**Largest Rectangle area:**

int largest\_rectangle\_area(vector<int> &v){

int n = v.size(),max\_area = -1;

vector<int> left(n), right(n);

stack<int> stk;

for (int i = 0; i < n; i++){

if (stk.empty())

left[i] = 0, stk.push(i);

else{

while (!stk.empty() && v[stk.top()] >= v[i])

stk.pop();

if (stk.empty())

left[i] = 0;

else

left[i]=stk.top() + 1;

stk.push(i);

}

}

while (!stk.empty())

stk.pop();

for (int i = n - 1; i >= 0; i--){

if (stk.empty())

right[i] = n - 1, stk.push(i);

else{

while (!stk.empty() && v[stk.top()] >= v[i])

stk.pop();

if (stk.empty())

right[i] = n - 1;

else

right[i]=stk.top() - 1;

stk.push(i);

}

}

for (int i = 0; i < n; i++)

max\_area = max(max\_area, (v[i] \* (right[i] - left[i] + 1)));

return max\_area;

}

-------------------------------------

**SET BALANCING:**

void balance(multiset<int> right, multiset<int> &left){

while (true){

int st = right.size();

int sl = left.size();

if (st == sl || st == sl + 1)

break;

if (st < sl)

right.insert(\*left.begin()), left.erase(left.begin());

else

left.insert(\*right.rbegin()), right.erase(right.rbegin());

}

}

void insert\_in\_set(multiset<int> &right, multiset<int> &left, int value)

{

if (right.emptleft())

right.insert(value);

else

{

auto it = right.end();

it--;

if (value < \*it)

right.insert(value);

else

left.insert(value);

}

}

------------------------------------

**BINARY EXPONENTIATION:(a^b)**

int binaryExponentiation(int base, int power)////O(log2(power))

{

if (power == 0)

return 1;

int ans = binaryExponentiation(base, power / 2);

if (power % 2 == 1)

return (((ans \* ans) % mod) \* base) % mod;

return (ans \* ans) % mod;

}

**BINARY EXPONENTIATION:(a^b^c)**

int binaryExponetiation(int base, int power, int modulo)

{

int ans = 1;

while (power)

{

if (power % 2 == 1)

ans = (ans \* base) % modulo;

base = (base \* base) % modulo;

power /= 2;

}

return ans;

}

//it also works for a^b.

//this if faster then previous. Cz recursion need extra time.

//for bc we need to pass (mod-1) and for a^b^c we need to pass mod.

//mod=10^9+7

-------------------------------------

**DIVISOR COUNT:**  
int maxVal = 1e6 + 1;

vector<int> countDivisor(maxVal, 0);

void countingDivisor(){

for (int i = 1; i < maxVal; i++)

for(int j= i; j<maxVal; j += i)

countDivisor[j]++;

}

-------------------------------------

**MAX GCD OF an ARRAY:**

//Here we take input in the freq array

for (int i = 0; i < n; i++)

cin >> x, freq[x]++;

vector<int> multiples(maxVal, 0);

for (int i = 1; i < maxVal; i++)

for(int j = i; j < maxVal; j += i)

if (freq[j])

multiples[i] += freq[j];

int ans = 0;

for (int i = 0; i < maxVal; i++)

if (multiples[i] > 1)

ans = i;

-------------------------------------

**Matching string:**

**KMP ALGORITHM:**

//complexity O(n + m)

vector<int> createLPS(string pattern){

vector<int> lps(pattern.length());

int index = 0;

for(int i=1;i < pattern.length();){

if(pattern[index]== pattern[i]){

lps[i] = index + 1;

index++, i++;

}

else{

if (index != 0)

index = lps[index - 1];

else

lps[i] = index, i++;

}

}

return lps;

}

int kmp(string text, string pattern)

{

int cnt\_of\_match = 0;

vector<int> lps = createLPS(pattern);

debug(lps);

int i = 0, j = 0;

// i -> text, j -> pattern

while (i < text.length()){

if (text[i] == pattern[j])

i++, j++;

else{

if (j != 0)

j = lps[j - 1];

else

i++;

}

if (j == pattern.length()){

cnt\_of\_match++;

// the index where match found -> (i - pattern.length());

j = lps[j - 1];

}

}

return cnt\_of\_match;

}

-------------------------------------

**SEGMENT TREE:**

vector<int> v(2\*1e5 +5),seg(4\*1e5 + 5);

void build(int ti, int low, int high)

{

if (high == low){

seg[ti] = v[low];

return;

}

int mid = (low + high) / 2;

build(2 \* ti + 1, low, mid);

build(2 \* ti + 2, mid + 1, high);

seg[ti] = seg[2\*ti+1]+seg[ti\*2+2];

}

//tree left, tree right, query left, query right, index

int findValue(int tl, int tr, int ql, int qr, int ti)

{

if (tl > qr or tr < ql)

return 0;(sum, xor)

// return INT\_MAX;(min)

// return INT\_MIN;(max)

if (tl >= ql and tr <= qr)

return seg[ti];

int mid = (tl + tr) / 2;

int l = findValue(tl, mid, ql, qr, 2 \* ti + 1);

int r = findValue(mid + 1, tr, ql, qr, 2 \* ti + 2);

return l + r;(sum)

// return min(l,r);

// return max(l,r);

}

void update(int ti, int low, int high, int id, int val){

if (id > high or id < low)

return;

if (id == high and high == low){

seg[ti] = val;

return;

}

int mid = (low + high) / 2;

update(2 \* ti + 1, low, mid, id, val);

update(2 \* ti + 2, mid + 1, high, id, val);

seg[ti] = (seg[2 \* ti + 1] + seg[ti \* 2 + 2]);

}

----------------------------------

**1’s complement:**

int bitSize = log2(n) + 1;

int ans = (~((1 << bitSize) - 1)) ^ (~n);

cout << ans << endl;

------------------------------------

**BIPARTITE GRAPH:**

bool dfs(int start, int clr)

{

color[start]=clr,visited[start]= 1;

for (auto child : adj[start]){

if (!visited[child]){

if (!dfs(child, clr ^ 1))

return false;

}

else if (color[child] == color[start])

return false;

}

return true;

}

**----------------------------**

**Dijkstra:**

void dijkstra(int source, vector<int>& distance, vector<vector<pairi>>& adj, vector<int>& parent) {

set<pairi> st;

st.insert({0, source});

distance[source] = 0;

while (!st.empty()) {

auto top = \*(st.begin());

int nodeDis = top.first;

int topNode = top.second;

// dbg(topNode);

st.erase(st.begin());

for (auto child:adj[topNode]) {

int adjNode = child.first;

int adjWt = child.second;

if (nodeDis + adjWt < distance[adjNode]) {

auto record = st.find({distance[adjNode], adjNode});

if (record != st.end())

st.erase(record);

distance[adjNode] = nodeDis + adjWt;

st.insert({distance[adjNode], adjNode});

parent[adjNode] = topNode;

}

}

}

}

------------------------------------

**Prims:**

typedef pair<int,pair<int,int>> pairUV;

map<int, bool> visited;

map<int, vector<pair<int, int>>> adj;

void Prims() {

int sum = 0, c = 0;

vector<pairUV> ans;

priority\_queue<pairUV, vector<pairUV>, greater<pairUV>> pq;

pq.push({0, {1, -1}});

while (!pq.empty()) {

pairUV k = pq.top();

pq.pop();

int u = k.second.first;

int v = k.second.second;

int wt = k.first;

if (visited[u])

continue;

sum += wt;

visited[u] = 1;

if (v != -1)

ans.pb({wt, {u, v}});

for (auto it : adj[u]) {

int adjNode = it.first;

int adjwt = it.second;

if (!visited[adjNode])

pq.push({adjwt, {adjNode, u}});

}

}

}

------------------------------------

**Bellmon Ford:**

// vector<vector<pair<int,int>>> edges

// u {v, wt}

void bellmonFord(int n, int m, int src, vector<vector<pair<int, int>>> &edges) {

vector<int> dist(n + 1, 30000);

dist[src] = 0;

for (int step = 0; step < n; step++) {

for (int i = 1; i <= n; i++) {

for (auto it : edges[i]) {

int u = i;

int v = it.first;

int wt = it.second;

if (dist[u] != 30000 && ((dist[u] + wt) < dist[v])) {

dist[v] = dist[u] + wt;

}

}

}

}

for (int i = 1; i <= n; i++)

cout << dist[i] << " ";

cout << endl;

}------------------------------------