Searching

Unordered Linear search :

```
int unorderedLinearSearch(vector<int>& arr, int x) {
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == x) return i;
    }
    return -1;
}
```

Ordered/sorted linear search :

```
int orderedLinearSearch(vector<int>& arr, int x) {
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i] == x) return i;
        if (arr[i] > x) break;
    }
    return -1;
}
```

Binary search :

```
int binarySearch(vector<int>& arr, int x) {
    int low = 0, high = arr.size() - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == x) return mid;
        if (arr[mid] < x) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}
```

Interpolation search :

```
int interpolationSearch(vector<int>& arr, int x) {
    int low = 0, high = arr.size() - 1;
    while (low <= high && x >= arr[low] && x <= arr[high]) {
        int pos = low + ((x - arr[low]) * (high - low) / (arr[high] -
arr[low]));
        if (arr[pos] == x) return pos;
        if (arr[pos] < x) low = pos + 1;
        else high = pos - 1;
    }
    return -1;
}
```

Problems

Probem 1 & 2 & 4 :

Given an array of n numbers, give an algorithm for checking whether there are any duplicate elements in the array or no?

```
void checkDuplicates(int A[], int n) {
    for(int i = 0; i < n; i++) {
        for(int j = i+1; j < n; j++) {
            if(A[i] == A[j]) {
                cout << "Duplicates exist";
                return;
            }
        }
    }
    cout << "No duplicates";
}
```

Improved :

```
void checkDuplicates(int A[], int n) {
    sort(A, n);
    for(int i = 0; i < n-1; i++) {
            if(A[i] == A[j]) {
                cout << "Duplicates exist";
                return;
            }
    }
    cout << "No duplicates";
}
```

Improved :

```
void checkDuplicates(int A[], int n) {
    for(int i = 0; i < n; i++) {
            if(A[abs(a[i])] < 0) {
                cout << "Duplicates exist";
                return;
            } else {
                A[A[i]] =- A[A[i]];
            }
    }
    cout << "No duplicates";
}
```

Problem 5 & 8 :

Given an array of n numbers. Give an algorithm for finding the element which appears the maximum number of times in the array?

```
int maxRepetitions(int A[], int n) {
    int counter = 0, max = 0;
    for(int i = 0; i < n; i++) {
        counter = 0;
        for(int j = 0 j < n; j++) {
            if(A[i] ==A[j]) {
                counter++;
            }
        }
        if(counter > max) {
            max = counter;
        }
    }
    return max;
}
```

Improved :

```
int maxRepetitions(int A[], int n) {
    int i = 0; max = 0; maxIndex;
    for(i = 0; i < n; i++) {
        A[A[i]%n] += n;
    }
    for(i = 0; i < n; i++) {
        if(A[i]/n > max) {
            max = A[i]/n;
            maxIndex = i;
        }
    }
    return maxIndex;
}
```

Problem 13 & 17 :

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 the 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

```
int findMissingNumber(int A[], int n) {
    int i, j, found = 0;
    for(i = 1; i <= n; i++) {
        found = 0;
        for(j = 0; j < n; j++) {
            if(A[j] == i) {
                found = 1;
            }
        }
        if(!found) {
            return i;
        }
    }
    return -1;
}
```

Using xor :

```
int findMissingNumber(int A[], int n) {
    int i, x, y;
    for(i = 0; i < n; i++) {
        x ^= A[i];
    }
    for(i = 1; i <= n; i++) {
        y ^= i;
    }
    return x ^ y;
}
```

Problem 19 & 21 & 22:

Find the two repeating elements in a given array: Given an array with size, 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. For example: if the array is 4,2,4,5,2,3,1 with size = 7 and n = 5. This input has n + 2 = 7 elements with all elements occurring once except 2 and 4 which occur twice. So the output should be 4 2.

```
void repeatedElements(int A[], int n) {
    for(int i = 0; i < n; i++) {
        for(int j = i+1; j < n; j++) {
            if(A[i] == A[j]) {
                cout << A[i];
            }
        }
    }
}
```

Improved TC :

```
void repeatedElements(int A[], int n) {
    int xor = A[0];
    int i, right_most_set_bit_no , x = 0, y=0;
    for(i = 0; i < n; i++) {
        xor ^= A[i];
    }
    for(i = 1; i <= n; i++) {
        xor ^= i;
    }
    right_most_set_bit_no = xor & ~(xor-1);
    for(i = 0; i < n; i++) {
        if(A[i] & right_most_set_bit_no) {
            x =x ^ A[i];
        } else {
            y = y ^ A[i];
        }
    }
    for(i = 1; i <= n; i++) {
        if(i & right_most_set_bit_no) {
            x =x ^ A[i];
        } else {
            y = y ^ A[i];
        }
    }
    cout << x << y;
}
```

Problem 25 & 26 :

Given an array of n elements. Find two elements in the array such that their sum is equal to given element K.

```
void search(int A[], int n, int k) {
    for(int i = 0; i < n; i++) {
        for(int j = il j < n; j++) {
            if(A[i] + A[j] == k) {
                cout << "Nums found" << i << j;
                return;
            }
        }
    }
    cout << "Nums not found";
}
```

Improved :

```
void search(int A[], int n, int k) {
    int loind, hiind, sum;
    sort(A, n);
    for(loind = 0; hiind = n-1; loind < hiind) {
        sum = A[loind] + A[hiind];
        if(sum == k) {
            cout << "Elements found" << loind << hiind;
            return;
        } else if(sum < k) {
            loind++;
        } else {
            hiind--;
        }
    }
    return;
}
```

Problem 29 :

Given an array A of n elements. Find three indices, i,j & k such that A[i]2 +A[j]2 = A[k]2

```
sort(A);
for(int i = 0; i < n; i++) {
    A[i] = A[i] * A[i];
}
for(int i = n; i > 0; i--) {
    res = false;
    if(res) {
        //two sum sol
    }
}
```

Problem 30 & 31 :

Two elements whose sum is closest to zero. Given an array with both positive and negative numbers, find the two elements such that their sum is closest to zero. For the below array, algorithm should give -80 and 85. Example:

1 60 -10 70 -80 85

Brute Force :

```
void solve(int A[], int n) {
    int i, j, min_sum, nim_i, min_j, inv_count = 0;
    if(n < 2) {
        cout << "Invalid input\n";
        return;
    }
    min_i = 0;
    min_j = 1;
    min_sum = A[0] + A[1];
    for(int i = 0; i < n-1; i++) {
        for(int j = i+1; j < n; j++) {
            sum = A[i] + A[j];
            if(abs(min_sum) > abs(sum)) {
                min_sum = sum;
                min_i = i;
                min_j = j;
            }
        }
    }
    cout << "The two elements are " << A[min_i] << A[min_j];
}
```

Improved TC by sorting :

```
void solve(int A[], int n) {
    int i = 0, j = n-1, temp, positiveClosest = INT_MAX,
negativeClosest = INT_MIN;
    sort(A, n);
    while(i < j) {
        temp = A[i] + A[j];
        if(temp > 0) {
            if(temp < positiveClosest) {
                positiveClosest = temp;
            }
            j--;
        } else if(temp < 0) {
            if(temp > negativeClosest) {
                negativeClosest = temp;
            }
            i++;
        } else {
            cout << "Closest Sum " << A[i]+A[j];
        }
    }
    return (abs(negativeClosest) > positiveClosest ? positiveClosest:
negativeClosest);
}
```

Problem 32 & 34 :

Given an array of n elements. Find three elements in the array such that their sum is equal to given element K?

Brute Force : 3 for loops

Using sorting :

```
void search(int A[], int n, int data) {
    sort(A, n);
    for(int k = 0; k < n; k++) {
        for(int i = k+1, j = n-1; i < j;) {
            if(A[k] + A[i] + A[j] == data) {
                cout << "Found " << i << j << j;
            } else if(A[k] + A[i] + A[j] < data) {
                i++;
            } else {
                j--;
            }
        }
    }
    return;
}
```

Problem 37 & 42 & 43:

Let A be an array of n distinct integers. Suppose A has the following property: there exists an index 1 ≤ k ≤ n such that A[1],..., A[k] is an increasing sequence and A[k + 1],..., A[n] is a decreasing sequence. Design and analyze an efficient algorithm for finding k. Similar question: Let us assume that the given array is sorted but starts with negative numbers and ends with positive numbers [such functions are called monotonically increasing functions]. In this array find the starting index of the positive numbers. Assume that we know the length of the input array. Design a O(logn) algorithm.

Idea : use a variant of binary search

```
int search(int A[], int n, int first, int last) {
    int mid, first = 0, last = n-1;
    while(first <= last) {
        if(first == last) {
            return A[first];
        } else if(first == last-1) {
            return max(A[first], A[last]);
        } else {
            mid = (first + last) / 2;
            if(A[mid-1] < A[mid] && A[mid] > A[mid+1]) {
                return A[mid];
            } else if(A[mid-1] < A[mid] && A[mid] < A[mid+1]) {
                first = mid+1;
            } else if(A[mid-1] > A[mid] && A[mid] > A[mid+1]) {
                last = mid-1;
            } else {
                return INT_MIN;
            }
        }
    }
}
```

Problem 40 & 41:

Given a sorted array of n integers that has been rotated an unknown number of times, give a O(logn) algorithm that finds an element in the array. Example: Find 5 in array (15 16 19 20 25 1 3 4 5 7 10 14) Output: 8 (the index of 5 in the array)

Idea : using above question’s sol
Find pivot and divide into two subarrays. Call binary search on one of the two subarrays

```
int FindPivot(int A[], int start, int finish) {
    if(finish == start) {
        return start;
    } else if(start == finish - 1) {
        if(A[start] > A[finish]) {
            return start;
        } else {
            return finish;
        }
    } else {
        mid = (start + finish) / 2;
        if(A[start] >= A[mid]) {
            return FindPivot(A, start, mid);
        } else {
            return FindPivot(A, mid, finish);
        }
    }
}

int Search(int A[], int n, int x) {
    int pivot = FindPivot(A, 0, n-1);
    if(A[pivot] == x) {
        return pivot;
    }
    if(A[pivot] <= x) {
        return BinarySearch(A, 0, pivot-1, x);
    } else {
        reurn BinarySearch(A, pivot+1, n-1, x);
    }
}
```

Solving using recursion :

```
int BinarySearchRotated(int A[], int start, int finish, int data) {
    int mid = (start + finsh) / 2;
    if(start > finsh) {
        return -1;
    }
    if(data == A[mid]) {
        return mid;
    } else if(A[start] <= A[mid]) {
        if(data >= A[start] && data < A[mid]) {
            return BinarySearchRotated(A, start, mid-1, data);
        } else {
            return BinarySearchRotated(A, mid+1, finish, data);
        }
    } else {
        if(data > A[mid] && data <= A[finish]) {
            return BinarySearchRotated(A, mid+1, finish, data);
        } else {
            return BinarySearchRotated(A, start, mid-1, data);
        }
    }
}
```

<u>Problem 46 :</u>

Given a sorted array A of n elements, possibly with duplicates, find the index of the first occurrence of a number in O(logn) time.

```
int BinarySearchFirstOccurrence(int A[], int low, int high, int data)
{
    int mid;
    if(high >= low) {
        mid = (low + high) / 2;
        if((mid == low && A[mid] == data) || (A[mid] == data &&
A[mid-1] < data)) {
            return mid;
        } else if(A[mid] >= data) {
            return BinarySearchFirstOccurrence(A, low, mid-1, data);
        } else {
            return BinarySearchFirstOccurrence(A, mid+1, high, data);
        }
    }
    return -1;
}
```

Problem 47 :

Given a sorted array A of n elements, possibly with duplicates. Find the index of the last occurrence of a number in O(logn) time.

```
int BinarySearchLastOccurrence(int A[], int low, int high, int data)
{
    int mid;
    if(high >= low) {
        mid = (low + high) / 2;
        if((mid == low && A[mid] == data) || (A[mid] == data && 
A[mid+1] > data)) {
            return mid;
        } else if(A[mid] <= data) {
            return BinarySearchFirstOccurrence(A, mid+1, high, data);
        } else {
            return BinarySearchFirstOccurrence(A, low, mid-1, data);
        }
    }
    return -1;
}
```

Problem 48 :

Given a sorted array of n elements, possibly with duplicates. Find the number of occurrences of a number.

Brute force :

```
int LinearSearchCount(int A[], int n, int data) {
    int count = 0;
    for(int i = 0; i < n; i++) {
        if(A[i] == data) {
            count++;
        }
    }
    return count;
}
```

Other way : by sorting

Improved TC :

Algorithm:

- Find first occurrence of *data* and call its index as *firstOccurrence* (for algorithm refer to Problem-46)
- Find last occurrence of *data* and call its index as *lastOccurrence* (for algorithm refer to Problem-47)
- Return *lastOccurrence* – *firstOccurrence* + 1

Time Complexity = O(*logn* + *logn*) = *O*(*logn*).

Problem 58 :

An element is a majority if it appears more than n/2 times. Give an algorithm takes an array of n element as argument and identifies a majority (if it exists).

```
int MajorityNum(int A[], int n) {
    int count = 0, element = -1;
    for(int i = 0; i < n; i++) {
        if(count == 0) {
            element = A[i];
            count = 1;
        } else if(element == A[i]) {
            count++;
        } else {
            count--;
        }
    }
    return element;
}

//O(n), O(1)
```

Problem 60 :

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.

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

Problem 67 :

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

Idea : logic similar to quick sort

```
void solve(int A[], int n) {
    int left = 0; right = n-1;
    while(left < right) {
        while(A[left]%2 == 0 && left < right) {
            left++;
        }
        while(A[right]%2 == 1 && left < right) {
            right--;
        }
        if(left < right) {
            swap(&A[left], &A[right]);
            left++;
            right--;
        }
    }
}
```

Problem 69 :

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 the left side and 1’s on the right side of the array. Traverse the 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]

```
void Separate0and1(int A[], int n) {
    int left = 0, right = n - 1;

    while (left < right) {
        while (A[left] == 0 && left < right) {
            left++;
        }
        while (A[right] == 1 && left < right) {
            right--;
        }
        if (left < right) {
            A[left] = 0;
            A[right] = 1;
            left++;
            right--;
        }
    }
}
```

Problem 70 :

Sort an array of 0’s, 1’s and 2’s [or R’s, G’s and B’s]: Given an array A[] consisting of 0’s, 1’s and 2’s, give an algorithm for sorting A[].The algorithm should put all 0’s first, then all 1’s and finally all 2’s at the end. Example Input = {0,1,1,0,1,2,1,2,0,0,0,1}, Output = {0,0,0,0,0,1,1,1,1,1,2,2}

```
void Sorting012sDutchFlagProblem(int A[], int n) {
    int low = 0, mid = 0, high = n - 1;

    while (mid <= high) {
        switch (A[mid]) {
            case 0:
                swap(A[low], A[mid]);
                low++;
                mid++;
                break;
            case 1:
                mid++;
                break;
            case 2:
                swap(A[mid], A[high]);
                high--;
                break;
        }
    }
}
```

Problem 73 :

Given a number n, give an algorithm for finding the number of trailing zeros in n!

```
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;
}
```

Problem 74 :

Given an array of 2n integers in the following format a1 a2 a3 ...an b1 b2 b3 ...bn. Shuffle the array to a1 b1 a2 b2 a3 b3 ... an bn without any extra memory

```
void ShuffleArray() {
    int n = 4;
    int A[] = {1, 3, 5, 7, 2, 4, 6, 8};
    for (int i = 0, q = 1, k = n; i < n; i++, k++, q++) {
        for (int j = k; j > i + q; j--) {
            int tmp = A[j - 1];
            A[j - 1] = A[j];
            A[j] = tmp;
        }
    }
    for (int i = 0; i < 2 * n; i++)
        printf("%d ", A[i]);
}
```

Problem 76 :

Given an array A[], find the maximum j - i such that A[j] > A[i]. For example, Input: {34, 8, 10, 3, 2, 80, 30, 33, 1} and Output: 6 (j = 7, i = 1).

Brute Force :

```
int maxIndexDiff(int A[], int n) {
    int maxDiff = -1;
    int i, j;
    for(i = 0; i < n; i++) {
        for(j = n-1; j > i; j--) {
            if(A[j] > A[i] && maxDiff < (j-i)) {
                maxDiff = j-i;
            }
        }
    }
    return maxDiff;
}
```

Improved TC :

```
int maxIndexDiff(int A[], int n) {
    int maxDiff, i, j;
    int *LeftMins, *RightMaxs;
    LeftMins[0] = A[0];
    for(int i = 1; i < n; i++) {
        LeftMins[i] = min(A[i], LeftMins[i-1]);
    }
    RightMaxs[n-1] = A[n-1];
    for(j = n-2; j >= 0; j--) {
        RightMaxs[j] = max(A[j], RightMaxs[j+1]);
    }
    i = 0, j =0, maxDiff = -1;
    while(j < n && i < n) {
        if(LeftMins[i] < RightMaxs[j]) {
            maxDiff = max(maxDiff, j-i);
            j++;
        } else {
            i++;
        }
    }
    return maxDiff;
}
```

Problem 78 :

Given an array of elements, how do you check whether the list is pairwise sorted or not? A list is considered pairwise sorted if each successive pair of numbers is in sorted (non-decreasing) order.

```
int checkPairwiseSorted(int A[], int n) {
    if(n == 0 || n == 1) {
        return 1;
    }
    for(int i = 0; i < n-1; i+=2) {
        if(A[i] > A[i+1]) {
            return 0;
        }
    }
    return 1;
}
```

Problem 79 :

Given an array of n elements, how do you print the frequencies of elements without using extra space. Assume all elements are positive, editable and less than n.

Use negotiation technique

```
void frequencyCounter(int A[], int n) {
    int pos = 0;
    while(pos < n) {
        int expectedPos = A[pos] - 1;
        if(A[pos]>0 && A[expectedPos]>0) {
            swap(A[pos], A[expectedPos]);
            A[expectedPos] = -1;
        } else if(A[pos] > 0) {
            A[expectedPos]--;
            A[pos++] = 0;
        } else {
            pos++;
        }
    }
    for(int i = 0; i < n; i++) {
        cout << "Frequency is " << i+1 << abs(A[i]);
    }
}
```