Remove duplicates from a sorted arrayCW

https://leetcode.com/problems/remove-duplicates-from-sorted-array/

```
int removeDuplicates(vector<int>& nums) {
    int i = 0;
    for (int j = 1; j < nums.size(); j++) {
        if (nums[j] != nums[i]) {
            i++;
            nums[i] = nums[j];
        }
    }
    return i + 1;
}
```

Implementing insertion sortCW

https://www.geeksforgeeks.org/problems/insertion-sort/1

```
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i], j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```

Contains duplicateHW

https://leetcode.com/problems/contains-duplicate/description/

```
bool containsDuplicate(vector<int>& nums) {
    unordered_set<int> s(nums.begin(), nums.end());
    return s.size() != nums.size();
}
```

Two SumCW

https://leetcode.com/problems/two-sum/

```
vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int> m;
    for (int i = 0; i < nums.size(); i++) {
        if (m.count(target - nums[i])) return {m[target - nums[i]], i};
        m[nums[i]] = i;
    }
    return {};
}
```

Jump GameHW

https://leetcode.com/problems/jump-game/

```
bool canJump(vector<int>& nums) {
    int reach = 0;
    for (int i = 0; i < nums.size(); i++) {
        if (i > reach) return false;
        reach = max(reach, i + nums[i]);
    }
    return true;
}
```

Majority ElementHW

https://leetcode.com/problems/majority-element

```
bool isPalindrome(string s) {
    int left = 0, right = s.size() - 1;
    while (left < right) {
        while (left < right && !isalnum(s[left])) left++;
        while (left < right && !isalnum(s[right])) right--;
        if (tolower(s[left++]) != tolower(s[right--])) return false;
    }
    return true;}
```

Valid PalindromCW

https://leetcode.com/problems/valid-palindrome/

```
bool isPalindrome(string s) {
    int left = 0, right = s.size() - 1;
    while (left < right) {
        while (left < right && !isalnum(s[left])) left++;
        while (left < right && !isalnum(s[right])) right--;
        if (tolower(s[left++]) != tolower(s[right--])) return false;
    }
    return true;
}
```

Jump Game 2HW

https://leetcode.com/problems/jump-game-ii

```
int jump(vector<int>& nums) {
    int jumps = 0, end = 0, farthest = 0;
    for (int i = 0; i < nums.size() - 1; i++) {
        farthest = max(farthest, i + nums[i]);
        if (i == end) {
            jumps++;
            end = farthest;
        }
    }
    return jumps;
}
```

3SumHW

https://leetcode.com/problems/3sum/

```
vector<vector<int>> threeSum(vector<int>& nums) {
    sort(nums.begin(), nums.end());
    vector<vector<int>> res;
    for (int i = 0; i < nums.size(); i++) {
        if (i > 0 && nums[i] == nums[i - 1]) continue;
```

```
        int left = i + 1, right = nums.size() - 1;
        while (left < right) {
            int sum = nums[i] + nums[left] + nums[right];
            if (sum < 0) left++;
            else if (sum > 0) right--;
            else {
                res.push_back({nums[i], nums[left], nums[right]});
                while (left < right && nums[left] == nums[left + 1]) left++;
                while (left < right && nums[right] == nums[right - 1]) right--;
                left++; right--;
            }
        }
    }
    return res;
}
```

Set Matrix zeroesCW

https://leetcode.com/problems/set-matrix-zeroes/

```
void setZeroes(vector<vector<int>>& matrix) {
    int m = matrix.size(), n = matrix[0].size();
    bool firstRow = false, firstCol = false;

    for (int i = 0; i < m; i++) if (matrix[i][0] == 0) firstCol = true;
    for (int j = 0; j < n; j++) if (matrix[0][j] == 0) firstRow = true;

    for (int i = 1; i < m; i++)
        for (int j = 1; j < n; j++)
            if (matrix[i][j] == 0) matrix[i][0] = matrix[0][j] = 0;

    for (int i = 1; i < m; i++)
        for (int j = 1; j < n; j++)
            if (matrix[i][0] == 0 || matrix[0][j] == 0) matrix[i][j] = 0;
```

```
    if (firstRow) fill(matrix[0].begin(), matrix[0].end(), 0);
    if (firstCol) for (int i = 0; i < m; i++) matrix[i][0] = 0;
}
```

Longest substring withput repeating charactersHW

https://leetcode.com/problems/longest-substring-without-repeating-characters/description/

```
int lengthOfLongestSubstring(string s) {
    vector<int> lastIndex(256, -1);
    int maxLen = 0, start = -1;
    for (int i = 0; i < s.size(); i++) {
        if (lastIndex[s[i]] > start) start = lastIndex[s[i]];
        lastIndex[s[i]] = i;
        maxLen = max(maxLen, i - start);
    }
    return maxLen;
}
```

Finding duplicate numberHW

https://leetcode.com/problems/find-the-duplicate-number/description/

```
int findDuplicate(vector<int>& nums) {
    int slow = nums[0], fast = nums[0];
    do {
        slow = nums[slow];
        fast = nums[nums[fast]];
    } while (slow != fast);

    slow = nums[0];
    while (slow != fast) {
        slow = nums[slow];
        fast = nums[fast];
    }
    return slow;
```

```
}
```

Print linked listCW

https://www.geeksforgeeks.org/problems/print-linked-list-elements/0

```
void printList(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
}
```

Remove duplicates from a sorted listCW

https://leetcode.com/problems/remove-duplicates-from-sorted-list

```
ListNode* deleteDuplicates(ListNode* head) {
    ListNode* curr = head;
    while (curr && curr->next) {
        if (curr->val == curr->next->val)
            curr->next = curr->next->next;
        else
            curr = curr->next;
    }
    return head;
}
```

Reverse a linked listCW

https://leetcode.com/problems/reverse-linked-list/

```
ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    while (head) {
        ListNode* next = head->next;
        head->next = prev;
        prev = head;
        head = next;
    }
```

```
    return prev;
}
```

Delete middle node of a listCW

https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list

```
ListNode* deleteMiddle(ListNode* head) {
    if (!head || !head->next) return nullptr;
    ListNode *slow = head, *fast = head, *prev = nullptr;
    while (fast && fast->next) {
        prev = slow;
        slow = slow->next;
        fast = fast->next->next;
    }
    prev->next = slow->next;
    delete slow;
    return head;
}
```

Merge two sorted linked listsCW

https://leetcode.com/problems/merge-two-sorted-lists

```
ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    if (!l1) return l2;
    if (!l2) return l1;

    if (l1->val < l2->val) {
        l1->next = mergeTwoLists(l1->next, l2);
        return l1;
    } else {
        l2->next = mergeTwoLists(l1, l2->next);
        return l2;
    }
}
```

Remove duplicates from sorted lists 2CW

https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii

```
ListNode* deleteDuplicates(ListNode* head) {
    ListNode dummy(0);
    dummy.next = head;
    ListNode *prev = &dummy;

    while (head) {
        while (head->next && head->val == head->next->val)
            head = head->next;

        if (prev->next == head)
            prev = prev->next;
        else
            prev->next = head->next;

        head = head->next;
    }

    return dummy.next;
}
```

Detect a cycle in a linked listCW

https://leetcode.com/problems/linked-list-cycle

```
bool hasCycle(ListNode* head) {
    ListNode *slow = head, *fast = head;
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if (slow == fast) return true;
    }
    return false;
```

```
}
```

Reverse linked list 2CW

https://leetcode.com/problems/reverse-linked-list-ii

```
ListNode* reverseBetween(ListNode* head, int left, int right) {
    if (!head || left == right) return head;

    ListNode dummy(0);
    dummy.next = head;
    ListNode* prev = &dummy;

    for (int i = 1; i < left; i++) prev = prev->next;

    ListNode* curr = prev->next;
    ListNode* next = nullptr;

    for (int i = 0; i < right - left; i++) {
        next = curr->next;
        curr->next = next->next;
        next->next = prev->next;
        prev->next = next;
    }

    return dummy.next;
}
```

rotate a listCW

https://leetcode.com/problems/rotate-list

```
ListNode* rotateRight(ListNode* head, int k) {
    if (!head || !head->next || k == 0) return head;

    int length = 1;
    ListNode* tail = head;
```

```
    while (tail->next) {
        tail = tail->next;
        length++;
    }

    k = k % length;
    if (k == 0) return head;

    tail->next = head;
    for (int i = 0; i < length - k; i++)
        tail = tail->next;

    head = tail->next;
    tail->next = nullptr;

    return head;
}
```

Merge k sorted listsCW

https://leetcode.com/problems/merge-k-sorted-lists/

```
struct Compare {
    bool operator()(ListNode* a, ListNode* b) {
        return a->val > b->val;
    }
};

ListNode* mergeKLists(vector<ListNode*>& lists) {
    priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap;

    for (ListNode* list : lists) {
        if (list) minHeap.push(list);
    }
```

```
    ListNode dummy(0), *tail = &dummy;

    while (!minHeap.empty()) {
        tail->next = minHeap.top();
        minHeap.pop();
        tail = tail->next;
        if (tail->next) minHeap.push(tail->next);
    }

    return dummy.next;
}
```

Sort ListHW

https://leetcode.com/problems/sort-list/description/

```
ListNode* merge(ListNode* l1, ListNode* l2) {
    if (!l1) return l2;
    if (!l2) return l1;

    if (l1->val < l2->val) {
        l1->next = merge(l1->next, l2);
        return l1;
    } else {
        l2->next = merge(l1, l2->next);
        return l2;
    }
}

ListNode* sortList(ListNode* head) {
    if (!head || !head->next) return head;

    ListNode* slow = head, *fast = head->next;
```

```
    while (fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
    }

    ListNode* mid = slow->next;
    slow->next = nullptr;

    return merge(sortList(head), sortList(mid));
}
```

Merge k sorted listsCW

https://leetcode.com/problems/merge-k-sorted-lists/

```
struct Compare {
    bool operator()(ListNode* a, ListNode* b) {
        return a->val > b->val;
    }
};

ListNode* mergeKLists(vector<ListNode*>& lists) {
    priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap;

    for (ListNode* list : lists) {
        if (list) minHeap.push(list);
    }

    ListNode dummy(0), *tail = &dummy;

    while (!minHeap.empty()) {
        tail->next = minHeap.top();
        minHeap.pop();
        tail = tail->next;
```

```
        if (tail->next) minHeap.push(tail->next);
    }

    return dummy.next;
}
```