```
                count = 1;
            }
            else if(element == A[i])
            {
                // Increment counter If the counter is not 0 and
                // element is same as current candidate.
                count++;
            }
            else
            {
                // Decrement counter If the counter is not 0 and
                // element is different from current candidate.
                count--;
            }
    }
    return element;
}
```

Time Complexity: $O(n)$.
Space Complexity: $O(1)$.

Problem-47 Given an array of $2n$ elements of which n elements are same and the remaining n elements are all different. Find the majority element.

Solution: The repeated elements will occupy half the array. No matter what arrangement it is, only one of the below will be true,

- All duplicate elements will be at a relative distance of 2 from each other. Ex: n, 1, n, 100, n, 54, n ...
- At least two duplicate elements will be next to each other
  Ex: $n, n, 1, 100, n, 54, n, \ldots.$
  $n, 1, n, n, n, 54, 100 \ldots$
  $1, 100, 54, n, n, n, n \ldots.$

So, in worst case, we need will two passes over the array,

First Pass: compare $A[i]$ and $A[i+1]$
Second Pass: compare $A[i]$ and $A[i+2]$

Something will match and that's your element.

This will cost $O(n)$ in time and $O(1)$ in space.

Problem-48 Given an array with $2n + 1$ integer elements, n elements appear twice in arbitrary places in the array and a single integer appears only once somewhere inside. Find the lonely integer with $O(n)$ operations and $O(1)$ extra memory.

Solution: Since except one element all other elements are repeated. We know that $A\ XOR\ A\ = 0$. Based on this if we XOR all the input elements then we get the remaining element.

```
int solution(int* A)
{
        int i, res;
        for (i = res = 0; i < 2n+1; i++)
                res = res ^ A[i];
         return res;
}
```

Time Complexity: $O(n)$.
Space Complexity: $O(1)$.

Problem-49 Throwing eggs from an n-story building.

Suppose that we have an n story building and a set of eggs. Also assume that an egg breaks if it is thrown off floor F or higher, and will not break otherwise. Devise a strategy to determine the floor F, while breaking $O(logn)$ eggs.

Solution: Refer *Divide and Conquer* chapter.

Problem-50 Local minimum of an array.

Given an array A of n distinct integers, design an $O(logn)$ algorithm to find a *local minimum*: an index i such that $A[i-1] < A[i] < A[i+1]$.

Solution: Check the middle value $A[n/2]$, and two neighbors $A[n/2 - 1]$ and $A[n/2 + 1]$. If $A[n/2]$ is local minimum, stop; otherwise search in half with smaller neighbor.

Problem-51 Give an $n \times n$ array of elements such that each row is in ascending order and each column is in ascending order, devise an $O(n)$ algorithm to determine if a given element x in the array. You may assume all elements in the $n \times n$ array are distinct.

Solution: Let us assume that the given matrix is $A[n][n]$. Start with the last row, first column [or first row - last column]. If the element we are searching for is greater than the element at $A[1][n]$, then the column 1 can be eliminated. If the search element is less than the element at $A[1][n]$, then the last row can be completely eliminated. Now, once the first column or the last row is eliminated, now, start over the process again with left-bottom end of the remaining array. In this algorithm, there would be maximum n elements that the search element would be compared with.

Time Complexity: $O(n)$. This is because we will traverse at most 2 n points.

Space Complexity: $O(1)$.

Problem-52 Given an $n \times n$ array a of n^2 numbers, Give an $O(n)$ algorithm to find a pair of indices i and j such that $A[i][j] < A[i+1][j], A[i][j] < A[i][j+1], A[i][j] < A[i-1][j]$, and $A[i][j] < A[i][j-1]$.

Solution: This problem is same as Problem-51.

Problem-53 Given $n \times n$ matrix, and in each row all 1's are followed 0's. Find row with maximum number of 0's.

Solution: Start with first row, last column. If the element is 0 then move to the previous column in the same row and at the same time increase the counter to indicate the maximum number of 0's. If the element is 1 then move to the next row in the same column. Repeat this process until we reach last row, first column.

Time Complexity: $O(2n) \approx O(n)$ (very much similar to Problem-51).

Problem-54 Given an input array of size unknown with all numbers in the beginning and special symbols in the end. Find the index in the array from where special symbols start.

Solution: Refer *Divide and Conquer* chapter.

Problem-55 Finding the Missing Number

We are given a list of $n-1$ integers and these integers are in the range of 1 to n. There are no duplicates in list. One of the integers is missing in the list. Given an algorithm to find the missing integer.

Example:

I/P $[1, 2, 4, 6, 3, 7, 8]$

O/P 5

Solution: Use sum formula

1) Get the sum of numbers, $sum = n * (n+1)/2$
2) Subtract all the numbers from sum and you will get the missing number.

Time Complexity: $O(n)$, this is because we need to scan the complete array.

Problem-56 In Problem-55, if the sum of the numbers goes beyond maximum allowed integer, then there can be integer overflow and we may not get correct answer. Can we solve this problem?

Solution:

1) XOR all the array elements, let the result of XOR be X.
2) XOR all numbers from 1 to n, let XOR be Y.
3) XOR of X and Y gives the missing number.

```
int FindMissingNumber(int A[], int n)
{
        int i, X, Y;
        for (i = 0; i < 9; i ++)
                X ^= A[i];
        for (i = 1; i <= 10; i ++)
                Y ^= i;
        //In fact, one variable is enough.
        return X ^ Y;
}
```

Time Complexity: $O(n)$, this is because we need to scan the complete array.

Problem-57 Find the Number Occurring Odd Number of Times

Given an array of positive integers, all numbers occurs even number of times except one number which occurs odd number of times. Find the number in $O(n)$ time & constant space.

Example:

I/P = $[1, 2, 3, 2, 3, 1, 3]$

O/P = 3

Solution: Do a bitwise XOR of all the elements. Finally we get the number which has odd occurrences. This is because of the fact that, $A\ XOR\ A = 0$.

Time Complexity: $O(n)$.

Problem-58 Find the two repeating elements in a given array

Given an array with $n + 2$ elements, all elements of the array are in range 1 to n and also all elements occur only once except two numbers which occur twice. Find those two repeating numbers.

Example: $6, 2, 6, 5, 2, 3, 1$ and $n = 5$

The above input has $n + 2 = 7$ elements with all elements occurring once except 2 and 6 which occur twice. So the output should be $6\ 2$.

Solution: One simple way to scan the complete array for each element of the input elements. That means use two loops. In the outer loop, select elements one by one and count the number of occurrences of the selected element in the inner loop.

```
void PrintRepeatedElements(int A[], int n)
{
        int i, j;
        for(i = 0; i < n; i++)
                for(j = i+1; j <n; j++)
                        if(A[i] == A[j])
                                printf("%d", A[i]);
}
```

Time Complexity: $O(n^2)$.
Space Complexity: $O(1)$.

Problem-59 For the Problem-58, can we improve the time complexity?

Solution: Sort the array using any comparison sorting algorithm and see if there are any elements which contiguous with same value.

Time Complexity: $O(nlogn)$.
Space Complexity: $O(1)$.

Problem-60 For the Problem-58, can we improve the time complexity?

Solution: Use Count Array. This solution is like using a hash table. But for simplicity we can use array for storing the counts. 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 we see an element whose count is already set, print it as duplicate.

```
void PrintRepeatedElements(int A[], int n)
{
        int *count = (int *)calloc(sizeof(int), (n - 2));
        for(int i = 0; i < size; i++)
        {
                if(count[A[i]] == 1)
                        printf("%d", A[i]);
                else
                        count[A[i]]++;
        }
}
```

Time Complexity: $O(n)$.
Space Complexity: $O(n)$.

Problem-61 Consider the Problem-58. Let us assume that the numbers are in the range 1 to n. Is there any other way of solving the problem?

Solution: Using XOR Operation. Let the repeating numbers be X and Y, if we xor all the elements in the array and all integers from 1 to n, then the result is $X\ XOR\ Y$.

The 1's in binary representation of $X\ XOR\ Y$ is corresponding to the different bits between X and Y. Suppose that the k^{th} bit of $X\ XOR\ Y$ is 1, we can XOR all the elements in the array and all integers from 1 to n, whose k^{th} bits are 1. The result will be one of X and Y.

```
void PrintRepeatedElements (int A[], int size)
{
        int XOR = A[0];
        int right_most_set_bit_no;
        int n = size - 2;
        int X= 0, Y = 0;

        /* Compute XOR of all elements in A[]*/
        for(int i = 0; i < n; i++)
                XOR ^= A[i];

        /* Compute XOR of all elements {1, 2 ..n} */
        for(i = 1; i <= n; i++)
                XOR ^= i;

         /* Get the rightmost set bit in right_most_set_bit_no */
         right_most_set_bit_no = XOR & ~( XOR -1);

        /* Now divide elements in two sets by comparing rightmost set */
        for(i = 0; i < n; i++)
        {
                if(A[i] & right_most_set_bit_no)
                        X = X^ A[i]; /*XOR of first set in A[] */
                else
                        Y = Y ^ A[i]; /*XOR of second set inA[] */
        }
        for(i = 1; i <= n; i++)
        {
                if(i & right_most_set_bit_no)
                        X = X ^ i; /*XOR of first set in A[] and {1, 2, ...n }*/
                else
                        Y = Y ^ i; /*XOR of second set in A[] and {1, 2, ...n } */
        }

        printf("%d and %d",X, Y);
}
```

Time Complexity: $O(n)$.
Space Complexity: $O(1)$.

Problem-62 Consider the Problem-58. Let us assume that the numbers are in the range 1 to n. Is there yet other way of solving the problem?

Solution: We can solve this by creating two simple mathematical equations. Let us assume that two numbers which we are going to find are X and Y. We know the sum of n numbers is $n(n+1)/2$ and product is $n!$. Make two equations using these sum and product formulae, and get values of two unknowns using the two equations.

Let summation of all numbers in array be S and product be P and the numbers which are being repeated are X and Y.

$X + Y = S - n(n+1)/2$
$XY = P/n!$

Using above two equations, we can find out X and Y.
There can be addition and multiplication overflow problem with this approach.

Time Complexity: $O(n)$.
Space Complexity: $O(1)$.

Problem-63 Similar to Problem-58. Let us assume that the numbers are in the range 1 to n. Also, $n-2$ elements are repeating thrice and remaining two elements are repeating twice. Find the element which is repeating twice.

Solution: If we xor all the elements in the array and all integers from 1 to n, then the all the elements which are trice will become zero This is because, since the element is repeating trice and XOR with another time from range makes that element appearing four times. As a result, output of a $XOR\ a\ XOR\ a\ XOR\ a = 0$. Same is case with all elements which repeated thrice.

With the same logic, for the element which repeated twice, if we XOR the input elements and also the range, then the total number of appearances for that element is 3. As a result, output of $a\ XOR\ a\ XOR\ a = a$. Finally, we get the element which repeated twice.

Time Complexity: $O(n)$.
Space Complexity: $O(1)$.

Problem-64 **Separate Even and Odd numbers**

Given an array $A[\,]$, write a function that segregates even and odd numbers. The functions should put all even numbers first, and then odd numbers.

Example:
Input = {12, 34, 45, 9, 8, 90, 3}
Output = {12, 34, 90, 8, 9, 45, 3}

In the output, order of numbers can be changed, i.e., in the above example 34 can come before 12 and 3 can come before 9.

Solution: The problem is very similar to *Separate 0's and 1's (*Problem-65*)* in an array, and both of these problems are variation of famous *Dutch national flag problem*.

Algorithm: Logic is little similar to Quick sort.

1) Initialize two index variables left and right: $left = 0$, $right = n - 1$
2) Keep incrementing left index until we see an odd number.
3) Keep decrementing right index until we see an even number.
4) If $left < right$ then swap $A[left]$ and $A[right]$

Implementation:

```
void DutchNationalFlag(int A[], int n)
{
        /* Initialize left and right indexes */
        int left = 0, right = n-1;
        while(left < right)
        {
                /* Increment left index while we see 0 at left */
                while(A[left]%2 == 0 && left < right)
                        left++;

                /* Decrement right index while we see 1 at right */
                while(A[right]%2 == 1 && left < right)
                        right--;

                if(left < right)
                {
                        /* Swap A[left] and A[right]*/
                        swap(&A[left], &A[right]);
                        left++;
                        right--;
                }
        }
}
```

Time Complexity: $O(n)$.

Problem-65 Other way of asking Problem-64 but with little difference.

Separate 0's and 1's in an array

We are given an array of 0's and 1's in random order. Separate 0's on left side and 1's on right side of the array. Traverse array only once.

Input array = [0, 1, 0, 1, 0, 0, 1, 1, 1, 0]
Output array = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]

Solution: Counting 0's or 1's

1. Count the number of $0's$. Let count be C.
2. Once we have count, we can put C 0's at the beginning and $1's$ at the remaining $n - C$ positions in array.

Time Complexity: $O(n)$. This solution scans the array two times.

Problem-66 Can we solve the Problem-65 in once scan?

Solution: Yes. Use two indexes to traverse:

Maintain two indexes. Initialize first index left as 0 and second index right as $n - 1$.

Do following while $left < right$:

1) Keep incrementing index left while there are 0s at it
2) Keep decrementing index right while there are 1s at it
3) If left < right then exchange $A[left]$ and $A[right]$

```
/*Function to put all 0s on left and all 1s on right*/
void Separate0and1(int A[], int n)
{
        /* Initialize left and right indexes */
        int left = 0, right = n-1;
      while(left < right)
      {
            /* Increment left index while we see 0 at left */
            while(A[left] == 0 && left < right)
                  left++;

            /* Decrement right index while we see 1 at right */
            while(A[right] == 1 && left < right)
                  right–;

            /* If left is smaller than right then there is a 1 at left
            and a 0 at right.  Swap A[left] and A[right]*/
            if(left < right)
            {
```

```
                    A[left] = 0;
                    A[right] = 1;
                    left++;
                    right–;
                }
            }
}
```

Time Complexity: $O(n)$.
Space Complexity: $O(1)$.

Problem-67 Maximum difference between two elements

Given an array $A[]$ of integers, find out the difference between any two elements such that larger element appears after the smaller number in $A[]$.

Examples: If array is $[2, 3, 10, 6, 4, 8, 1]$ then returned value should be 8 (Diff between 10 and 2). If array is $[7, 9, 5, 6, 3, 2]$ then returned value should be 2 (Difference between 7 and 9)

Solution: Refer *Divide and Conquer* chapter.

Problem-68 Given an array of 101 elements. Out of them 25 elements are repeated twice, 12 elements are repeated 4 times and one element is repeated 3 times. Find the element which repeated 3 times in $O(1)$.

Solution: Before solving this problem let us consider the following XOR operation property.

$$a\ XOR\ a = 0$$

That means, if we apply the XOR on same elements then the result is 0. Let us apply this logic for this problem.

Algorithm:

- XOR all the elements of the given array and assume the result is A.
- After this operation, 2 occurrences of number which appeared 3 times becomes 0 and one occurrence will remain.
- The 12 elements which are appearing 4 times become 0.
- The 25 elements which are appearing 2 times become 0.

So just $XOR'ing$ all the elements give the result.

Time Complexity: $O(n)$, because we are doing only once scan.
Space Complexity: $O(1)$.

Problem-69 Given an array A of n numbers. Find all pairs of X and Y in the array such that $K = X * Y$. Give an efficient algorithm without sorting.

Solution: Create a hash table from the numbers that divide K. Divide K by a number and check for quotient in the table.

Problem-70 Given a number n, give an algorithm for finding the number of trailing zeros in $n!$.

Solution:

```
int NumberOfTrailingZerosInNumber(int n)
{
        int i, count = 0;
        if (n < 0)
                return -1;
        for (i = 5; n / i > 0; i *= 5)
                count += n / i;
        return count;
}
```

Time Complexity: $O(logn)$,