quickPlacement Preparation
Must Do Coding Questions
Answer Sheet

Array

Subarray with given sum
Count the triplets
Kadane's Algorithm
Missing number in array
Merge two sorted arrays
Rearrange array alternatively
Number of pairs
Inversion of Array
Sort an array of 0s, 1s and 2s
Equilibrium point
Leaders in an array
Minimum Platforms
Reverse array in groups
K'th smallest element
Trapping Rain Water
Pythagorean Triplet
Chocolate Distribution Problem
Stock buy and sell
Element with left side smaller and right side greater
Convert array into Zig-Zag fashion
Last Index of 1
Spirally traversing a matrix
Largest Number formed from an Array

String

Reverse words in a given string
Permutations of a given string
Longest Palindrome in a String
Recursively remove all adjacent duplicates
Check if string is rotated by two places
Roman Number to Integer
Anagram
Remove Duplicates
Form a Palindrome
Longest Distinct Characters in the string
Implement Atoi

Implement strstr
Longest Common Prefix

Linked List

Finding middle element in a linked list - Done
Reverse a linked list - Done
Rotate a Linked List - Done
Reverse a Linked List in groups of given size - Done
Intersection point in Y shaped linked lists
Detect Loop in linked list
Remove loop in Linked List
n’th node from end of linked list
Flattening a Linked List
Merge two sorted linked lists
Intersection point of two Linked Lists
Pairwise swap of a linked list
Add two numbers represented by linked lists
Check if Linked List is Palindrome
Implement Queue using Linked List
Implement Stack using Linked List
Given a linked list of 0s, 1s and 2s, sort it
Delete without head pointer

Stack and Queue

Parenthesis Checker
Next larger element
Queue using two Stacks
Stack using two queues
Get minimum element from stack
LRU Cache
Circular tour
First non-repeating character in a stream
Rotten Oranges
Maximum of all subarrays of size k

Tree

Print Left View of Binary Tree
Check for BST
Print Bottom View of Binary Tree
Print a Binary Tree in Vertical Order
Level order traversal in spiral form
Connect Nodes at Same Level
Lowest Common Ancestor in a BST
Convert a given Binary Tree to Doubly Linked List

Write Code to Determine if Two Trees are Identical or Not
Given a binary tree, check whether it is a mirror of itself
Height of Binary Tree
Maximum Path Sum
Diameter of a Binary Tree
Number of leaf nodes
Check if given Binary Tree is Height Balanced or Not
Serialize and Deserialize a Binary Tree

Heap

Find median in a stream
Heap Sort
Operations on Binary Min Heap
Rearrange characters
Merge K sorted linked lists
Kth largest element in a stream
Recursion
Flood fill Algorithm
Number of paths
Combination Sum – Part 2
Special Keyboard
Josephus problem

Hashing

Relative Sorting
Sorting Elements of an Array by Frequency
Largest subarray with 0 sum
Common elements
Find all four sum numbers
Swapping pairs make sum equal
Count distinct elements in every window
Array Pair Sum Divisibility Problem
Longest consecutive subsequence
Array Subset of another array
Find all pairs with a given sum
Find first repeated character
Zero Sum Subarrays
Minimum indexed character
Check if two arrays are equal or not
Uncommon characters
Smallest window in a string containing all the characters of another string
First element to occur k times
Check if frequencies can be equal

Graph

Depth First Traversal
Breadth First Traversal
Detect cycle in undirected graph
Detect cycle in a directed graph
Topological sort
Find the number of islands
Implementing Dijkstra
Minimum Swaps
Strongly Connected Components
Shortest Source to Destination Path
Find whether path exist
Minimum Cost Path
Circle of Strings
Floyd Warshall
Alien Dictionary
Snake and Ladder Problem
Greedy
Activity Selection
N meetings in one room
Coin Piles
Maximize Toys
Page Faults in LRU
Largest number possible
Minimize the heights
Minimize the sum of product
Huffman Decoding
Minimum Spanning Tree
Shop in Candy Store
Geek collects the balls

Dynamic Programming

Minimum Operations
Max length chain
Minimum number of Coins
Longest Common Substring
Longest Increasing Subsequence
Longest Common Subsequence
0 – 1 Knapsack Problem
Maximum sum increasing subsequence
Minimum number of jumps
Edit Distance
Coin Change Problem
Subset Sum Problem
Box Stacking
Rod Cutting
Path in Matrix

Minimum sum partition
Count number of ways to cover a distance
Egg Dropping Puzzle
Optimal Strategy for a Game
Shortest Common Supersequence

Divide and Conquer

Find the element that appears once in sorted array
Search in a Rotated Array
Binary Search
Sum of Middle Elements of two sorted arrays
Quick Sort
Merge Sort
K-th element of two sorted Arrays

Backtracking

N-Queen Problem
Solve the Sudoku
Rat in a Maze Problem
Word Boggle
Generate IP Addresses
Bit Magic
Find first set bit
Rightmost different bit
Check whether K-th bit is set or not
Toggle bits given range
Set kth bit
Power of 2
Bit Difference
Rotate Bits
Swap all odd and even bits
Count total set bits
Longest Consecutive 1’s
Sparse Number
Alone in a couple
Maximum subset XOR

Some More Questions on Arrays

Find Missing And Repeating
Maximum Index
Consecutive 1’s not allowed
Majority Element
Two numbers with sum closest to zero
Nuts and Bolts Problem

Boolean Matrix Problem
Smallest Positive missing number
Jumping Caterpillars

Some More Questions on Strings

Most frequent word in an array of strings
CamelCase Pattern Matching
String Ignorance
Smallest window in a string containing all the characters of another string
Design a tiny URL or URL shortener
Permutations of a given string
Non Repeating Character
Check if strings are rotations of each other or not
Save Ironman
Repeated Character
Remove common characters and concatenate
Geek and its Colored Strings
Second most repeated string in a sequence

Some more Questions on Trees

Mirror Tree
Longest consecutive sequence in Binary tree
Bottom View of Binary Tree
Lowest Common Ancestor in a Binary Tree
Binary to DLL

- [x] ~~Arrays~~
- [x] ~~String~~
- [x] ~~Linked List~~
- [x] ~~Stack and Queue~~
- [x] ~~Tree and BST~~
- [ ] Heap
- [x] ~~Recursion~~
- [ ] Hashing
- [ ] Graph
- [ ] Greedy
- [x] ~~Dynamic Programming~~
- [x] ~~Divide and Conquer~~
- [ ] Backtracking
- [ ] Bit Magic

Arrays

Subarray with given sum

```
#include <bits/stdc++.h>
using namespace std;
```

```
void subarraySum(int arr[], int n, int sum){
    map<int,int>mp;
    int res = 0;
    for(int i=0; i<n; i++){
        res += arr[i];
        if(res == sum){
            cout<<0<<":"<<i<<endl;
        }
        if(mp.find(sum-res) != mp.end()){
            cout<<mp[sum-res]<<":"<<i<<endl;
        }
        mp[res] = i
    }
}

int main(){
    int arr[] = {10, 2, -2, -20, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    int sum = -10;
    subarraySum(arr, n, sum);
    return 0;
}
```

Count the triplets

```
#include <bits/stdc++.h>
using namespace std;

void countTriplets(int arr[], int n){
    unordered_set<int> s;
    for(int i=0; i<n; i++){
        s.insert(arr[i]);
    }
    int count = 0;
    for(int i=0; i<n; i++){
        for(int j=0; j<i; j++){
            int sum = arr[i] + arr[j];
            if(s.find(sum) != s.end()){
                count += 1;
            }
        }
    }
    cout<<count;
}
```

```
int main(){
    int arr[] = {1, 5, 3, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    countTriplets(arr, n);
    return 0;
}
```

Kadane's Algorithm

```
#include <bits/stdc++.h>
using namespace std;

void kadane(int arr[], int n){
    int max_so_far = 0;
    int final_max = INT_MIN;
    for(int i=0; i<n; i++){
        max_so_far += arr[i];
        if(final_max < max_so_far){
            final_max = max_so_far;
        }
        if(max_so_far < 0){
            max_so_far = 0;
        }
    }
    cout<<final_max;
}

int main(){
    int arr[] = {-2, -3, 4, -1, -2, 1, 5, -3};
    int n = sizeof(arr)/sizeof(arr[0]);
    kadane(arr, n);
    return 0;
}
```

Missing number in array

```
#include <bits/stdc++.h>
using namespace std;

void missingNum(int arr[], int n){
    int res = (n+1)*(n+2)/2;
    for(int i=0; i<n; i++){
```

```
        res -= arr[i];
    }
    cout<<res;
}

int main(){
    int arr[] = {1,2,4,5,6};
    int n = sizeof(arr)/sizeof(arr[0]);
    missingNum(arr, n);
    return 0;
}
```

Merge two sorted arrays

```
#include <bits/stdc++.h>
using namespace std;

void mergeArrays(int arr1[], int arr2[], int m, int n){
    int res[m+n];
    int i=0, j=0, k=0;
    while(i<m && j<n){
        if(arr1[i]<arr2[j]){
            res[k++] = arr1[i++];
        }
        else{
            res[k++] = arr2[j++];
        }
    }
    while(i<m){
        res[k++] = arr1[i++];
    }
    while(j<n){
        res[k++] = arr2[j++];
    }
    for(auto it: res){
        cout<<it<<" ";
    }
}

int main(){
    int arr1[] = { 3, 4, 7, 9 };
    int m = sizeof(arr1)/sizeof(arr1[0]);
    int arr2[] = { 5, 6, 8, 9 };
    int n = sizeof(arr2)/sizeof(arr2[0]);
```

```
    mergeArrays(arr1, arr2, m, n);
    return 0;
}
```

Rearrange array alternatively

```
#include <bits/stdc++.h>
using namespace std;

void rearrange(int arr[], int n){
    int temp[n];
    bool flag = true;
    int high = n-1, low = 0;
    for(int i=0; i<n; i++){
        if(flag){
            temp[i] = arr[high--];
        }
        else{
            temp[i] = arr[low++];
        }
        flag = (!flag);
    }
    for(int i=0; i<n ; i++){
        cout<<temp[i]<<" ";
    }
}

int main(){
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = sizeof(arr)/sizeof(arr[0]);
    rearrange(arr, n);
    return 0;
}
```

Sort an array of 0s, 1s and 2s

```
#include <bits/stdc++.h>
using namespace std;

void sortNum(int arr[], int n){
    map<int,int> mp;
    for(int i=0; i<n; i++){
        mp[arr[i]]++;
```

```
    }
    for(auto it: mp){
        for(int i=0; i<it.second; i++){
            cout<<it.first<<" ";
        }
    }
}

int main(){
    int arr[] = { 0, 1, 1, 0, 2, 1, 0, 2, 1};
    int n = sizeof(arr)/sizeof(arr[0]);
    sortNum(arr, n);
    return 0;
}
```

Equilibrium point

```
#include <bits/stdc++.h>
using namespace std;

void equilibrium(int arr[], int n){
    int left_sum = 0;
    int sum = 0;
    for(int i=0; i<n; i++){
        sum += arr[i];
    }
    for(int i=0; i<n; i++){
        sum -= arr[i];
        if(sum == left_sum){
            cout<<i;
            return;
        }
        left_sum += arr[i];
    }
}

int main(){
    int arr[] = { -7, 1, 5, 2, -4, 3, 0 };
    int n = sizeof(arr)/sizeof(arr[0]);
    equilibrium(arr, n);
    return 0;
}
```

Leaders in an array

```
#include <bits/stdc++.h>
using namespace std;

void leaders(int arr[], int n){
    int right_max = arr[n-1];
    cout<<arr[n-1]<<" ";
    for(int i=n-2; i>=0; i--){
        if(arr[i] > right_max){
            cout<<arr[i]<<" ";
            right_max = arr[i];
        }
    }
}

int main(){
    int arr[] = {16, 17, 4, 3, 5, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    leaders(arr, n);
    return 0;
}
```

Time Complexity: O(n)

Minimum Platforms

```
#include <bits/stdc++.h>
using namespace std;

void maxPlatforms(int arr[], int dep[], int n){
    int i=0;
    int platform = 1;
    for(int j=1; j<n; j++){
        if(arr[j]>dep[i]){
            platform++;
        }
        i++;
    }
    cout<<platform;
}

int main(){
```

```
    int arr[] = { 900, 940, 950, 1100, 1500, 1800 };
    int dep[] = { 910, 1200, 1120, 1130, 1900, 2000 };
    int n = sizeof(arr)/sizeof(arr[0]);
    maxPlatforms(arr, dep, n);
    return 0;
}
```

Reverse array in groups

```
#include <bits/stdc++.h>
using namespace std;

void kReverse(int arr[], int n, int k){
    for(int i=0; i<n; i+=k){
        int left = i;
        int right = min(i + k - 1, n - 1);
        while(left<right){
            swap(arr[left++], arr[right--]);
        }
    }
    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }
}

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

K’th smallest element

```
#include <bits/stdc++.h>
using namespace std;

int *harr;
int hsize;

int left(int i){
    return (2*i+1);
```

```
}

int right(int i){
    return (2*i+2);
}

void minHeapify(int i){
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if(l<hsize && harr[l] < harr[i]){
        smallest = l;
    }
    if(r<hsize && harr[r] < harr[smallest]){
        smallest = r;
    }
    if(smallest != i){
        swap(harr[i], harr[smallest]);
        minHeapify(smallest);
    }
}

void minHeap(int arr[], int n){
    harr = arr;
    hsize = n;
    int i = (hsize-1)/2;
    while(i){
        minHeapify(i);
        i--;
    }
}

int extractMin(){
    if(hsize == 0){
        return INT_MAX;
    }
    int root = harr[0];
    if(hsize > 1){
        harr[0] = harr[hsize - 1];
        minHeapify(0);
    }
    hsize--;
    return root;
}
```

```
void kthSmallest(int arr[], int n, int k){
    minHeap(arr, n);
    for(int i=0; i<k-1; i++){
        extractMin();
    }
    cout<<harr[0]<<endl;
}

int main(){
    int arr[] = {2,5,9,3,6};
    int n = sizeof(arr)/sizeof(arr[0]);
    int kthSmall = 2;
    int kthLarge = 4;
    kthSmallest(arr, n, kthSmall);
    kthSmallest(arr, n, n - kthLarge);
    return 0;
}
```

Trapping Rain Water

```
#include <bits/stdc++.h>
using namespace std;

void trappingWater(int arr[], int n){
    int left_max[n];
    int right_max[n];
    left_max[0] = arr[0];
    right_max[n-1] = arr[n-1];
    int res = 0;
    for(int i=0; i<n; i++){
        left_max[i] = max(left_max[i-1], arr[i]);
    }
    for(int i=n-1; i>=0; i--){
        right_max[i] = max(right_max[i+1], arr[i]);
    }
    for(int i=0; i<n; i++){
        res += min(left_max[i], right_max[i]) - arr[i];
    }
    cout<<res;
}

int main(){
```

```
    int arr[] = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };
    int n = sizeof(arr)/sizeof(arr[0]);
    trappingWater(arr, n);
    return 0;
}
```

Pythagorean Triplet

```
#include <bits/stdc++.h>
using namespace std;

bool checkTriplet(int arr[], int n){
    map<int,int> mp;
    for(int i=0; i<n; i++){
        mp[arr[i]] = arr[i]*arr[i];
    }
    for(auto it=mp.begin(); it != mp.end(); it++){
        cout<<it->second<<" ";
    }
    cout<<endl;
    for(int i=1; i<n; i++){
        for(int j=0; j<i; j++){
            int res = arr[i]*arr[i]+arr[j]*arr[j];
            if(mp.find(res) != mp.end()){
                cout<<res;
                return true;
            }
        }
    }
    return false;
}

int main()
{
    int arr[] = { 3, 2, 4, 6, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    if (checkTriplet(arr, n))
        cout << "Yes";
    else
        cout << "No";
}
```

Chocolate Distribution Problem

```
#include <bits/stdc++.h>
using namespace std;

void minChoc(int arr[], int n){
    int res[n];
    for(int i=0; i<n; i++){
        res[i] = 1;
    }
    int sum = 0;
    for(int i=1; i<n; i++){
        if(arr[i] > arr[i-1]){
            res[i] = res[i-1] + 1;
        }
        else{
            res[i] = 1;
        }
    }
    for(int j=n-2; j>=0; j--){
        if(arr[j+1] < arr[j]){
            res[j] = max(res[j], res[j+1] + 1);
        }
        else{
            res[j] = max(res[j], 1);
        }
    }
    for(int i=0; i<n; i++){
        sum += res[i];
    }
    cout<<sum;
}

int main(){
    int arr[] = { 23, 14, 15, 14, 56, 29, 14 };
    int n = sizeof(arr)/sizeof(arr[0]);
    minChoc(arr, n);
    return 0;
}
```

Stock buy and sell

```
#include <bits/stdc++.h>
using namespace std;
```

```
void stockBuySell(int arr[], int n){
    int minStock = arr[0];
    int tmpProfit = 0;
    int finalProfit = INT_MIN;
    for(int i=0; i<n; i++){
        minStock = min(minStock, arr[i]);
        tmpProfit = arr[i] - minStock;
        finalProfit = max(finalProfit, tmpProfit);
    }
    cout<<finalProfit;
}

int main(){
    int price[] = { 7, 1, 5, 3, 6, 4 };
    int n = sizeof(price)/sizeof(price[0]);
    stockBuySell(price, n);
    return 0;
}
```

Element with left side smaller and right side greater

```
#include <bits/stdc++.h>
using namespace std;

void leftmaxRightmin(int arr[], int n){
    int leftmax[n];
    int rightmin = INT_MAX;
    leftmax[0] = INT_MIN;
    for(int i=1; i<n; i++){
        leftmax[i] = max(leftmax[i-1], arr[i-1]);
    }
    for(int i=n-1; i>=0; i--){
        if(arr[i] < rightmin && arr[i] > leftmax[i]){
            cout<<i<<" ";
            return;
        }
        rightmin = min(rightmin, arr[i]);
    }
}

int main(){
    int arr[] = {5, 1, 4, 3, 6, 8, 10, 7, 9};
    int n = sizeof(arr)/sizeof(arr[0]);
```

```
    leftmaxRightmin(arr, n);
    return 0;
}
```

Convert array into Zig-Zag fashion

```
#include <bits/stdc++.h>
using namespace std;

void zigZag(int arr[], int n){
    bool flag = true;
    for(int i=0; i<n-1; i++){
        if(flag){
            if(arr[i] < arr[i+1]){
                swap(arr[i], arr[i+1]);
            }
        }
        else{
            if(arr[i] > arr[i+1]){
                swap(arr[i], arr[i+1]);
            }
        }
        flag = !flag;
    }
    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }
}

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

Last Index of 1

```
#include <bits/stdc++.h>
using namespace std;

void lastIndex(int arr[], int n){
    int res = 0;
```

```
    for(int i=0; i<n; i++){
        if(arr[i] == 1){
            res = i;
        }
    }
    cout<<res;
}

int main(){
    int arr[] = {0,1,0,0,0,1,0,1,0};
    int n = sizeof(arr)/sizeof(arr[0]);
    lastIndex(arr, n);
    return 0;
}
```

Spirally traversing a matrix

```
#include <iostream>
using namespace std;

#define R 4
#define C 4

void print(int arr[R][C], int i, int j, int m, int n){
    if(i>=m || j>=n){
        return;
    }
    for(int p=j; p<n; p++){
        cout<<arr[i][p]<<" ";
    }
    for(int p=i+1; p<m; p++){
        cout<<arr[p][n-1]<<" ";
    }
    if(m-1 != i){
        for(int p=n-2; p>=j; p--){
            cout<<arr[m-1][p]<<" ";
        }
    }
    if(n-1 != i){
        for(int p=m-2; p>i; p--){
            cout<<arr[p][j]<<" ";
        }
    }
```

```
    print(arr, i+1, j+1, m-1, n-1);
}

int main()
{

    int a[R][C] = { { 1, 2, 3, 4 },
                    { 5, 6, 7, 8 },
                    { 9, 10, 11, 12 },
                    { 13, 14, 15, 16 } };
    print(a, 0, 0, R, C);
    return 0;
}
```

Largest Number formed from an Array

```
#include <bits/stdc++.h>
using namespace std;

int compare(string X, string Y){
        string XY = X.append(Y);
        string YX = Y.append(X);
        return XY.compare(YX) > 0 ? 1 : 0;
}

void printLargest(vector<string> arr){
        sort(arr.begin(), arr.end(), compare);
        for (int i=0; i<arr.size(); i++){
            cout<<arr[i];
        }
}

int main(){
        vector<string> arr = {"54","546","548","60"};
        printLargest(arr);
        return 0;
}
```

Largest subarray with zero sum

```
#include <bits/stdc++.h>
using namespace std;
```

```
void subarrayZeroSum(int arr[], int n){
    unordered_map<int,int> mp;
    int sum = 0;
    int maxLen = 0;
    for(int i=0; i<n; i++){
        sum += arr[i];
        if(sum == 0){
            maxLen = max(maxLen, i+1);
        }
        if(mp.find(sum) == mp.end()){
            maxLen = max(maxLen, i-mp[sum]);
        }
        mp[sum] = i;
    }
    cout<<maxLen;
}

int main(){
    int arr[] = {2,-1,-3,3,-1,4};
    int n = sizeof(arr)/sizeof(arr[0]);
    subarrayZeroSum(arr, n);
    return 0;
}
```

Trailing Zeros

```
#include <bits/stdc++.h>
using namespace std;

void trailingZeros(int n){
    int cnt = 0;
    while(n>5){
        cnt += n/5;
        n = n/5;
    }
    cout<<cnt;
}

int main(){
    int n = 100;
    trailingZeros(n);
    return 0;
}
```

Strings

Reverse words in a given string

```
#include <bits/stdc++.h>
using namespace std;

void reverseWords(string s){
    string temp = "";
    vector<string> v;
    for(int i=0; i<s.length(); i++){
        if(s[i] == ' '){
            v.push_back(temp);
            temp = "";
        }
        else{
            temp += s[i];
        }
    }
    v.push_back(temp);
    for (int i = v.size() - 1; i > 0; i--){
        cout << v[i] << " ";
    }
    cout << v[0] << endl;
}

int main(){
    string str = "I love Coding";
    reverseWords(str);
    return 0;
}
```

Permutations of a given string

```
#include <bits/stdc++.h>
using namespace std;

void permutate(string str, int l, int h){
    if(l == h){
        cout<<str<<endl;
```

```
    }
    else{
        for(int i=l; i<=h; i++){
            swap(str[l], str[i]);
            permutate(str, l+1, h);
            swap(str[l], str[i]);
        }
    }
}

int main(){
    string str = "ABC";
    int l = 0;
    int h = str.length() - 1;
    permutate(str, l ,h);
    return 0;
}
```

Longest Palindrome in a String
Recursively remove all adjacent duplicates
Check if string is rotated by two places
Roman Number to Integer
Anagram
Remove Duplicates
Form a Palindrome
Longest Distinct Characters in the string
Implement Atoi
Implement strstr
Longest Common Prefix

Alternate Merge

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    string str1 = "fisrt";
    string str2 = "second";
    int i = 0, j = 0;
    string res = "";
```

```
    while(i<str1.length() || j<str2.length()){
        if(i<str1.length()){
            res += str1[i];
            i++;
        }
        if(j<str2.length()){
            res += str2[j];
            j++;
        }
    }
    cout << res;
    return 0;
}
```

Lower Upper swap

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    string str = "This is Coding";
    for (int i=0; i<str.length(); i++){
        int c = str[i];
        if (islower(c)){
            str[i] = toupper(c);
        }
        if (isupper(c)){
            str[i] = tolower(c);
        }
    }
    cout << str;
    return 0;
}
```

Linked List

Finding middle element in a linked list

```
#include <bits/stdc++.h>
using namespace std;
```

```
class Node{
    public:
    int data;
    Node* next;
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void middlell(Node* head){
    Node* slow = head;
    Node* fast = head;
    while(fast && fast->next){
        slow = slow->next;
        fast = fast->next->next;
    }
    cout<<slow->data;
    return;
}

int main(){
    Node* head = NULL;
    push(&head, 10);
    push(&head, 23);
    push(&head, 56);
    push(&head, 12);
    middlell(head);
    return 0;
}
```

Reverse a linked list

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* next;
```

```
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void reverse(Node** head){
    Node* prev = NULL;
    Node* current = *head;
    Node* next = NULL;
    while (current) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

void print(Node* head){
    Node* temp = head;
    while(temp){
        cout<<temp->data<<"->";
        temp = temp->next;
    }
    cout<<"NULL"<<endl;
}

int main(){
    Node* head = NULL;
    push(&head, 12);
    push(&head, 17);
    push(&head, 24);
    push(&head, 32);
    push(&head, 87);
    print(head);
    reverse(&head);
    print(head);
    return 0;
}
```

Rotate a Linked List

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* next;
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

Node* rotate(Node* head, int k){
    Node* current = head;
    int cnt = 1;
    while(cnt<k && current){
        current = current->next;
        cnt++;
    }
    if(!current){
        return head;
    }
    Node* kthnode = current;
    while(current->next){
        current = current->next;
    }
    current->next = (head);
    (head) = kthnode->next;
    kthnode->next = NULL;
    return head;
}

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

```
    cout<<"NULL"<<endl;
}

int main(){
    Node* head = NULL;
    push(&head, 10);
    push(&head, 20);
    push(&head, 30);
    push(&head, 40);
    push(&head, 50);
    push(&head, 60);
    int k = 4;
    print(head);
    head = rotate(head, k);
    print(head);
    return 0;
}
```

Reverse a Linked List in groups of given size

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* next;
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}



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

```
    cout<<"NULL"<<endl;
}

int main(){
    Node* head = NULL;
    push(&head, 10);
    push(&head, 20);
    push(&head, 30);
    push(&head, 40);
    push(&head, 50);
    push(&head, 60);
    int k = 2;
    print(head);
    head = reversek(head, k);
    print(head);
    return 0;
}
```

Intersection point in Y shaped linked lists

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* next;
};

int intersection(Node* head1, Node* head2){
    Node* curr1 = head1;
    Node* curr2 = head2;
    while(curr1 != curr2){
        if(!curr1){
            curr1 = head2;
        }
        curr1 = curr1->next;
        if(!curr2){
            curr2 = head1;
        }
        curr2 = curr2->next;
    }
    return curr1->data;
}
```

```
void print(Node* head){
    Node* temp = head;
    while(temp){
        cout<<temp->data<<"->";
        temp = temp->next;
    }
    cout<<"NULL"<<endl;
}

int main(){
    Node* newNode;
    Node* head1 = new Node();
    head1->data = 10;
    Node* head2 = new Node();
    head2->data = 3;
    newNode = new Node();
    newNode->data = 6;
    head2->next = newNode;
    newNode = new Node();
    newNode->data = 9;
    head2->next->next = newNode;
    newNode = new Node();
    newNode->data = 15;
    head1->next = newNode;
    head2->next->next->next = newNode;
    newNode = new Node();
    newNode->data = 30;
    head1->next->next = newNode;
    head1->next->next->next = NULL;
    cout<<intersection(head1, head2);
    return 0;
}
```

Detect Loop in linked list

```
#include <bits/stdc++.h>
using namespace std;

class Node {
    public:
    int key;
    struct Node* next;
};
```

```
Node* newNode(int key){
    Node* new_node = new Node;
    new_node->key = key;
    new_node->next = NULL;
    return new_node;
}

void printList(Node* head){
    while(head != NULL){
        cout<<head->key << " ";
        head = head->next;
    }
    cout<<endl;
}

void detectAndRemove(Node* head){
    unordered_map<Node*, int> node_map;
    Node* last = NULL;
    while(head != NULL){
        if(node_map.find(head) == node_map.end()){
            node_map[head]++;
            last = head;
            head = head->next;
        }
        else{
            last->next = NULL;
            break;
        }
    }
}

int main(){
    Node* head = newNode(50);
    head->next = newNode(20);
    head->next->next = newNode(15);
    head->next->next->next = newNode(4);
    head->next->next->next->next = newNode(10);
    head->next->next->next->next->next = head->next->next;
    detectAndRemove(head);
    printList(head);
    return 0;
}
```

Remove loop in Linked List

```
#include <bits/stdc++.h>
using namespace std;

class Node {
    public:
    int key;
    struct Node* next;
};

Node* newNode(int key){
    Node* new_node = new Node;
    new_node->key = key;
    new_node->next = NULL;
    return new_node;
}

void printList(Node* head){
    while(head != NULL){
        cout<<head->key << " ";
        head = head->next;
    }
    cout<<endl;
}

void detectAndRemove(Node* head){
    unordered_map<Node*, int> node_map;
    Node* last = NULL;
    while(head != NULL){
        if(node_map.find(head) == node_map.end()){
            node_map[head]++;
            last = head;
            head = head->next;
        }
        else{
            last->next = NULL;
            break;
        }
    }
}

int main(){
```

```
    Node* head = newNode(50);
    head->next = newNode(20);
    head->next->next = newNode(15);
    head->next->next->next = newNode(4);
    head->next->next->next->next = newNode(10);
    head->next->next->next->next->next = head->next->next;
    detectAndRemove(head);
    printList(head);
    return 0;
}
```

n’th node from end of linked list

```
#include <bits/stdc++.h>
using namespace std;
class Node{
    public:
    int data;
    Node* next;
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

int nthfromend(Node* head, int n){
    Node* current = head;
    int len = 0;
    int res = 0;
    while(current){
        len++;
        current = current->next;
    }
    current = head;
    for(int i=0; i<len-n+1; i++){
        res = current->data;
        current = current->next;
    }
    return res;
}
```

```
int main(){
    Node* head = NULL;
    push(&head, 10);
    push(&head, 20);
    push(&head, 30);
    push(&head, 40);
    push(&head, 50);
    push(&head, 60);
    int n = 4;
    cout<<nthfromend(head, n);
    return 0;
}
```

Flattening a Linked List

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* right;
    Node* down;
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = NULL;
    new_node->down = (*head_ref);
    (*head_ref) = new_node;
}

Node* merge(Node* a, Node* b){
    if(!a){
        return b;
    }
    if(!b){
        return a;
    }
    Node* result;
    if(a->data < b->data){
        result = a;
        result->down = merge(a->down, b);
```

```
    }
    else{
        result = b;
        result->down = merge(a, b->down);
    }
    return result;
}

Node* flatten(Node* root){
    if(!root || !root->right){
        return root;
    }
    return merge(root, flatten(root->right));
}

void print(Node* root){
    Node* temp = root;
    while(temp){
        cout<<temp->data<<"->";
        temp = temp->down;
    }
    cout<<"NULL"<<endl;
}

int main(){
    Node* root = NULL;
    push( &root, 30 );
    push( &root, 8 );
    push( &root, 7 );
    push( &root, 5 );

    push( &( root->right ), 20 );
    push( &( root->right ), 10 );

    push( &( root->right->right ), 50 );
    push( &( root->right->right ), 22 );
    push( &( root->right->right ), 19 );

    push( &( root->right->right->right ), 45 );
    push( &( root->right->right->right ), 40 );
    push( &( root->right->right->right ), 35 );
    push( &( root->right->right->right ), 20 );
    root = flatten(root);
    print(root);
```

```
    return 0;
}
```

Merge two sorted linked lists

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* next;
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

Node* merge(Node* a, Node* b){
    if(!a){
        return b;
    }
    if(!b){
        return a;
    }
    Node* result = NULL;
    if(a->data < b->data){
        result = a;
        result->next = merge(a->next, b);
    }
    else{
        result = b;
        result->next = merge(a, b->next);
    }
    return result;
}

void print(Node* head){
    Node* temp = head;
    while(temp){
        cout<<temp->data<<"->";
```

```
        temp = temp->next;
    }
    cout<<"NULL"<<endl;
}

int main(){
    Node* a = NULL;
    Node* b = NULL;
    Node* root = NULL;
    push(&a, 15);
    push(&a, 10);
    push(&a, 5);
    push(&b, 20);
    push(&b, 3);
    push(&b, 2);
    root = merge(a, b);
    print(root);
    return 0;
}
```

Intersection point of two Linked Lists

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* next;
};

int intersection(Node* head1, Node* head2){
    Node* curr1 = head1;
    Node* curr2 = head2;
    while(curr1 != curr2){
        if(!curr1){
            curr1 = head2;
        }
        curr1 = curr1->next;
        if(!curr2){
            curr2 = head1;
        }
        curr2 = curr2->next;
    }
    return curr1->data;
```

```
}

void print(Node* head){
    Node* temp = head;
    while(temp){
        cout<<temp->data<<"->";
        temp = temp->next;
    }
    cout<<"NULL"<<endl;
}

int main(){
    Node* newNode;
    Node* head1 = new Node();
    head1->data = 10;
    Node* head2 = new Node();
    head2->data = 3;
    newNode = new Node();
    newNode->data = 6;
    head2->next = newNode;
    newNode = new Node();
    newNode->data = 9;
    head2->next->next = newNode;
    newNode = new Node();
    newNode->data = 15;
    head1->next = newNode;
    head2->next->next->next = newNode;
    newNode = new Node();
    newNode->data = 30;
    head1->next->next = newNode;
    head1->next->next->next = NULL;
    cout<<intersection(head1, head2);
    return 0;
}
```

Pairwise swap of a linked list

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* next;
```

```
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void pairwiseSwap(Node* head){
    Node* temp = head;
    while(temp && temp->next){
        swap(temp->data, temp->next->data);
        temp = temp->next->next;
    }
}

void print(Node* head){
    Node* temp = head;
    while(temp){
        cout<<temp->data<<" ";
        temp = temp->next;
    }
    cout<<endl;
}

int main(){
    Node* head = NULL;
    push(&head, 20);
    push(&head, 10);
    push(&head, 40);
    push(&head, 30);
    push(&head, 60);
    push(&head, 50);
    print(head);
    pairwiseSwap(head);
    print(head);
    return 0;
}
```

Add two numbers represented by linked lists

```
#include <bits/stdc++.h>
using namespace std;
```

```
class Node {
public:
	int data;
	Node* next;
};

Node* newNode(int data){
	Node* new_node = new Node();
	new_node->data = data;
	new_node->next = NULL;
	return new_node;
}

void push(Node** head_ref, int new_key){

	Node* new_node = new Node();
	new_node->data = new_key;
	new_node->next = (*head_ref);
	(*head_ref) = new_node;
}

Node* addTwoLists(Node* first, Node* second){
	Node* res = NULL;
	Node* temp = NULL;
	Node* prev = NULL;
	int carry = 0
	int sum;
	while (first || second){
		sum = carry + (first ? first->data : 0) + (second ? second->data : 0);
		carry = (sum >= 10) ? 1 : 0;
		sum = sum % 10;
		temp = newNode(sum);
		if(!res){
			res = temp;
		}
		else{
			prev->next = temp;
		}
		prev = temp;
		if(first){
			first = first->next;
		}
		if(second){
```

```
                        second = second->next;
                }
        }
        if(carry > 0){
           temp->next = newNode(carry);
        }
        return res;
}

void printList(Node* node){
        while(node != NULL) {
                cout<<node->data << " ";
                node = node->next;
        }
        cout << endl;
}

int main(void){
        Node* res = NULL;
        Node* first = NULL;
        Node* second = NULL;
        push(&first, 6);
        push(&first, 4);
        push(&first, 9);
        push(&first, 5);
        push(&first, 7);
        printList(first);
        push(&second, 4);
        push(&second, 8);
        printList(second);
        res = addTwoLists(first, second);
        printList(res);
        return 0;
}
```

Check if Linked List is Palindrome

```
#include <bits/stdc++.h>
using namespace std;

class Node{
   public:
   int data;
   Node* next;
```

```
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

bool isPalindrome(Node* head){
    Node* current = head;
    stack<int> s;
    while(current){
        s.push(current->data);
        current = current->next;
    }
    current = head;
    while(current){
        int i = s.top();
        s.pop();
        if(current->data != i){
            return false;
        }
        current = current->next;
    }
    return true;
}


int main(){
    Node* head = NULL;
    push(&head, 10);
    push(&head, 20);
    push(&head, 30);
    push(&head, 40);
    push(&head, 50);
    if(isPalindrome){
        cout<<"Yes";
    }
    else{
        cout<<"No";
    }
    return 0;
}
```

Given a linked list of 0s, 1s and 2s, sort it

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* next;
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void sortNum(Node* head){
    int count[3] = {0,0,0};
    Node* current = head;
    int i=0;
    while(current){
        count[current->data] += 1;
        current = current->next;
    }
    Node* temp = head;
    while(temp){
        if(count[i] == 0){
            i++;
        }
        else{
            temp->data = i;
            count[i]--;
            temp = temp->next;
        }
    }
}

void print(Node* head){
    Node* temp = head;
    while(temp){
```

```
        cout<<temp->data<<"->";
        temp = temp->next;
    }
    cout<<"END";
}

int main(){
    Node* head = NULL;
    push(&head, 0);
    push(&head, 2);
    push(&head, 1);
    push(&head, 2);
    push(&head, 1);
    push(&head, 0);
    sortNum(head);
    print(head);
    return 0;
}
```

Delete without head pointer

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* next;
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void delwithouthead(Node* del){
    if(!del->next){
        cout<<"Can't delete";
    }
    Node* temp = del->next;
    del->data = del->next->data;
    del->next = del->next->next;
```

```
    free(temp);
}

void print(Node* head){
    Node* temp = head;
    while(temp){
        cout<<temp->data<<"->";
        temp = temp->next;
    }
    cout<<"NULL"<<endl;
}

int main(){
    Node* head = NULL;
    push(&head, 10);
    push(&head, 20);
    push(&head, 30);
    push(&head, 40);
    push(&head, 50);
    Node* del = head->next->next;
    print(head);
    delwithouthead(del);
    print(head);
    return 0;
}
```

Dynamic Programming

Minimum Operations

```
#include <bits/stdc++.h>
using namespace std;

int minOperations(int n){
    int x = 0;
    int dp[n+1];
    dp[1] = 0;
    for(int i=2; i<=n; i++){
        dp[i] = INT_MAX;
        if(i%2 == 0){
```

```
            x = dp[i/2];
            if(x+1<dp[i]){
                dp[i] = x+1;
            }
        }
        if(i%3 == 0){
            x = dp[i/3];
            if(x+1<dp[i]){
                dp[i] = x+1;
            }
        }
        x = dp[i-1];
        if(x+1<dp[i]){
            dp[i] = x+1;
        }
    }
    return dp[n];
}

int main(){
    int n = 15;
    cout<<minOperations(n);
    return 0;
}
```

Max length chain

```
#include <bits/stdc++.h>
using namespace std;

class Pair{
    public:
    int a;
    int b;
};

int maxLengthChain(Pair arr[], int n){
    int max = INT_MIN;
    int dp[n+1] = {1};
    for(int i=1; i<=n; i++){
        for(int j=0; j<i; j++){
            if(arr[i].a > arr[j].b && dp[j]+1>dp[i]){
                dp[i] = dp[j]+1;
            }
```

```
        }
        if(dp[i]>max){
            max = dp[i];
        }
    }

    return max;
}

int main(){
    Pair arr[] = { {5, 24}, {15, 25}, {27, 40}, {50, 60} };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout<<maxLengthChain(arr, n);
    return 0;
}
```

Minimum number of Coins

Longest Common Substring

```
#include <bits/stdc++.h>
using namespace std;

int longestSub(char X[], char Y[], int m, int n){
    int dp[m+1][n+1] = {};
    int maxLen = 0;
    for(int i=0; i<=m; i++){
        for(int j=0; j<=n; j++){
            if(i==0 || j==0){
                dp[i][j] = 0;
            }
            if(X[i-1] == Y[j-1]){
                dp[i][j] = dp[i-1][j-1] + 1;
                maxLen = max(maxLen, dp[i][j]);
            }
            else{
                dp[i][j] = 0;
            }
        }
    }
    return maxLen;
}

int main(){
```

```
    char X[] = "OldSite:GeeksforGeeks.org";
    char Y[] = "NewSite:GeeksQuiz.com";
    int m = strlen(X);
    int n = strlen(Y);
    cout<<longestSub(X,Y,m,n);
    return 0;
}
```

Longest Increasing Subsequence

```
#include <bits/stdc++.h>
using namespace std;

int LIS(int arr[], int n){
    int dp[n+1] = {1};
    int maxLen = 0;
    for(int i=1; i<=n; i++){
        for(int j=0; j<i; j++){
            if(arr[i]>=arr[j] && dp[i]<dp[j]+1){
                dp[i] = dp[j]+1;
            }
        }
        maxLen = max(maxLen, dp[i]);
    }
    return maxLen;
}

int main(){
    int arr[] = { 10, 22, 9, 33, 21, 50, 41, 60 };
    int n = sizeof(arr)/sizeof(arr[0]);
    cout<<LIS(arr, n);
    return 0;
}
```

Longest Common Subsequence

```
#include <bits/stdc++.h>
using namespace std;

int LCS(char X[], char Y[], int m, int n){
    int dp[m+1][n+1] = {};
    int maxLen = 0;
    for(int i=0; i<=m; i++){
```

```
        for(int j=0; j<=n; j++){
            if(i==0 || j==0){
                dp[i][j] = 0;
            }
            else if(X[i-1] == Y[j-1]){
                dp[i][j] = dp[i-1][j-1] + 1;
            }
            else{
                dp[i][j] = max(dp[i-1][j-1], max(dp[i-1][j], dp[i][j-1]));
            }
        }
    }
    return dp[m][n];
}

int main(){
    char X[] = "AGGTAB";
    char Y[] = "GXTXAYB";
    int m = strlen(X);
    int n = strlen(Y);
    cout<<LCS(X, Y, m, n);
    return 0;
}
```

0 – 1 Knapsack Problem

```
#include <bits/stdc++.h>
using namespace std;

int knapsack(int val[], int wt[], int W, int n){
    int dp[n+1][W+1];
    for(int i=0; i<=n; i++){
        for(int j=0; j<=W; j++){
            if(i==0 || j==0){
                dp[i][j] = 0;
            }
            else if(wt[i-1]>j){
                dp[i][j] = dp[i-1][j];
            }
            else{
                dp[i][j] = max(val[i-1] + dp[i-1][j-wt[i-1]], dp[i-1][j]);
            }
        }
    }
```

```
    return dp[n][W];
}

int main(){
    int val[] = { 60, 100, 120 };
    int wt[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(val)/sizeof(val[0]);
    cout<<knapsack(val, wt, W, n);
    return 0;
}
```

Maximum sum increasing subsequence

```
#include <bits/stdc++.h>
using namespace std;

int MIS(int arr[], int n){
    int dp[n+1];
    int maxLen = 0;
    for(int i=0; i<=n; i++){
        dp[i] = arr[i];
    }
    for(int i=0; i<=n; i++){
        for(int j=0; j<i; j++){
            if(arr[i]>arr[j] && dp[i] < dp[j] + arr[i]){
                dp[i] = dp[j] + arr[i];
            }
        }
        if(maxLen < dp[i]){
            maxLen = dp[i];
        }
    }
    return maxLen;
}

int main(){
    int arr[] = {1, 101, 2, 3, 100, 4, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout<<MIS(arr, n);
    return 0;
}
```

Minimum number of jumps

Edit Distance

```
#include <bits/stdc++.h>
using namespace std;


int editDistance(char X[], char Y[], int m, int n){
    int dp[m+1][n+1];
    for(int i=0; i<=m; i++){
        for(int j=0; j<=n; j++){
            if(i==0){
                dp[i][j] = j;
            }
            else if(j==0){
                dp[i][j] = i;
            }
            else if(X[i-1] == Y[j-1]){
                dp[i][j] = dp[i-1][j-1];
            }
            else{
                dp[i][j] = 1 + min(dp[i-1][j-1], min(dp[i-1][j], dp[i][j-1]));
            }
        }
    }
    return dp[m][n];
}

int main(){
    char X[] = "sunday";
    char Y[] = "saturday";
    int m = strlen(X);
    int n = strlen(Y);
    cout<<editDistance(X, Y, m, n);
    return 0;
}
```

Coin Change Problem
Subset Sum Problem

```
#include <bits/stdc++.h>
```

```
using namespace std;

bool subsetSum(int set[], int sum, int n){
    int dp[n+1][sum+1];
    for(int i=0; i<=n ; i++){
        for(int j=0; j<=sum; j++){
            if(i==0){
                dp[i][j] = false;
            }
            else if(j==0){
                dp[i][j] = true;
            }
            else if(j < set[i-1]){
                dp[i][j] = dp[i-1][j];
            }
            else{
                dp[i][j] = dp[i-1][j] || dp[i-1][j-set[i-1]];
            }
        }
    }
    return dp[n][sum];
}

int main(){
    int set[] = { 3, 34, 4, 12, 2, 8};
    int sum = 10;
    int n = sizeof(set) / sizeof(set[0]);
    if(subsetSum(set, sum, n)){
        cout<<"Yes";
    }
    else{
        cout<<"No";
    }
    return 0;
}
```

Box Stacking
Rod Cutting
Path in Matrix
Minimum sum partition
Count number of ways to cover a distance
Egg Dropping Puzzle

Optimal Strategy for a Game
Shortest Common Supersequence

Find the element that appears once in sorted array

```
#include <bits/stdc++.h>
using namespace std;

void notPair(int arr[], int n){
    int res =0;
    for(int i=0; i<n; i++){
        res = res^arr[i];
    }
    cout<<res<<endl;
}

int main(){
    int arr[] = {11,2,3,2,4,3,11};
    int n = sizeof(arr)/sizeof(arr[0]);
    notPair(arr, n);
    return 0;
}
```

Search in a Rotated Array

```
#include <bits/stdc++.h>
using namespace std;

bool search(int arr[], int low, int high, int find){
    if(low>high){
        return false;
    }
    int mid = low + (high - low)/2;
    if(arr[mid] == find){
        return true;
    }
```

```
    if(arr[low]<arr[mid]){
        if(find>arr[low] && find<arr[mid-1]){
            return search(arr, low, mid-1, find);
        }
        return search(arr, mid+1, high, find);
    }
    if(find>arr[mid] && find<arr[high]){
        return search(arr, mid+1, high, find);
    }
    return search(arr, low, mid-1, find);
}

int main(){
    int arr[] = {19,23,1,3,4,6,7,8};
    int n = sizeof(arr)/sizeof(arr[0]);
    int low = 0;
    int high = n-1;
    int find = 6;
    if(search(arr, low, high, find)){
        cout<<"Found";
    }
    else{
        cout<<"Not Found";
    }
    return 0;
}
```

Binary Search

```
#include <bits/stdc++.h>
using namespace std;

bool search(int arr[], int low, int high, int find){
    if(low>high){
        return false;
    }
    int mid = (high + low)/2;
    if(arr[mid] == find){
        return true;
    }
    else if(arr[mid] > find){
        search(arr, low, mid-1, find);
    }
    else{
```

```
        search(arr, mid+1, high, find);
    }
}

int main(){
    int arr[] = {1,3,4,6,7,8,19,23};
    int n = sizeof(arr)/sizeof(arr[0]);
    int low = 0;
    int high = n-1;
    int find = 2;
    if(search(arr, low, high, find)){
        cout<<"Found";
    }
    else{
        cout<<"Not Found";
    }
    return 0;
}
```

Sum of Middle Elements of two sorted arrays

```
#include <bits/stdc++.h>
using namespace std;

void medianSum(int arr1[], int arr2[], int m, int n){
    int result[m+n];
    int i=0, j=0, k=0;
    while(i<m && j<n){
        if(arr1[i]<arr2[j]){
            result[k++] = arr1[i++];
        }
        else{
            result[k++] = arr2[j++];
        }
    }
    while(i<m){
        result[k++] = arr1[i++];
    }
    while(j<n){
        result[k++] = arr2[j++];
    }
    for(int i=0; i<k; i++){
        cout<<result[i]<<" ";
    }
```

```
    cout<<endl;
    if(k%2 != 0){
        cout<<"Median: "<<result[k/2];
    }
    else{
        cout<<result[k/2 - 1]<<" ";
        cout<<result[k/2]<<" ";
        cout<<"Median: "<<(result[k/2 - 1] + result[k/2])/2;
    }
}

int main(){
    int arr1[] = {2,3,5,7,8};
    int arr2[] = {3,4,9};
    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);
    medianSum(arr1, arr2, m, n);
    return 0;
}
```

Quick Sort

```
#include <bits/stdc++.h>
using namespace std;

int partition (int arr[], int low, int high)
{
        int pivot = arr[high];
        int i = low;
        for (int j = low; j < high; j++)
        {
                if (arr[j] < pivot)
                {
                        swap(arr[i], arr[j]);
                        i++;
                }
        }
        swap(arr[i], arr[high]);
        return i;
}

void quickSort(int arr[], int low, int high)
{
        if(low >= high){
```

```
            return;
        }
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
}

void printArray(int arr[], int size)
{
        int i;
        for (i = 0; i < size; i++)
                cout << arr[i] << " ";
        cout << endl;
}

int main()
{
        int arr[] = {1,5,3,7,2,8};
        int n = sizeof(arr) / sizeof(arr[0]);
        quickSort(arr, 0, n - 1);
        cout << "Sorted array: \n";
        printArray(arr, n);
        return 0;
}
```

Merge Sort

```
#include <bits/stdc++.h>
using namespace std;

void merge(int arr[], int low, int mid, int high){
    int n1 = mid-low +1;
    int n2 = high-mid;
    int L[n1], R[n2];
    for(int i=0; i<n1; i++){
        L[i] = arr[low+i];
    }
    for(int j=0; j<n2; j++){
        R[j] = arr[mid+1+j];
    }
    int i=0, j=0, k=low;
    while(i<n1 && j<n2){
        if(L[i]<=R[j]){
```

```
            arr[k++] = L[i++];
        }
        else{
            arr[k++] = R[j++];
        }
    }
    while(i<n1){
        arr[k++] = L[i++];
    }
    while(j<n2){
        arr[k++] = R[j++];
    }
}

void mergeSort(int arr[], int low, int high){
    if(low>=high){
        return;
    }
    int mid = low + (high - low)/2;
    mergeSort(arr, low, mid);
    mergeSort(arr, mid+1, high);
    merge(arr, low, mid, high);
}

void print(int arr[], int n){
    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
}

int main(){
    int arr[] = {3,7,1,6,2,8};
    int n = sizeof(arr)/sizeof(arr[0]);
    int low = 0;
    int high = n-1;
    mergeSort(arr, low, high);
    print(arr, n);
    return 0;
}
```

K-th element of two sorted Arrays

```
#include <bits/stdc++.h>
```

```
using namespace std;

void kthSorted(int arr1[], int arr2[], int m, int n, int key){
    int i=0, j=0, k=0;
    int arr3[m+n];
    while(i<m && j<n){
        if(arr1[i]<arr2[j]){
            arr3[k++] = arr1[i++];
        }
        else{
            arr3[k++] = arr2[j++];
        }
    }
    while(i<m){
        arr3[k++] = arr1[i++];
    }
    while(j<n){
        arr3[i++] = arr2[j++];
    }
    for(int i=0; i<k; i++){
        cout<<arr3[i]<<" ";
    }
    cout<<endl;
    cout<<"Kth Element: "<<arr3[key-1];
}

int main(){
    int arr1[] = {3,7,9};
    int arr2[] = {1,2,6,8};
    int key = 3;
    int m = sizeof(arr1)/sizeof(arr1[0]);
    int n = sizeof(arr2)/sizeof(arr2[0]);
    kthSorted(arr1, arr2, m, n, key);
    return 0;
}
```

Parenthesis Checker

```
#include <bits/stdc++.h>
using namespace std;

bool balancePar(string str){
    stack<char> st;
    char x;
```

```
    int n = str.length();
    for(int i=0; i<n; i++){
        if(str[i] == '{' || str[i] == '(' || str[i] == '['){
            st.push(str[i]);
        }
        if(st.empty()){
            return false;
        }
        if(str[i] == '}'){
            x = st.top();
            st.pop();
            if(x == '(' || x == '['){
                return false;
            }
        }
        if(str[i] == ')'){
            x = st.top();
            st.pop();
            if(x == '{' || x == '['){
                return false;
            }
        }
        if(str[i] == ']'){
            x = st.top();
            st.pop();
            if(x == '{' || x == '('){
                return false;
            }
        }
    }
    return (st.empty());
}

int main(){
    string str = "{()}]";
    if (balancePar(str))
        cout << "Balanced";
    else
        cout << "Not Balanced";
    return 0;
}
```

Next larger element

```
#include <bits/stdc++.h>
using namespace std;

void nextGreater(int arr[], int n){
    stack<int> s;
    s.push(arr[0]);
    for(int i=1; i<n; i++){
        if(s.empty()){
            s.push(arr[i]);
        }
        while(!s.empty() && s.top() < arr[i]){
            cout<<s.top()<<"->"<<arr[i]<<endl;
            s.pop();
        }
        s.push(arr[i]);
    }
    while(!s.empty()){
        cout<<s.top()<<"-->"<<-1<<endl;
        s.pop();
    }
}

int main(){
    int arr[] = {2,3,5,4,1};
    int n = sizeof(arr)/sizeof(arr[0]);
    nextGreater(arr, n);
    return 0;
}
```

Queue using two Stacks

```
#include <bits/stdc++.h>
using namespace std;

stack<int> s1;
stack<int> s2;

void enqueue(int x){
    while(!s1.empty()){
        s2.push(s1.top());
        s1.pop();
    }
    s2.push(x);
    while(!s2.empty()){
```

```
        s1.push(s2.top());
        s2.pop();
    }
}

void dequeue(){
    if(s1.empty()){
        cout<<"No element";
    }
    s1.pop();
}

void print(){
    stack<int> temp = s1;
    while(!temp.empty()){
        cout<<temp.top()<<"->";
        temp.pop();
    }
    cout<<"END"<<endl;
}

int main(){
    enqueue(10);
    enqueue(11);
    enqueue(12);
    dequeue();
    enqueue(13);
    enqueue(14);
    dequeue();
    print();
    return 0;
}
```

Stack using two queues

```
#include <bits/stdc++.h>
using namespace std;

queue<int> q1;
queue<int> q2;

void push(int x){
    q2.push(x);
    while(!q1.empty()){
```

```
        q2.push(q1.front());
        q1.pop();
    }
    queue<int> q = q1;
    q1 = q2;
    q2 = q;
}

void pop(){
    if(q1.empty()){
        cout<<"No Elements";
    }
    q1.pop();
}

void print(){
    queue<int> temp = q1;
    while(!temp.empty()){
        cout<<temp.front()<<"->";
        temp.pop();
    }
    cout<<"END"<<endl;
}

int main(){
    push(10);
    push(11);
    push(12);
    pop();
    push(13);
    push(14);
    pop();
    print();
    return 0;
}
```

Get minimum element from stack

```
#include <bits/stdc++.h>
using namespace std;

int minEle = 0;
stack<int> s;
```

```
void push(int x){
    if(s.empty()){
        minEle = x;
        s.push(x);
    }
    if(minEle > x){
        s.push(2*x - minEle);
        minEle = x;
    }
    s.push(x);
}

void pop(){
    if(s.empty()){
        cout<<"Empty";
    }
    int t = s.top();
    s.pop();
    if(t < minEle){
        minEle = 2*minEle - t;
    }
}

int main(){
    push(10);
    cout<<minEle<<endl;
    push(2);
    push(3);
    cout<<minEle<<endl;
    pop();
    push(49);
    pop();
    push(1);
    cout<<minEle<<endl;
    return 0;
}
```

LRU Cache

```
#include <bits/stdc++.h>
using namespace std;

void lru(int pages[], int n, int capacity){
    unordered_set<int> s;
```

```
    unordered_map<int,int> mp;
    int page_fault = 0;
    for(int i=0; i<n; i++){
        if(s.size()<capacity){
            if(s.find(pages[i]) == s.end()){
                s.insert(pages[i]);
                page_fault++;
            }
            mp[pages[i]] = i;
        }
        else{
            int lru = INT_MAX;
            int val;
            if(s.find(pages[i]) == s.end()){
                for(auto it = s.begin(); it != s.end(); it++){
                    if(mp[*it] < lru){
                        lru = mp[*it];
                        val = *it;
                    }
                }
                s.erase(val);
                s.insert(pages[i]);
                page_fault++;
            }
            mp[pages[i]] = i;
        }
    }
    cout<<page_fault;
}

int main(){
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int n = sizeof(pages)/sizeof(pages[0]);
    int capacity = 4;
    lru(pages, n, capacity);
    return 0;
}

#include <bits/stdc++.h>
using namespace std;

void refer(unordered_map<int, list<int>::iterator>ma, list<int>&dq, int csize, int x){
        if (ma.find(x) == ma.end()) {
                if (dq.size() == csize) {
```

```
                int last = dq.back();
                dq.pop_back();
                ma.erase(last);
            }
        }
        else{
            dq.erase(ma[x]);
        }
        dq.push_front(x);
        ma[x] = dq.begin();
}

void display(list<int>dq){
        for (auto it = dq.begin(); it != dq.end(); it++)
            cout << (*it) << " ";
        cout << endl;
}

int main(){
        int csize = 4;
        unordered_map<int, list<int>::iterator> ma;
        list<int>dq;
        refer(ma, dq, csize, 1);
        refer(ma, dq, csize, 2);
        refer(ma, dq, csize, 3);
        refer(ma, dq, csize, 1);
        refer(ma, dq, csize, 4);
        refer(ma, dq, csize, 5);
        display(dq);
        return 0;
}
```

First non-repeating character in a stream
Rotten Oranges

```
#include <bits/stdc++.h>
using namespace std;
#define R 3
#define C 5

bool isSafe(int i, int j){
    if(i>=0 && i<R && j >=0 && j<C){
        return true;
```

```
    }
    return false;
}

int rotOranges(int v[R][C]){
    int no = 2;
    bool changed = false;
    while(true){
        for(int i=0; i<R; i++){
            for(int j=0; j<C; j++){
                if(v[i][j] == no){
                    if(isSafe(i+1, j) && v[i+1][j]){
                        v[i+1][j] = v[i][j] + 1;
                        changed = true;
                    }
                    if(isSafe(i, j+1) && v[i][j+1]){
                        v[i][j+1] = v[i][j] + 1;
                        changed = true;
                    }
                    if(isSafe(i-1, j) && v[i-1][j]){
                        v[i-1][j] = v[i][j] + 1;
                        changed = true;
                    }
                    if(isSafe(i, j-1) && v[i][j-1]){
                        v[i][j-1] = v[i][j] + 1;
                        changed = true;
                    }
                }
            }
        }
        if(!changed){
            break;
        }
        changed = false;
        no++;
    }
    for(int i=0; i<R; i++){
        for(int j=0; j<C; j++){
            if(v[i][j] == 1){
                return -1;
            }
        }
    }
    return no-2;
```

```
}

int main(){
    int v[R][C] = {{ 2, 1, 0, 2, 1 },{ 1, 0, 1, 2, 1 },{ 1, 0, 0, 2, 1 }};
    cout << "Max time incurred: " << rotOranges(v);
    return 0;
}
```

Maximum of all subarrays of size k

Expression Evaluation

```
#include <bits/stdc++.h>
using namespace std;

void evaluateExp(string str, int n){
    stack<int> st;
    for(int i=n-1; i>=0; i--){
        if(isdigit(str[i])){
            st.push(str[i] - '0');
        }
        else{
            int val1 = st.top();
            st.pop();
            int val2 = st.top();
            st.pop();
            if(str[i] == '+'){
                st.push(val1+val2);
            }
            if(str[i] == '-'){
                st.push(val1-val2);
            }
            if(str[i] == '*'){
                st.push(val1*val2);
            }
            if(str[i] == '/'){
                st.push(val1/val2);
            }
        }
    }
    int res = st.top();
    cout<<res;
    return;
```

```
}

int main(){
    string str = "+9*26";
    int n = str.length();
    evaluateExp(str, n);
    return 0;
}
```

Relative Sorting
Sorting Elements of an Array by Frequency
Largest subarray with 0 sum
Common elements
Find all four sum numbers
Swapping pairs make sum equal
Count distinct elements in every window
Array Pair Sum Divisibility Problem
Longest consecutive subsequence
Array Subset of another array
Find all pairs with a given sum

```
#include <bits/stdc++.h>
using namespace std;

void findPair(int arr[], int n, int k){
    map<int,int> mp;
    for(int i=0; i<n; i++){
        int rem = k - arr[i];
        if(mp.find(rem) != mp.end()){
            cout<<"Pair: "<<rem<<","<<arr[i]<<endl;
        }
        mp[arr[i]]++;
    }
    return;
}

int main(){
    int arr[] = {6,7,3,4,8,6};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 12;
```

```
    findPair(arr, n, k);
    return 0;
}
```

Find first repeated character

```
#include <bits/stdc++.h>
using namespace std;

void firstRepeat(string str, int n){
    unordered_map<char,int> mp;
    for(int i=0; i<n; i++){
        if(mp.find(str[i]) != mp.end()){
            cout<<"First repeated character: "<<str[i]<<endl;
            return;
        }
        mp[str[i]]++;
    }
}

int main(){
    string str = "checker";
    int n = str.length();
    firstRepeat(str, n);
    return 0;
}
```

Zero Sum Subarrays

```
#include <bits/stdc++.h>
using namespace std;

bool subarrayZero(int arr[], int n){
    unordered_set<int> st;
    int sum = 0;
    for(int i=0; i<n; i++){
        sum += arr[i];
        if(sum == 0 || st.find(sum) != st.end()){
            return true;
        }
        st.insert(sum);
    }
    return false;
```

```
}

int main(){
    int arr[] = {1,3,-1,2,1,-4};
    int n = sizeof(arr)/sizeof(arr[0]);
    (subarrayZero(arr, n)) ? cout<<"True" : cout<<"False";
    return 0;
}
```

Minimum indexed character

```
#include <bits/stdc++.h>
using namespace std;

void checkMinIndex(string str1, string str2){
    int l1 = str1.length();
    int l2 = str2.length();
    map<char,int> mp;
    for(int i=0; i<l2; i++){
        mp[str2[i]]++;
    }
    for(int i=0; i<l1; i++){
        if(mp.find(str1[i]) != mp.end()){
            cout<<str1[i];
            return;
        }
    }
    cout<<"Not found"<<endl;
}

int main(){
    string str1 = "checkstring";
    string str2 = "semi";
    checkMinIndex(str1, str2);
    return 0;
}
```

Check if two arrays are equal or not

```
#include <bits/stdc++.h>
using namespace std;

bool checkCommon(int arr1[], int arr2[], int l1, int l2){
```

```
    unordered_map<int,int> mp;
    for(int i=0; i<l1; i++){
        mp[arr1[i]]++;
    }
    for(int i=0; i<l2; i++){
        if(mp.find(arr2[i]) == mp.end()){
            return false;
        }
    }
    return true;
}

int main(){
    int arr1[] = {3,4,6,2,1};
    int arr2[] = {4,6,2,1,3};
    int l1 = sizeof(arr1)/sizeof(arr1[0]);
    int l2 = sizeof(arr2)/sizeof(arr2[0]);
    if(checkCommon(arr1, arr2, l1, l2)){
        cout<<"True";
    }
    else{
        cout<<"False";
    }
    return 0;
}
```

Uncommon characters

```
#include <bits/stdc++.h>
using namespace std;

void checkUncommon(string str1, string str2){
    int m = str1.length();
    int n = str2.length();
    map<int,char> mp1;
    map<int,char> mp2;
    for(int i=0; i<m; i++){
        mp1[str1[i]]++;
    }
    for(int i=0; i<n; i++){
        mp2[str2[i]]++;
    }
    for(int i=0; i<n; i++){
        if(mp1.find(str2[i]) == mp1.end()){
```

```
            cout<<str2[i]<<" ";
        }
    }
    for(int i=0; i<m; i++){
        if(mp2.find(str1[i]) == mp2.end()){
            cout<<str1[i]<<" ";
        }
    }
    cout<<endl;
}

int main(){
    string str1 = "character";
    string str2 = "alphabet";
    checkUncommon(str1, str2);
    return 0;
}
```

Smallest window in a string containing characters of another string

First element to occur k times

```
#include <bits/stdc++.h>
using namespace std;

void firstKfreq(int arr[], int n, int k){
    unordered_map<int,int> mp;
    for(int i=0; i<n; i++){
        mp[arr[i]]++;
    }
    for(int i=0; i<n; i++){
        if(mp[arr[i]] == k){
            cout<<arr[i];
            return;
        }
    }
    cout<<"No Element Found";
    return;
}

int main(){
    int arr[] = {2,4,5,2,3,7,8,3,4,5,5};
```

```
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    firstKfreq(arr, n, k);
    return 0;
}
```

Check if frequencies can be equal

Print Left View of Binary Tree

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* right;
    Node* left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = NULL;
    new_node->left = NULL;
    return new_node;
}

void leftView(Node* root){
    if(!root){
        return;
    }
    queue<Node*> q;
    q.push(root);
```

```
    while(!q.empty()){
        int n = q.size();
        for(int i=0; i<n; i++){
            Node* temp = q.front();
            q.pop();
            if(i==0){
                cout<<temp->data<<" ";
            }
            if(temp->left){
                q.push(temp->left);
            }
            if(temp->right){
                q.push(temp->right);
            }
        }
    }
}

int main(){
    Node* root = newNode(10);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(7);
    root->left->right = newNode(8);
    root->right->right = newNode(15);
    root->right->left = newNode(12);
    root->right->right->left = newNode(14);
    leftView(root);
    return 0;
}
```

Check for BST

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node *right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
```

```
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

bool isBSTUtil(Node* root, int min, int max){
    if(!root){
        return true;
    }
    if(root->data < min || root->data > max){
        return false;
    }
    return isBSTUtil(root->left, min, root->data-1) && isBSTUtil(root->right, root->data+1, max);
}

bool isBST(Node* root){
    return isBSTUtil(root, INT_MIN, INT_MAX);
}

int main(){
    Node *root = newNode(4);
    root->left = newNode(2);
    root->right = newNode(5);
    root->left->left = newNode(10);
    root->left->right = newNode(3);
    isBST(root) ? cout<<"Yes" : cout<<"No";
    return 0;
}
```

Print Bottom View of Binary Tree

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node *right, *left;
    int hd;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
```

```
    new_node->hd = 0;
    new_node->left = new_node->right = NULL;
    return new_node;
}

void bottomView(Node* root){
    if(!root){
        return;
    }
    map<int,int> mp;
    int hd = 0;
    root->hd = hd;
    queue<Node*> q;
    q.push(root);
    while(!q.empty()){
        Node* temp = q.front();
        q.pop();
        hd = temp->hd;
        mp[hd] = temp->data;
        if(temp->left){
            temp->left->hd = hd-1;
            q.push(temp->left);
        }
        if(temp->right){
            temp->right->hd = hd+1;
            q.push(temp->right);
        }
    }
    for(auto it = mp.begin(); it != mp.end(); it++){
        cout<<it->second<<" ";
    }
}

int main(){
    Node* root = newNode(20);
    root->left = newNode(8);
    root->right = newNode(22);
    root->left->left = newNode(5);
    root->left->right = newNode(3);
    root->right->left = newNode(4);
    root->right->right = newNode(25);
    root->left->right->left = newNode(10);
    root->left->right->right = newNode(14);
    bottomView(root);
```

```
    return 0;
}
```

Print a Binary Tree in Vertical Order

```
#include <bits/stdc++.h>
using namespace std;

map<int, vector<int>> mp;

class Node{
    public:
    int data;
    Node *right, *left;
    int hd;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->hd = 0;
    new_node->left = new_node->right = NULL;
    return new_node;
}

void getVerticalOrder(Node* root, int hd){
    if(!root){
        return;
    }
    mp[hd].push_back(root->data);
    getVerticalOrder(root->left, hd-1);
    getVerticalOrder(root->right, hd+1);
}

void verticalOrder(Node* root){
    int hd = 0;
    getVerticalOrder(root, hd);
    for(auto it = mp.begin(); it != mp.end(); it++){
        for(int i=0; i<it->second.size(); i++){
            cout<<it->second[i]<<" ";
        }
        cout<<endl;
    }
}
```

```
int main(){
    Node *root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    root->right->left->right = newNode(8);
    root->right->right->right = newNode(9);
    verticalOrder(root);
    return 0;
}
```

Level order traversal in spiral form

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node *right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

void printSpiral(Node* root){
    if(!root){
        return;
    }
    stack<Node*> s1;
    stack<Node*> s2;
    s1.push(root);
    while(!s1.empty() || !s2.empty()){
        while(!s1.empty()){
            Node* temp = s1.top();
            s1.pop();
```

```
            cout<<temp->data<<" ";
            if(temp->right){
                s2.push(temp->right);
            }
            if(temp->left){
                s2.push(temp->left);
            }
        }
        while(!s2.empty()){
            Node* temp = s2.top();
            s2.pop();
            cout<<temp->data<<" ";
            if(temp->left){
                s1.push(temp->left);
            }
            if(temp->right){
                s1.push(temp->right);
            }
        }
    }
}

int main(){
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(7);
    root->left->right = newNode(6);
    root->right->left = newNode(5);
    root->right->right = newNode(4);
    printSpiral(root);
    return 0;
}
```

Connect Nodes at Same Level

Lowest Common Ancestor in a BT

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
```

```
    int data;
    Node *left, *right;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->left = new_node->right = NULL;
    return new_node;
}

Node* findLCA(Node* root, int a, int b){
    if(!root){
        return NULL;
    }
    if(root->data == a || root->data == b){
        return root;
    }
    Node* left_lca = findLCA(root->left, a, b);
    Node* right_lca = findLCA(root->right, a, b);
    if(left_lca && right_lca){
        return root;
    }
    return (left_lca) ? left_lca : right_lca;
}

int main(){
    Node* root = newNode(10);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    Node* temp = findLCA(root, 4, 5);
    cout <<temp->data;
    return 0;
}
```

Lowest Common Ancestor in a BST

```
#include <bits/stdc++.h>
using namespace std;

class Node{
```

```
    public:
    int data;
    Node* right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

Node* lca(Node* root, int a, int b){
    if(!root){
        return NULL;
    }
    if(root->data < a && root->data < b){
        return lca(root->right, a, b);
    }
    if(root->data > a && root->data > b){
        return lca(root->left, a, b);
    }
    else{
        return root;
    }
}

int main(){
    Node *root = newNode(20);
    root->left = newNode(8);
    root->right = newNode(22);
    root->left->left = newNode(4);
    root->left->right = newNode(12);
    root->left->right->left = newNode(10);
    root->left->right->right = newNode(14);
    int a = 10;
    int b = 22;
    Node* temp = lca(root, a, b);
    cout<<temp->data<<endl;
    return 0;
}
```

Convert a given Binary Tree to Doubly Linked List

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node *right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

void BTtoDLL(Node* root, Node** head){
    if(!root){
        return;
    }
    static Node* prev = NULL;
    BTtoDLL(root->left, head);
    if(prev == NULL){
        *head = root;
    }
    else{
        prev->right = root;
        root->left = prev;
    }
    prev = root;
    BTtoDLL(root->right, head);
}

void printList(Node *temp)
{
    while (temp!=NULL)
    {
        cout << temp->data << " ";
        temp = temp->right;
    }
}

int main(){
```

```
    Node* root        = newNode(10);
    root->left        = newNode(12);
    root->right       = newNode(15);
    root->left->left  = newNode(25);
    root->left->right = newNode(30);
    root->right->left = newNode(36);
    Node* head = NULL;
    BTtoDLL(root, &head);
    printList(head);
    return 0;
}
```

Write Code to Determine if Two Trees are Identical or Not

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node *right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

bool isIdentical(Node* root1, Node* root2){
    if(!root1 && !root2){
        return true;
    }
    if(!root1 || !root2){
        return false;
    }
    return ((root1->data == root2->data) && (isIdentical(root1->right, root2->right) &&
(isIdentical(root1->left, root2->left));
}

int main(){
    Node* root1 = newNode(10);
    root1->left = newNode(6);
```

```
    root1->right = newNode(12);
    root1->left->left = newNode(4);
    root1->right->right = newNode(15);
    Node* root2 = newNode(10);
    root2->left = newNode(6);
    root2->right = newNode(12);
    root2->left->left = newNode(4);
    root2->right->right = newNode(15);
    isIdentical(root1, root2) ? cout<<"Yes" : cout<<"No";
    return 0;
}
```

Given a binary tree, check whether it is a mirror of itself

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node *right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

bool isMirrorUtil(Node* root1, Node* root2){
    if(!root1 && !root2){
        return true;
    }
    return (root1 && root2) && (root1->data == root2->data) && isMirrorUtil(root1->left,
root2->right) && isMirrorUtil(root1->right, root2->left);
}

bool isMirror(Node* root){
    return isMirrorUtil(root, root);
}

int main(){
    Node* root = newNode(1);
```

```
    root->left = newNode(2);
    root->right = newNode(2);
    root->left->left = newNode(3);
    root->left->right = newNode(4);
    root->right->left = newNode(4);
    root->right->right = newNode(3);
    isMirror(root) ? cout<<"Yes" : cout<<"No";
    return 0;
}
```

Height of Binary Tree

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node *right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

int height(Node* root){
    if(!root){
        return 0;
    }
    return 1+max(height(root->left), height(root->right));
}

int main(){
    Node* root = newNode(10);
    root->left = newNode(6);
    root->right = newNode(8);
    root->left->right = newNode(4);
    root->left->left = newNode(2);
    root->right->left = newNode(12);
    cout<<height(root);
```

```
    return 0;
}
```

Maximum Path Sum

Diameter of a Binary Tree

```
#include <bits/stdc++.h>
using namespace std;

class node {
        int data;
        struct node *left, *right;
};


int diameter(struct node* tree){
        if (tree == NULL)
                return 0;
        int lheight = height(tree->left);
        int rheight = height(tree->right);
        int ldiameter = diameter(tree->left);
        int rdiameter = diameter(tree->right);
        return max(lheight + rheight + 1, max(ldiameter, rdiameter));
}

int height(struct node* node){
        if (node == NULL)
                return 0;
        return 1 + max(height(node->left), height(node->right));
}

class node* newNode(int data){
        struct node* node = (struct node*)malloc(sizeof(struct node));
        node->data = data;
        node->left = NULL;
        node->right = NULL;
        return (node);
}

int main(){
        Node* root = newNode(1);
```

```
        root->left = newNode(2);
        root->right = newNode(3);
        root->left->left = newNode(4);
        root->left->right = newNode(5);
        cout<<diameter(root);
        return 0;
}
```

Number of leaf nodes

```
#include <bits/stdc++.h>
using namespace std;

int count = 0;

class Node{
    public:
    int data;
    Node *right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

void noofLeafNodes(Node* root, int &count){
    if(!root){
        return;
    }
    if(!root->right && !root->left){
        count++;
    }
    noofLeafNodes(root->left, count);
    noofLeafNodes(root->right, count);
}

int main(){
    Node* root = newNode(10);
    root->left = newNode(2);
    root->right = newNode(3);
```

```
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    int count = 0;
    noofLeafNodes(root, count);
    cout<<count;
    return 0;
}
```

Check if given Binary Tree is Height Balanced or Not

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node *right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

int height(Node* root){
    if(!root){
        return 0;
    }
    return 1+max(height(root->right), height(root->left));
}

bool isHeightBalanced(Node* root){
    if(!root){
        return true;
    }
    int lh = height(root->left);
    int rh = height(root->right);
    return(abs(lh-rh)<=1 && isHeightBalanced(root->right) && isHeightBalanced(root->left);
}

int main(){
    Node* root = newNode(10);
```

```
    root->left = newNode(2);
    root->right = newNode(3);
    root->right->right = newNode(6);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    isHeightBalanced(root) ? cout<<"Yes" : cout<<"No";
    return 0;
}
```

Most frequent word in an array of strings

```
#include <bits/stdc++.h>
using namespace std;

void mostFreq(string arr[], int n){
    map<string,int> mp;
    for(int i=0; i<n; i++){
        mp[arr[i]]++;
    }
    int res = 0;
    for(auto it = mp.begin(); it != mp.end(); it++){
        res = max(res, it->second);
    }
    cout<<res;
}

int main(){
    string arr[] = {"one","two","one","three"};
    int n = sizeof(arr)/sizeof(arr[0]);
    mostFreq(arr, n);
    return 0;
}
```

CamelCase Pattern Matching

String Ignorance

Smallest window in a string containing all the characters of another string

Design a tiny URL or URL shortener

Permutations of a given string

Non Repeating Character
Check if strings are rotations of each other or not
Save Ironman
Repeated Character
Remove common characters and concatenate
Geek and its Colored Strings
Second most repeated string in a sequence

Activity Selection

```
#include <bits/stdc++.h>
using namespace std;

void activitySelection(int s[], int f[], int n){
    int i=0;
    cout<<i<<" ";
    for(int j=1; j<n; j++){
        if(s[j]>=f[i]){
            cout<<j<<" ";
            i = j;
        }
    }
    return;
}

int main(){
    int s[] =  {1, 3, 0, 5, 8, 5};
    int f[] =  {2, 4, 6, 7, 9, 9};
    int n = sizeof(s)/sizeof(s[0]);
    activitySelection(s, f, n);
    return 0;
}
```

N meetings in one room

```
#include <bits/stdc++.h>
using namespace std;
```

```
bool cmp(pair<int,int> a, pair<int,int> b){
    return a.second < b.second;
}

void maxMeetings(int s[], int f[], int n){
    vector<pair<int,int>> v;
    for(int i=0; i<n; i++){
        v.push_back(make_pair(s[i],f[i]));
    }
    sort(v.begin(), v.end(), cmp);
    int i=0;
    cout<<i<<" ";
    for(int j=1; j<n; j++){
        if(v[j].first >= v[i].second){
            cout<<j<<" ";
        }
    }
    return;

}

int main(){
    int s[] = { 0, 1, 5, 3, 8, 5 };
    int f[] = { 6, 2, 7, 4, 9, 9 };
    int n = sizeof(s)/sizeof(s[0]);
    maxMeetings(s, f, n);
    return 0;
}
```

Coin Piles

```
#include <bits/stdc++.h>
using namespace std;

void coinPiles(int arr[], int n, int k){
    int minEle = INT_MAX;
    for(int i=0; i<n; i++){
        if(arr[i]<minEle){
            minEle = arr[i];
        }
    }
    int res = 0;
    for(int i=0; i<n; i++){
        int diff = arr[i] - minEle;
```

```
        if (diff > k){
            res += (diff - k);
        }
    }
    cout<<minEle<<endl;
    cout<<res;
    return;
}

int main(){
    int arr[] = { 1, 5, 1, 2, 5, 1 };
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 3;
    coinPiles(arr, n, k);
    return 0;
}
```

Maximize Toys

```
#include <bits/stdc++.h>
using namespace std;

void maxToys(int arr[], int n, int k){
    int cnt = 0;
    int sum = 0;
    sort(arr, arr+n);
    for(int i=0; i<n; i++){
        sum += arr[i];
        if(sum<=k){
            cnt++;
        }
    }
    cout<<cnt;
}

int main(){
    int arr[] = { 1, 12, 5, 111, 200, 1000, 10 };
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 50;
    maxToys(arr, n, k);
    return 0;
}
```

Page Faults in LRU

```
#include <bits/stdc++.h>
using namespace std;

void lru(int pages[], int n, int capacity){
    unordered_set<int> s;
    unordered_map<int,int> mp;
    int page_fault = 0;
    for(int i=0; i<n; i++){
        if(s.size()<capacity){
            if(s.find(pages[i]) == s.end()){
                s.insert(pages[i]);
                page_fault++;
            }
            mp[pages[i]] = i;
        }
        else{
            int lru = INT_MAX;
            int val;
            if(s.find(pages[i]) == s.end()){
                for(auto it = s.begin(); it != s.end(); it++){
                    if(mp[*it] < lru){
                        lru = mp[*it];
                        val = *it;
                    }
                }
                s.erase(val);
                s.insert(pages[i]);
                page_fault++;
            }
            mp[pages[i]] = i;
        }
    }
    cout<<page_fault;
}

int main(){
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int n = sizeof(pages)/sizeof(pages[0]);
    int capacity = 4;
    lru(pages, n, capacity);
    return 0;
}
```

Largest number possible

```
#include <bits/stdc++.h>
using namespace std;

void maxNumber(int s, int m){
    if(s == 0){
        cout<<0;
    }
    int res[m];
    for(int i=0; i<m; i++){
        if(s>=9){
            res[i] = 9;
            s -= 9;
        }
        else if(s<9){
            res[i] = s;
            s = 0;
        }
    }
    for(int i=0; i<m; i++){
        cout<<res[i];
    }
}

int main(){
    int s = 20, m = 3;
    maxNumber(s, m);
    return 0;
}
```

Minimize the heights

2D Array Search

```
#include <bits/stdc++.h>
using namespace std;
#define M 3
#define N 4

bool searchRow(int arr[], int k){
    int low = 0, high = N - 1;
```

```
    while(low <= high){
        int mid = low + (high - low)/2;
        if(k == arr[mid]){
            return true;
        }
        if(k > arr[mid]){
            low = mid + 1;
        }
        else{
            high = mid - 1;
        }
    }
    return false;
}

bool searchMatrix(int matrix[M][N], int k){
    int low = 0, high = M-1;
    while(low <= high){
        int mid = low + (high - low)/2;
        if(k >= matrix[mid][0] && k <= matrix[mid][N-1]){
            return searchRow(matrix[mid], k);
        }
        if(k < matrix[mid][0]){
            high = mid - 1;
        }
        else{
            low = mid + 1;
        }
    }
    return false;
}

int main(){
    int matrix[M][N] = { { 1, 3, 5, 7 },
                         { 10, 11, 16, 20 },
                         { 23, 30, 34, 50 } };
    int K = 8;
    if (searchMatrix(matrix, K))
        cout << "Found" << endl;
    else
        cout << "Not found" << endl;
    return 0;
}
```

Minimize the sum of product

```
#include <bits/stdc++.h>
using namespace std;

void minSumProduct(int arr1[], int arr2[], int n){
    sort(arr1, arr1+n);
    sort(arr2, arr2+n);
    int res = 0;
    for(int i=0; i<n; i++){
        res += arr1[i]*arr2[n-i-1];
    }
    cout<<res;
}

int main(){
    int arr1[] = { 3, 1, 1 };
    int arr2[] = { 6, 5, 4 };
    int n = sizeof(arr1)/sizeof(arr1[0]);
    minSumProduct(arr1, arr2, n);
    return 0;
}
```

Huffman Decoding
Minimum Spanning Tree

Shop in Candy Store

```
#include <bits/stdc++.h>
using namespace std;

void maxCandies(int arr[], int n, int k){
    sort(arr, arr+n);
    int index = 0;
    int res = 0;
    for(int i=n-1; i>=index; i--){
        res += arr[i];
        index += k;
    }
    cout<<res<<endl;
}
```

```
void minCandies(int arr[], int n, int k){
    sort(arr, arr+n);
    int res = 0;
    for(int i=0; i<n; i++){
        res += arr[i];
        n -= k;
    }
    cout<<res<<endl;
}

int main(){
    int arr[] = { 3, 2, 1, 4 };
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 2;
    maxCandies(arr, n, k);
    minCandies(arr, n, k);
    return 0;
}
```

Find first set bit

```
#include <bits/stdc++.h>
using namespace std;

void firstSet(int n){
    int position = 1;
    while(!(n & 1)){
        n = n>>1;
        position++;
    }
    cout<<position;
}

int main(){
    int n = 10;
    firstSet(n);
    return 0;
}
```

Rightmost different bit

Check whether K-th bit is set or not

```
#include <bits/stdc++.h>
using namespace std;

int kthSet(int n, int k){
    return ((n>>(k-1)) & 1);
}

int main(){
    int n = 75, k = 4;
    kthSet(n, k) ? cout<<"Yes" : cout<<"No";
    return 0;
}
```

Toggle bits given range
Set kth bit

```
#include <bits/stdc++.h>
using namespace std;

void setKth(int n, int k){
    cout<< ((1<<(k-1)) | n);
}

int main(){
    int n = 15;
    int k = 3;
    setKth(n, k);
    return 0;
}
```

Power of 2

```
#include <bits/stdc++.h>
using namespace std;

bool powerof2(int n){
    if(n == 0){
        return false;
    }
    return !(n&(n-1));
}

int main(){
```

```
    int n = 16;
    powerof2(n) ? cout<<"Yes" : cout<<"No";
    return 0;
}
```

Bit Difference
Rotate Bits

```
#include <bits/stdc++.h>
using namespace std;

void leftrotateBits(int n, int k){
    cout<< ((n<<k) | (n>>(32-k)))<<endl;
}

void rightrotateBits(int n, int k){
    cout<< ((n>>k) | (n<<(32-k)))<<endl;
}

int main(){
    int n = 16;
    int k = 2;
    leftrotateBits(n, k);
    rightrotateBits(n, k);
    return 0;
}
```

Swap all odd and even bits

Count total set bits

```
#include <bits/stdc++.h>
using namespace std;

void countSet(int n){
    int cnt = 0;
    while(n){
        if(n & 1){
            cnt++;
        }
        n = n>>1;
```

```
    }
    cout<<cnt;
}

int main(){
    int n = 15;
    countSet(n);
    return 0;
}
```

Longest Consecutive 1’s

```
#include <bits/stdc++.h>
using namespace std;

void longestConsecutive1(int n){
    int cnt = 0;
    while(n){
        n = (n & n<<1);
        cnt++;
    }
    cout<<cnt;
}

int main(){
    int n = 15;
    longestConsecutive1(n);
    return 0;
}
```

Sparse Number
Alone in a couple

```
#include <bits/stdc++.h>
using namespace std;

void singleElement(int arr[], int n){
    int res = 0;
    for(int i=0; i<n; i++){
        res = res^arr[i];
    }
    cout<<res;
}
```

```
int main(){
    int arr[] = { 1, 2, 3, 2, 1 };
    int n = sizeof(arr)/sizeof(arr[0]);
    singleElement(arr, n);
    return 0;
}
```

Maximum subset XOR

Find Missing And Repeating

```
#include <bits/stdc++.h>
using namespace std;

void missAndRepeat(int arr[], int n){
    unordered_map<int,bool> mp;
    for(int i=0; i<n; i++){
        if(mp.find(arr[i]) == mp.end()){
            mp[arr[i]] = true;
        }
        else{
            cout<<arr[i]<<" ";
        }
    }
    cout<<endl;
    for(int i=1; i<n; i++){
        if(mp.find(i) == mp.end()){
            cout<<i<<" ";
        }
    }
    cout<<endl;
}

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

Maximum Index

Consecutive 1's not allowed

```
#include <bits/stdc++.h>
using namespace std;

void consecutiveOne(int n){
    int k = n+2;
    int dp[k+1];
    dp[0] = 0;
    dp[1] = 1;
    for(int i=2; i<=k; i++){
        dp[i] = dp[i-1] + dp[i-2];
    }
    cout<<dp[k];
}

int main(){
    int n = 3;
    consecutiveOne(n);
    return 0;
}
```

Majority Element

```
#include <bits/stdc++.h>
using namespace std;

void majorityEle(int arr[], int n){
    map<int,int> mp;
    for(int i=0; i<n; i++){
        mp[arr[i]]++;
    }
    for(auto it: mp){
        if(it.second > n/2){
            cout<<it.first;
            return;
        }
    }
    cout<<"No major element";
    return;
}
```

```
int main(){
    int arr[] = { 1, 3, 4, 5, 3, 3, 3};
    int n = sizeof(arr)/sizeof(arr[0]);
    majorityEle(arr, n);
    return 0;
}
```

Two numbers with sum closest to zero

Nuts and Bolts Problem

Boolean Matrix Problem

Smallest Positive missing number

Jumping Caterpillars

Anagram Pairs

```
#include <bits/stdc++.h>
using namespace std;
#define N 256

bool validAnagram(string str1, string str2){
    int arr[N] = {0};
    for (int i = 0; str1[i] && str2[i];  i++){
        arr[str1[i]]++;
        arr[str2[i]]--;
    }
    for(int i=0; i<N; i++){
        if(arr[i]){
            return false;
        }
    }
    return true;
}

void checkAnagram(string arr[], int n){
    for(int i=0; i<n-1; i++){
        for(int j=i+1; j<n; j++){
            if(validAnagram(arr[i],arr[j])){
                cout<<arr[i]<<" : "<<arr[j]<<endl;
            }
```

```
        }
    }
}

int main(){
    string arr[] = {"geeksquiz", "geeksforgeeks", "abcd",
                    "forgeeksgeeks", "zuiqkeegs"};
    int n = sizeof(arr)/sizeof(arr[0]);
    checkAnagram(arr, n);
    return 0;
}
```

Find median in a stream
Heap Sort

Operations on Binary Min Heap

```
#include <bits/stdc++.h>
using namespace std;

int* harr;
int hsize;
```

```
int left(int i){
    return (2*i + 1);
}

int right(int i){
    return (2*i + 2);
}

void minHeapify(int i){
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if(l<hsize && harr[l]<harr[i]){
        smallest = l;
    }
    if(r<hsize && harr[i]<harr[r]){
        smallest = r;
    }
    if(smallest != i){
        swap(harr[i], harr[smallest]);
        minHeapify(smallest);
    }
}

void minHeap(int arr[], int n){
    harr = arr;
    hsize = n;
    int i = (hsize - 1)/2;
    while(i){
        minHeapify(i);
        i--;
    }
}

int extractMin(){
    if(hsize == 0){
        return INT_MAX;
    }
    int root = harr[0];
    if(hsize > 1){
        harr[0] = harr[hsize - 1];
        minHeapify(0);
    }
    hsize--;
```

```
    return root;
}

void kthSmallest(int arr[], int n, int k){
    minHeap(arr, n);
    for(int i=0; i<n-1; i++){
        extractMin();
    }
    cout<<harr[0]<<endl;
}

int main(){
    int arr[] = {2,5,1,9,6};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 2;
    kthSmallest(arr, n, k);
    return 0;
}
```

Rearrange characters
Kth largest element in a stream

```
#include <bits/stdc++.h>
using namespace std;

int* harr;
int hsize;

int left(int i){
    return (2*i + 1);
}

int right(int i){
    return (2*i + 2);
}

void minHeapify(int i){
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if(l<hsize && harr[l]<harr[i]){
        smallest = l;
    }
```

```
    if(r<hsize && harr[i]<harr[r]){
        smallest = r;
    }
    if(smallest != i){
        swap(harr[i], harr[smallest]);
        minHeapify(smallest);
    }
}

void minHeap(int arr[], int n){
    harr = arr;
    hsize = n;
    int i = (hsize - 1)/2;
    while(i){
        minHeapify(i);
        i--;
    }
}

int extractMin(){
    if(hsize == 0){
        return INT_MAX;
    }
    int root = harr[0];
    if(hsize > 1){
        harr[0] = harr[hsize - 1];
        minHeapify(0);
    }
    hsize--;
    return root;
}

void kthSmallest(int arr[], int n, int k){
    minHeap(arr, n);
    for(int i=0; i<n-1; i++){
        extractMin();
    }
    cout<<harr[0]<<endl;
}

int main(){
    int arr[] = {2,5,1,9,6};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 2;
```

```
    kthSmallest(arr, n, k);
    return 0;
}
```

Merge K sorted linked lists

```
#include <bits/stdc++.h>
using namespace std;

class Node {
    public:
        int data;
        Node* next;
};

Node* newNode(int data){
        Node* new_node = new Node();
        new_node->data = data;
        new_node->next = NULL;
        return new_node;
}

class compare{
    public:
        bool operator()(Node* a, Node* b){
                return(a->data > b->data);
        }
};

Node* mergeKSortedLists(Node* arr[], int k){
        priority_queue<Node*, vector<Node*>, compare> pq;
        for(int i = 0; i < k; i++){
            if(arr[i] != NULL){
                pq.push(arr[i]);
            }
        }
        Node *dummy = newNode(0);
        Node *last = dummy;
        while(!pq.empty()){
                Node* curr = pq.top();
                pq.pop();
                last->next = curr;
                last = last->next;
                if(curr->next != NULL){
```

```
                pq.push(curr->next);
            }
        }
        return dummy->next;
}

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

int main(){
        int k = 3;
        Node* arr[k];
        arr[0] = newNode(1);
        arr[0]->next = newNode(3);
        arr[0]->next->next = newNode(5);
        arr[0]->next->next->next = newNode(7);
        arr[1] = newNode(2);
        arr[1]->next = newNode(4);
        arr[1]->next->next = newNode(6);
        arr[1]->next->next->next = newNode(8);
        arr[2] = newNode(0);
        arr[2]->next = newNode(9);
        arr[2]->next->next = newNode(10);
        arr[2]->next->next->next = newNode(11);
        Node* head = mergeKSortedLists(arr, k);
        printList(head);
        return 0;
}
```

Kth smallest element in a stream

```
#include <bits/stdc++.h>
using namespace std;

int* harr;
int hsize;

int left(int i){
    return (2*i + 1);
}
```

```
int right(int i){
    return (2*i + 2);
}

void minHeapify(int i){
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if(l<hsize && harr[l]<harr[i]){
        smallest = l;
    }
    if(r<hsize && harr[i]<harr[r]){
        smallest = r;
    }
    if(smallest != i){
        swap(harr[i], harr[smallest]);
        minHeapify(smallest);
    }
}

void minHeap(int arr[], int n){
    harr = arr;
    hsize = n;
    int i = (hsize - 1)/2;
    while(i){
        minHeapify(i);
        i--;
    }
}

int extractMin(){
    if(hsize == 0){
        return INT_MAX;
    }
    int root = harr[0];
    if(hsize > 1){
        harr[0] = harr[hsize - 1];
        minHeapify(0);
    }
    hsize--;
    return root;
}
```

```
void kthSmallest(int arr[], int n, int k){
    minHeap(arr, n);
    for(int i=0; i<n-1; i++){
        extractMin();
    }
    cout<<harr[0]<<endl;
}

int main(){
    int arr[] = {2,5,1,9,6};
    int n = sizeof(arr)/sizeof(arr[0]);
    int k = 2;
    kthSmallest(arr, n, k);
    return 0;
}
```

DFS

```
#include <bits/stdc++.h>
using namespace std;

void addEdge(vector<vector<int>> &adj, int s, int t){
    adj[s].push_back(t);
    adj[t].push_back(s);
}

void DFShelp(vector<vector<int>> &adj, vector<bool> &visited, int s){
    visited[s] = true;
    cout<<s<<" ";
    for(int i: adj[s]){x
        if(visited[i] == false){
            DFShelp(adj, visited, i);
        }
    }
    return;
}
```

```
void DFS(vector<vector<int>> &adj){
    vector<bool> visited(adj.size(),false);
    for(int i=0; i<adj.size(); i++){
        if(visited[i] == false){
            DFShelp(adj, visited, i);
        }
    }
    return;
}


int main(){
    int V = 6;
    vector<vector<int>> adj(V);
    vector<vector<int>> edges = {{1,2},{2,0},{2,3},{3,4}};
    for(auto &e: edges){
        addEdge(adj, e[0], e[1]);
    }
    cout<<"DFS: "<<" ";
    DFS(adj);
    return 0;
}
```

Other than must do coding

Search an element in sorted and rotated array

```
#include <bits/stdc++.h>
using namespace std;

void searchEle(int arr[], int l, int h, int key){
    if(l>h){
        return;
```

```
    }
    int mid = l + (h - l)/2;
    if(arr[mid] == key){
        cout<<"Found";
        return;
    }
    if(arr[l] <= arr[mid]){
        if(arr[l] <= key && key <= arr[mid]){
            return searchEle(arr, l, mid-1, key);
        }
        else{
            return searchEle(arr, l, mid+1, key);
        }
    }
    if(arr[mid] <= arr[h]){
        if(arr[mid] <= key && key <= arr[h]){
            return searchEle(arr, mid+1, h, key);
        }
        else{
            return searchEle(arr, l, mid-1, key);
        }
    }
    cout<<"Not Found";
}

int main(){
    int arr[] = {5,6,7,1,2,3,4,0};
    int n = sizeof(arr)/sizeof(arr[0]);
    int l = 0;
    int h = n-1;
    int key = 10;
    searchEle(arr, l, h, key);
    return 0;
}
```

Boundary Traversal

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
```

```
    Node *right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

void boundaryLeft(Node* root){
    if(!root){
        return;
    }
    if(root->left){
        cout<<root->data<<" ";
        boundaryLeft(root->left);
    }
    else if(root->right){
        cout<<root->data<<" ";
        boundaryLeft(root->right);
    }
}

void boundaryRight(Node* root){
    if(!root){
        return;
    }
    if(root->right){
        cout<<root->data<<" ";
        boundaryLeft(root->right);
    }
    else if(root->left){
        cout<<root->data<<" ";
        boundaryLeft(root->left);
    }
}

void leaves(Node* root){
    if(!root){
        return;
    }
    leaves(root->left);
    if(!root->left && !root->right){
```

```
        cout<<root->data<<" ";
    }
    leaves(root->right);
}

void boundaryTraversal(Node* root){
    if(!root){
        return;
    }
    cout<<root->data<<" ";
    boundaryLeft(root->left);
    leaves(root->left);
    leaves(root->right);
    boundaryRight(root->right);
}

int main(){
    Node* root = newNode(20);
    root->left = newNode(8);
    root->left->left = newNode(4);
    root->left->right = newNode(12);
    root->left->right->left = newNode(10);
    root->left->right->right = newNode(14);
    root->right = newNode(22);
    root->right->right = newNode(25);
    boundaryTraversal(root);
    return 0;
}
```

Sorting Algorithms

```
#include <bits/stdc++.h>
using namespace std;

void merge(int arr[], int low, int mid, int high){
    int n1 = mid - low + 1;
    int n2 = high - mid;
    int L[n1];
    int R[n2];
    for(int i=0; i<n1; i++){
        L[i] = arr[i+low];
    }
    for(int j=0; j<n2; j++){
```

```
        R[j] = arr[j+mid+1];
    }
    int i=0, j=0;
    int k = low;
    while(i<n1 && j<n2){
        if(L[i] < R[j]){
            arr[k++] = L[i++];
        }
        else{
            arr[k++] = R[j++];
        }
    }
    while(i<n1){
        arr[k++] = L[i++];
    }
    while(j<n2){
        arr[k++] = R[j++];
    }
}

void mergeSort(int arr[], int low, int high){
    if(low>=high){
        return;
    }
    int mid = low + (high - low)/2;
    mergeSort(arr, low, mid);
    mergeSort(arr, mid+1, high);
    merge(arr, low, mid, high);
}

void print(int arr[], int n){
    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
}

void Sort(int arr[], int n){
    for(int i=0; i<n; i++){
        for(int j=0; j<i; j++){
            if(arr[j]>arr[i]){
                swap(arr[i], arr[j]);
            }
        }
```

```
    }
}

void selectionSort(int arr[], int n){
    int min_idx = 0;
    for(int i=0; i<n-1; i++){
        min_idx = i;
        for(int j=i+1; j<n; j++){
            if(arr[j]<arr[min_idx]){
                min_idx = j;
            }
        }
        swap(arr[i], arr[min_idx]);
    }
}

void bubbleSort(int arr[], int n){
    for(int i=0; i<n; i++){
        for(int j=0; j<n-i-1; j++){
            if(arr[j]>arr[j+1]){
                swap(arr[j], arr[j+1]);
            }
        }
    }
}

int partition(int arr[], int low, int high){
    int pivot = arr[high];
    int i = low;
    for(int j=low; j<high; j++){
        if(arr[j] < pivot){
            swap(arr[i],arr[j]);
            i++;
        }
    }
    swap(arr[i],arr[high]);
    return i;
}

void quickSort(int arr[], int low, int high){
    if(low >= high){
        return;
    }
    int pi = partition(arr, low, high);
```

```
    quickSort(arr, low, pi-1);
    quickSort(arr, pi+1, high);
}

int main(){
    int arr[] = {3,6,4,8,1,2};
    int n = sizeof(arr)/sizeof(arr[0]);
    int low = 0;
    int high = n-1;
    //print(arr, n);
    //mergeSort(arr, low, high);
    //print(arr, n);
    //Sort(arr, n);
    //print(arr, n);
    //selectionSort(arr, n);
    //print(arr, n);
    //bubbleSort(arr, n);
    //print(arr, n);
    //quickSort(arr, low, high);
    //print(arr, n);
    return 0;
}

#include <bits/stdc++.h>
using namespace std;

void priorityShow(int arr[], int n){
    priority_queue<int, vector<int>, greater<int>> pq;
    for(int i=0; i<n; i++){
        pq.push(arr[i]);
    }
    while(!pq.empty()){
        cout<<pq.top()<<" ";
        pq.pop();
    }
}

int main(){
    int arr[] = {3,2,9,1,6};
    int n = sizeof(arr)/sizeof(arr[0]);
    priorityShow(arr, n);
    return 0;
}
```

Alternate linked list

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node* next;
};

void push(Node** head_ref, int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void alternateList(Node* head1, Node* head2, Node** newhead2){
    Node *q1 = head1, *q2 = head2;
    Node* q1_next;
    Node* q2_next;
    while(q1 && q2){
        q1_next = q1->next;
        q2_next = q2->next;
        q1->next = q2;
        q2->next = q1_next;
        q1 = q1_next;
        q2 = q2_next;
    }
    (*newhead2) = q2;
}

void print(Node* head1){
    Node* temp = head1;
    while(temp){
        cout<<temp->data<<"->";
        temp = temp->next;
    }
    cout<<"END"<<endl;
}

void printNew(Node* head1, Node* newhead2){
```

```
    Node* temp = head1;
    Node* temp2 = newhead2;
    while(temp){
        cout<<temp->data<<"->";
        temp = temp->next;
    }
    while(temp2){
        cout<<temp2->data<<"->";
        temp2 = temp2->next;
    }
    cout<<"END"<<endl;
}

int main(){
    Node* head1 = NULL;
    Node* head2 = NULL;
    Node* newhead2 = NULL;
    push(&head1, 50);
    push(&head1, 30);
    push(&head1, 10);
    push(&head2, 80);
    push(&head2, 60);
    push(&head2, 40);
    push(&head2, 20);
    print(head1);
    print(head2);
    alternateList(head1, head2, &newhead2);
    printNew(head1, newhead2);
    return 0;
}
```

String

```
#include <bits/stdc++.h>
using namespace std;

void URLShortner(long int n){
    char map[] =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    string result;
    while(n){
        result.push_back(map[n%62]);
        n = n/62;
```

```
    }
    reverse(result.begin(), result.end());
    cout<<result<<endl;
}

void permutation(string str, int low, int high){
    if(low == high){
        cout<<str<<" ";
    }
    else{
        for(int i=low; i<=high; i++){
            swap(str[low], str[i]);
            permutation(str, low+1, high);
            swap(str[low],str[i]);
        }
    }
}

void secondWord(string arr[], int n){
    unordered_map<string,int> mp;
    for(int i=0; i<n; i++){
        mp[arr[i]]++;
    }
    int first_max = 1;
    int second_max = 1;
    for(auto it = mp.begin(); it != mp.end(); it++){
        if(it->second > first_max){
            second_max = first_max;
            first_max = it->second;
        }
    }
    string result;
    for(auto it = mp.begin(); it != mp.end(); it++){
        if(it->second == second_max){
            result = it->first;
        }
    }
    cout<<result<<endl;
}

void camelCase(string str){
    const regex pattern("([a-z]+[0-9]+[A-Z]+[a-z])+");
    if(regex_match(str, pattern)){
        cout<<"Yes";
```

```
        return;
    }
    else{
        cout<<"No";
    }
}

void duplicates(string str, int n){
    unordered_set<char,int> st;
    string res = "";
    for(int i=0; i<n; i++){
        if(st.find(str[i]) == st.end()){
            res = res + str[i];
            st.insert(str[i]);
        }
    }
    cout<<res<<endl;
}

int main(){
    int n = 12345;
    URLShortner(n);
    string str = "abc";
    string str1 = "pranav09M";
    string arr[] = {"aaa","bbb","ccc"};
    int size = sizeof(arr)/sizeof(arr[0]);
    int low = 0;
    int high = str.length() - 1;
    permutation(str, low, high);
    cout<<endl;
    secondWord(arr, size);
    camelCase(str1);
    string str2 = "pranav";
    int n = str2.length();
    duplicates(str2, n);
    return 0;
}
```

Iterative Inorder

```
#include <bits/stdc++.h>
using namespace std;
```

```
class Node{
    public:
    int data;
    Node *right, *left;
};

Node* newNode(int new_key){
    Node* new_node = new Node();
    new_node->data = new_key;
    new_node->right = new_node->left = NULL;
    return new_node;
}

void inorder(Node* root){
    if(!root){
        return;
    }
    stack<Node*> s;
    Node* curr = root;
    while(curr || !s.empty()){
        while(curr){
            s.push(curr);
            curr = curr->left;
        }
        curr = s.top();
        s.pop();
        cout<<curr->data<<" ";
        curr = curr->right;
    }
}

int main(){
    Node* root = newNode(30);
    root->left = newNode(20);
    root->left->left = newNode(10);
    root->right = newNode(40);
    root->right->right = newNode(50);
    inorder(root);
    return 0;
}
```

OOPS Implementation

Class and Object

```
#include <bits/stdc++.h>
using namespace std;

class Animal{
    public:
    string name;
    string breed;
    void printBreed();
    void printName(){
        cout<<name<<endl;
    }
};

void Animal::printBreed(){
    cout<<breed<<endl;
}

int main(){
    Animal a1;
    a1.name = "Dog";
    a1.breed = "Lab";
    a1.printName();
    a1.printBreed();
    return 0;
}
```

Single-level Inheritance

```
#include <bits/stdc++.h>
using namespace std;

class Shape{
    public:
    void setWidth(int w){
        width = w;
    }
    void setHeight(int h){
        height = h;
```

```
    }
    void getArea();
    protected:
    int height;
    int width;
};

class Rectangle: public Shape{
    public:
    void getArea(){
        cout<<height*width<<endl;
    }
};

int main(){
    Rectangle rect;
    rect.setWidth(10);
    rect.setHeight(5);
    rect.getArea();
    return 0;
}
```

Multiple Inheritance

```
#include <bits/stdc++.h>
using namespace std;

class Shape{
    public:
    void setWidth(int w){
        width = w;
    }
    void setHeight(int h){
        height = h;
    }
    void getArea();
    protected:
    int height;
    int width;
};

class showDetails{
    public:
```

```
    void details(){
        cout<<"Rectangle is one of the shape";
    }
};

class Rectangle: public Shape, public showDetails{
    public:
    void getArea(){
        cout<<height*width<<endl;
    }
};

int main(){
    Rectangle rect;
    rect.setWidth(10);
    rect.setHeight(5);
    rect.getArea();
    rect.details();
    return 0;
}
```

Multi-level Inheritance

```
#include <bits/stdc++.h>
using namespace std;

class Shape{
    public:
    void setWidth(int w){
        width = w;
    }
    void setHeight(int h){
        height = h;
    }
    protected:
    int width;
    int height;
};

class Rectangle: public Shape{
    public:
    int area = 0;
```

```
    void getArea(){
        area = width*height;
        cout<<area<<endl;
    }
};

class Cost: public Rectangle{
    public:
    int cost = 0;
    void getCost(){
        cost = area*70;
        cout<<cost<<endl;
    }
};

int main(){
    Cost cst;
    cst.setWidth(10);
    cst.setHeight(5);
    cst.getArea();
    cst.getCost();
    return 0;
}
```

Function Overriding

```
#include <bits/stdc++.h>
using namespace std;

class Vehicle{
    public:
    void name(){
        cout<<"Hi from Vehicle"<<endl;
    }
};

class Car: public Vehicle{
    public:
    void name(){
        cout<<"Hi from Car"<<endl;
    }
};
```

```
int main(){
    Vehicle v;
    Car c;
    v.name();
    c.name();
    return 0;
}
```

Function Overloading

Exceptional Handling

```
#include <bits/stdc++.h>
using namespace std;

void errorCheck(){
    try{
        int x = 10;
        if(x < 0){
            throw(x);
        }
        cout<<"No exception: "<<x<<endl;
    }
    catch(int x){
        cout<<"Exception occured";
    }
}

int main(){
    errorCheck();
    return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;

class Node{
    public:
    int data;
    Node *right, *left;
```

```
};

Node* newNode(int new_key){
   Node* new_node = new Node();
   new_node->data = new_key;
   new_node->left = new_node->right = NULL;
   return new_node;
}

void printPreorder(Node* root){
   if(!root){
      return;
   }
   cout<<root->data<<" ";
   printPreorder(root->left);
   printPreorder(root->right);
   return;
}

void printInorder(Node* root){
   if(!root){
      return;
   }
   printInorder(root->left);
   cout<<root->data<<" ";
   printInorder(root->right);
   return;
}

void printPostorder(Node* root){
   if(!root){
      return;
   }
   printPostorder(root->left);
   printPostorder(root->right);
```

```
    cout<<root->data<<" ";
    return;
}


int main(){
    Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    //root->right->left = newNode(13);
    printPreorder(root);
    cout<<endl;
    printInorder(root);
    cout<<endl;
    printPostorder(root);
    return 0;
}
```

New Chapter 2024

Stack using Array

```
#include <iostream>
using namespace std;

int myStack[100];
int max = 100;
int top = -1;

void push(int val){
```

```
    if(top>=100){
        cout<<"Overflow"<<endl;
        return;
    }
    top++;
    myStack[top] = val;
}

void pop(){
    if(top <= -1){
        cout<<"Underflow"<<endl;
        return;
    }
    cout<<myStack[top]<<endl;
    top--;
}

void print() {
    for (int i = top; i >= 0; i--) {
        cout << myStack[i] << endl;
    }
}

int main(){
    push(10);
    push(20);
    push(30);
    push(40);
    pop();
    print();
    return 0;
}
```

Additional Solutions:
■Solutions for Remaining Problems:■■1. Chocolate Distribution Problem■```cpp■#include <bits/stdc