.

- One by One Rotation
- Reversal Algorithm
- Juggling Algotihm

## One By One Rotation Technique

### Introduction

The method for rotating the elements of an array in a circular fashion. It shifts the elements of the array one position to the left or right, depending on the direction of rotation.

### Rotation Direction

The rotation can be performed in two directions:

- **Left Rotation:** In this mode, the elements are shifted to the left. The last element of the array becomes the first, and all other elements are moved one position to the left.
- **Right Rotation:** In this mode, the elements are shifted to the right. The first element of the array becomes the last, and all other elements are moved one position to the right.

### Implementation

**Left Rotation** To perform a left rotation on an array, follow these steps:

1. Store the first element in a temporary variable.
2. Shift all other elements one position to the left.
3. Set the last element of the array to the value stored in the temporary variable.

**Right Rotation** To perform a right rotation on an array, follow these steps:

1. Store the last element in a temporary variable.
2. Shift all other elements one position to the right.
3. Set the first element of the array to the value stored in the temporary variable.

```
void LeftRotation_by_one(vector<int> & nums, int k, string direction){
    int len = nums.size()-1;

    if(direction == "left"){
        int temp = arr[0]
        for (int i=1 ; i<len; i++){
            arr[i-1] =arr[i];
        }
        arr[size-1] =temp;
    }
    else{
        int temp = arr[len]
        for (int i=len ; i>=0; i--){
            arr[i] =arr[i-1];
        }
        arr[0] =temp;
    }
}
```

## Reversal Algorithm Technique

### Introduction

The reversal algorithm is a simple and efficient technique used to reverse the elements of an array or a list. It is a fundamental operation in data manipulation and can be applied to various scenarios, including reversing a string, array, or linked list.

## Algorithm Overview

The reversal algorithm for array rotation involves two main steps:

1. Divide the array into two subarrays, where the division point represents the rotation point.
2. Reverse both subarrays.
3. Reverse the entire array to obtain the desired rotated array.

### Implementation

To perform rotation using reversal algorithm, follow these steps:

1. Divide the array into two subarrays at the rotation point.
2. Reverse both subarrays.
3. Reverse the entire array.

```
void reverse(vector<int>& nums, int low, int high){
    while(low<high){
        int temp = nums[low];
        nums[low] = nums[high];
        nums[high] = temp;
        low++;
        high--;
    }
}


void LeftRotation_by_one(vector<int> & nums, int k, string direction){
    int len = nums.size()-1;
    if (len <= 1 || k== 0) {
        return;
    }

    k = k % len;

    if (k == 0) {
        return;
    }

    if(direction == "left"){
        reverse(nums, 0, k-1);
        reverse(nums,k,len);
        reverse(nums,0,len);
    }
    else{
        reverse(nums, len-k+1, len);
        reverse(nums,0,len-k);
        reverse(nums,0,len);
    }
}
```

## Juggling Algorithm

### Introduction

- The Juggling Algorithm is an efficient method for rotating the elements of an array.
- It is particularly useful when you need to perform left or right rotations on an array by a fixed number of positions.
- This algorithm is known for its simplicity and speed in achieving array rotations.

## Algorithm Overview

The Juggling Algorithm for array rotation is based on the concept of GCD (Greatest Common Divisor) / HCF ( Highest Common Factor ). It involves three main steps:

1. Calculate the GCD of the array size `n` and the number of positions to be rotated `d`.
2. Divide the array into `gcd(n, d)` sets, where each set contains elements that need to be cycled within itself.
3. Perform cyclic swaps within each set, moving the elements to their new positions.

## Implementation

To perform Juggling Algorithm, follow these steps:

1. Calculate the GCD of `n` and `d` (denoted as `gcd`).
2. Divide the array into `gcd` sets.
3. Iterate over each set and perform cyclic swaps within the set.

```
int findGCD(int a, int b) {
    if (b == 0) {
        return a;
    }
    return findGCD(b, a % b);
}

void jugglingRotate(int arr[], int n, int d) {

    d = d % n;  // Ensures that d is within the given array size.

    int gcd = findGCD(n, d); // Calculates gcd for size by rotation.

    for (int i = 0; i < gcd; i++) {
        // Stores the first element to temproary varaible.
        int temp = arr[i];
        int j = i;
        // Looping to do swap elemnts from every block or we can say swap element after eve
        while (1) {
            int k = (j + d) % n;
            if (k == i) {
                break;
            }
            arr[j] = arr[k];
            j = k;
        }
        // First element again gets stored at last swapped position.
        arr[j] = temp;
    }
}
```

**Lets Visualize juggling**

- Suppose we have an array of size 6 and we have top perform 3 rotation ( n = 6 , k= 3).

| Index | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Array | 1 | 2 | 3 | 4 | 5 | 6 |

`size / n` = **6** and `rotation/k` = **3**, Therfore `gcd(3,6)` = **3**.

- Sets based on gcd are : `1 | 2` and `3 | 4` and `5 | 6`.

# Topic - Rotation Algorithms / Techniques

Rotation is some of the basic operations that we perform on array, There are multiple ways to perform rotation on array.

- One by One Rotation
- Reversal Algorithm
- Juggling Algotihm

## One By One Rotation Technique

### Introduction

The method for rotating the elements of an array in a circular fashion. It shifts the elements of the array one position to the left or right, depending on the direction of rotation.

### Rotation Direction

The rotation can be performed in two directions:

- **Left Rotation:** In this mode, the elements are shifted to the left. The last element of the array becomes the first, and all other elements are moved one position to the left.
- **Right Rotation:** In this mode, the elements are shifted to the right. The first element of the array becomes the last, and all other elements are moved one position to the right.

### Implementation

**Left Rotation** To perform a left rotation on an array, follow these steps:

1. Store the first element in a temporary variable.
2. Shift all other elements one position to the left.
3. Set the last element of the array to the value stored in the temporary variable.

**Right Rotation** To perform a right rotation on an array, follow these steps:

1. Store the last element in a temporary variable.
2. Shift all other elements one position to the right.
3. Set the first element of the array to the value stored in the temporary variable.

```
void LeftRotation_by_one(vector<int> & nums, int k, string direction){
    int len = nums.size()-1;

    if(direction == "left"){
        int temp = arr[0]
        for (int i=1 ; i<len; i++){
            arr[i-1] =arr[i];
        }
        arr[size-1] =temp;
    }
    else{
        int temp = arr[len]
        for (int i=len ; i>=0; i--){
            arr[i] =arr[i-1];
        }
        arr[0] =temp;
    }
}
```

## Reversal Algorithm Technique

### Introduction

The reversal algorithm is a simple and efficient technique used to reverse the elements of an array or a list. It is a fundamental operation in data manipulation and can be applied to various scenarios, including reversing a string, array, or linked list.

## Algorithm Overview

The reversal algorithm for array rotation involves two main steps:

1. Divide the array into two subarrays, where the division point represents the rotation point.
2. Reverse both subarrays.
3. Reverse the entire array to obtain the desired rotated array.

### Implementation

To perform rotation using reversal algorithm, follow these steps:

1. Divide the array into two subarrays at the rotation point.
2. Reverse both subarrays.
3. Reverse the entire array.

```
void reverse(vector<int>& nums, int low, int high){
    while(low<high){
        int temp = nums[low];
        nums[low] = nums[high];
        nums[high] = temp;
        low++;
        high--;
    }
}


void LeftRotation_by_one(vector<int> & nums, int k, string direction){
    int len = nums.size()-1;
    if (len <= 1 || k== 0) {
        return;
    }

    k = k % len;

    if (k == 0) {
        return;
    }

    if(direction == "left"){
        reverse(nums, 0, k-1);
        reverse(nums,k,len);
        reverse(nums,0,len);
    }
    else{
        reverse(nums, len-k+1, len);
        reverse(nums,0,len-k);
        reverse(nums,0,len);
    }
}
```

## Juggling Algorithm

### Introduction

- The Juggling Algorithm is an efficient method for rotating the elements of an array.
- It is particularly useful when you need to perform left or right rotations on an array by a fixed number of positions.
- This algorithm is known for its simplicity and speed in achieving array rotations.

## Algorithm Overview

The Juggling Algorithm for array rotation is based on the concept of GCD (Greatest Common Divisor) / HCF ( Highest Common Factor ). It involves three main steps:

1. Calculate the GCD of the array size `n` and the number of positions to be rotated `d`.
2. Divide the array into `gcd(n, d)` sets, where each set contains elements that need to be cycled within itself.
3. Perform cyclic swaps within each set, moving the elements to their new positions.

## Implementation

To perform Juggling Algorithm, follow these steps:

1. Calculate the GCD of `n` and `d` (denoted as `gcd`).
2. Divide the array into `gcd` sets.
3. Iterate over each set and perform cyclic swaps within the set.

```
int findGCD(int a, int b) {
    if (b == 0) {
        return a;
    }
    return findGCD(b, a % b);
}

void jugglingRotate(int arr[], int n, int d) {

    d = d % n;  // Ensures that d is within the given array size.

    int gcd = findGCD(n, d); // Calculates gcd for size by rotation.
    for (int i = 0; i < gcd; i++) {
        // Stores the first element to temproary varaible.
        int temp = arr[i];
        int j = i;
        // Looping to do swap elemnts from every block or we can say swap element after ever
        while (1) {
            int k = (j + d) % n;
            if (k == i) {
                break;
            }
            arr[j] = arr[k];
            j = k;
        }
        // First element again gets stored at last swapped position.
        arr[j] = temp;
    }
}
```

### Lets Visualize juggling

- Suppose we have an array of size 6 and we have top perform 3 rotation ( n = 6 , k= 3).

# Topic - Rotation Algorithms / Techniques

Rotation is some of the basic operations that we perform on array, There are multiple ways to perform rotation on array.

- One by One Rotation
- Reversal Algorithm
- Juggling Algotihm

## One By One Rotation Technique

### Introduction

The method for rotating the elements of an array in a circular fashion. It shifts the elements of the array one position to the left or right, depending on the direction of rotation.

### Rotation Direction

The rotation can be performed in two directions:

- **Left Rotation:** In this mode, the elements are shifted to the left. The last element of the array becomes the first, and all other elements are moved one position to the left.

- **Right Rotation:** In this mode, the elements are shifted to the right. The first element of the array becomes the last, and all other elements are moved one position to the right.

### Implementation

**Left Rotation** To perform a left rotation on an array, follow these steps:

1. Store the first element in a temporary variable.
2. Shift all other elements one position to the left.
3. Set the last element of the array to the value stored in the temporary variable.

**Right Rotation** To perform a right rotation on an array, follow these steps:

1. Store the last element in a temporary variable.
2. Shift all other elements one position to the right.
3. Set the first element of the array to the value stored in the temporary variable.

```
void LeftRotation_by_one(vector<int> & nums, int k, string direction){
    int len = nums.size()-1;

    if(direction == "left"){
        int temp = arr[0]
        for (int i=1 ; i<len; i++){
            arr[i-1] =arr[i];
        }
        arr[size-1] =temp;
    }
    else{
        int temp = arr[len]
        for (int i=len ; i>=0; i--){
            arr[i] =arr[i-1];
        }
        arr[0] =temp;
    }
}
```

## Reversal Algorithm Technique

### Introduction

The reversal algorithm is a simple and efficient technique used to reverse the elements of an array or a list. It is a fundamental operation in data manipulation and can be applied to various scenarios, including reversing a string, array, or linked list.

## Algorithm Overview

The reversal algorithm for array rotation involves two main steps:

1. Divide the array into two subarrays, where the division point represents the rotation point.
2. Reverse both subarrays.
3. Reverse the entire array to obtain the desired rotated array.

### Implementation

To perform rotation using reversal algorithm, follow these steps:

1. Divide the array into two subarrays at the rotation point.
2. Reverse both subarrays.
3. Reverse the entire array.

```
void reverse(vector<int>& nums, int low, int high){
    while(low<high){
        int temp = nums[low];
        nums[low] = nums[high];
        nums[high] = temp;
        low++;
        high--;
    }
}


void LeftRotation_by_one(vector<int> & nums, int k, string direction){
    int len = nums.size()-1;
    if (len <= 1 || k== 0) {
        return;
    }

    k = k % len;

    if (k == 0) {
        return;
    }

    if(direction == "left"){
        reverse(nums, 0, k-1);
        reverse(nums,k,len);
        reverse(nums,0,len);
    }
    else{
        reverse(nums, len-k+1, len);
        reverse(nums,0,len-k);
        reverse(nums,0,len);
    }
}
```

## Juggling Algorithm

### Introduction

- The Juggling Algorithm is an efficient method for rotating the elements of an array.
- It is particularly useful when you need to perform left or right rotations on an array by a fixed number of positions.
- This algorithm is known for its simplicity and speed in achieving array rotations.

## Algorithm Overview

The Juggling Algorithm for array rotation is based on the concept of GCD (Greatest Common Divisor) / HCF ( Highest Common Factor ). It involves three main steps:

1. Calculate the GCD of the array size `n` and the number of positions to be rotated `d`.
2. Divide the array into `gcd(n, d)` sets, where each set contains elements that need to be cycled within itself.
3. Perform cyclic swaps within each set, moving the elements to their new positions.

## Implementation

To perform Juggling Algorithm, follow these steps:

1. Calculate the GCD of `n` and `d` (denoted as `gcd`).
2. Divide the array into `gcd` sets.
3. Iterate over each set and perform cyclic swaps within the set.

```
int findGCD(int a, int b) {
    if (b == 0) {
        return a;
    }
    return findGCD(b, a % b);
}

void jugglingRotate(int arr[], int n, int d) {

    d = d % n;  // Ensures that d is within the given array size.
    int gcd = findGCD(n, d); // Calculates gcd for size by rotation.

    for (int i = 0; i < gcd; i++) {
        // Stores the first element to temproary varaible.
        int temp = arr[i];
        int j = i;
        // Looping to do swap elemnts from every block or we can say swap element after ever
        while (1) {
            int k = (j + d) % n;
            if (k == i) {
                break;
            }
            arr[j] = arr[k];
            j = k;
        }
        // First element again gets stored at last swapped position.
        arr[j] = temp;
    }
}
```

### Lets Visualize juggling

- Suppose we have an array of size 6 and we have top perform 3 rotation ( n = 6 , k= 3).

| Index | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Array | 1 | 2 | 3 | 4 | 5 | 6 |

`size / n` = **6** and `rotation/k` = **3**, Therfore `gcd(3,6)` = **3**.

- Sets based on gcd are : `1 | 2` and `3 | 4` and `5 | 6`.
- Perform a leftward cyclic rotation on the initial elements within each set .
- Similarly, subsequently iterate through the remaining elements applying cyclic rotations until the entire set is cycled.
- Finally we got

### FINALLY WE LEARNT HOW TO JUGGLE ARRAYS

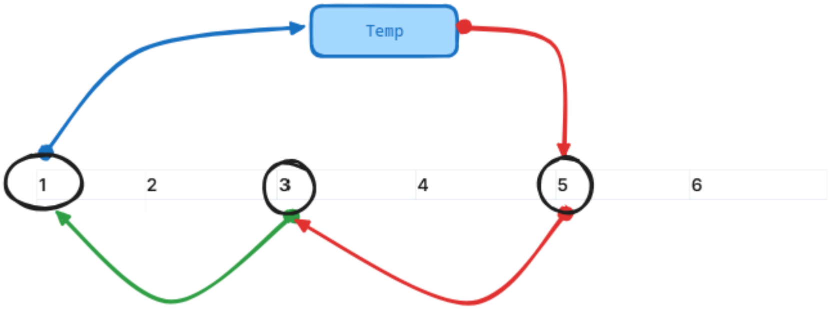


Figure 1: image

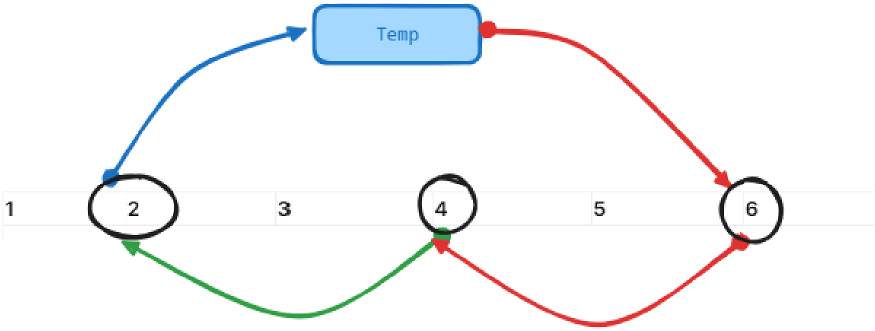
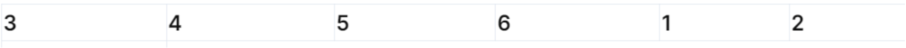


Figure 2: image



| | | | | | |
|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | 1 | 2 |

Figure 3: image

SUPER SIMPLE