

1. **Graph Theory**
   1. **Dijkstra Algorithm**

**#define INF 1<<30 //2^30**

**Vector <int> g[MAX], cost[MAX];**

**int dis[MAX];**

**struct node {**

**int u ,cost;**

**node(int \_u, int \_cost) {**

**u = \_u , cost = \_cost;**

**}**

**bool operator < ( const node& p ) const{**

**return cost > p.cost;**

**}**

**};**

**void dijstkra(int n,int source) {**

**for(i = 0 to n) dis[i] = INF;**

**priority\_queue <node> Q; Q.push(node(source,0));**

**dis[source] = 0;**

**while(!Q.empty()) {**

**int u = Q.top().u; Q.pop();**

**for(i=0 to g[u].size()) {**

**int v=g[u][i];**

**if(dis[u] + cost[u][i] < dis[v]) {**

**dis[v] = dis[u] + cost[u][i];**

**Q.push (node(v, dis [v]));**

**}**

**}**

**}**

**}**

* 1. **Kruskal (MST)**

**struct exo {**

**int u, v, w;**

**exo(int \_u,int \_v,int \_w){**

**u=\_u,v=\_v,w=\_w;**

**}**

**bool operator<(const exo& p) const {**

**return w < p.w;**

**}**

**};**

**int Parent[MAXN];**

**vector <exo> G;**

**int Find(int r) {**

**return (Parent[r] == r) ? r : Find(Parent[r]);**

**}**

**int Kruskal(int n) {**

**sort(G.begin(), G.end());**

**for (int i=0 to n ) Parent[i] = i;**

**int cnt = 0, cost = 0;**

**for (int i = 0 to G.size() ) {**

**int u = Find(G[i].u);**

**int v = Find(G[i].v);**

**if (u != v) {**

**Parent[u] = v;**

**cnt++;**

**cost += G[i].w;**

**if (cnt == n - 1) break;**

**}**

**}**

**return cost;**

**}**

* 1. **Strongly Connected Component**

**vector <int> edges[N],sortedNode, rEdges[N];**

**bool vis [N];**

**int comp [N]; // component number of a node**

**int inDegree [N]; // in-degree of a component**

**void reset () {**

**for ( int i = 0; i < N; i++ ) {**

**edges [i].clear(); rEdges [i].clear();**

**vis[i]=inDegree[i]=0;**

**}**

**sortedNode.clear();**

**}**

**void dfs1 (int x) {**

**vis [x] = true;**

**for (int u = 0; u < edges [x].size(); u++ ) {**

**if(!vis[edges[x][u]]) dfs1(edges [x] [u]);**

**}**

**sortedNode.push\_back(x);**

**}**

**void dfs2 (int x, int c) {**

**vis [x] = false; comp [x] = c;**

**for(int u = 0; u < rEdges [x].size(); u++ ) {**

**if(vis[rEdges[x][u]]) dfs2(rEdges[x][u], c);**

**}**

**}**

**int main () {**

**reset ();**

**for ( int i = 0; i < numberOfEdges; i++ ) {**

**cin>>from>>to;**

**edges [from].push\_back(to);**

**rEdges [to].push\_back(from);**

**}**

**for ( int i = 0; i < numberOfNodes; i++ ) {**

**if ( !vis [i] ) dfs1 (i);**

**}**

**int componentId = 0;**

**for( i = sortedNode.size() - 1; i >= 0; i-- ) {**

**if ( vis [sortedNode [i]] )**

**dfs2(sortedNode [i], ++componentId);**

**}**

**// component indegree**

**for(i = 0; i < numberOfNodes; i++ ) {**

**for (j = 0; j < edges [i].size(); j++ ) {**

**if (comp [i] != comp [edges [i] [j]] )**

**inDegree [comp [edges [i] [j]]]++;**

**}**

**}**

**return 0;**

**}**

* 1. **Floyed Warshall**

**typedef vector < int > vi;**

**typedef vector < vi > vvi;**

**/\* mat[i][i] = 0, mat[i][j] = distance from i to**

**j, path[i][j] = i \*/**

**void APSP( vvi &mat, vvi &path ) {**

**int V = mat.size();**

**for( int via=0; via<V; via++ ) {**

**for( int from=0; from<V; from++ ) {**

**for( int to=0; to<V; to++ ) {**

**if( mat[from][via] + mat[ via ][ to ]**

**< mat[ from ][to]) {**

**mat[ from ][ to ] =**

**mat[ from ][ via ] + mat[ via ][ to];**

**path[from][ to ] = path[ via ][ to ];**

**}**

**}**

**}**

**}**

**}**

**/// prints the path from i to j**

**void print( int i, int j ) {**

**if( i != j ) {**

**print( i, path[i][j] );**

**}**

**cout << j << "\n";**

**}**

**/// check if negative cycle exists**

**bool negative\_cycle( vvi &mat ) {**

**APSP( mat );**

**return (mat[0][0] < 0);**

**}**

**void transtitive\_closure( vvi &mat ) {**

**int V = mat.size();**

**for( int via=0; via; via<V; via++ ) {**

**for( int from=0; from<V; from++ ) {**

**for( int to=0; to<V; to++ ) {**

**mat[ from ][ to ] |=**

**( mat[ from ][ via ] & mat[ via ][ to] );**

**}**

**}**

**}**

**}**

**/// finding a path between two nodes that maximizes the minimum cost**

**void mini\_max( vvi &mat ) {**

**int V = mat.size();**

**for( int via=0; via; via<V; via++ ) {**

**for( int from=0; from<V; from++ ) {**

**for( int to=0; to<V; to++ ) {**

**mat[ from ][ to ] = min( mat[ from][to ],**

**max( mat[ from ][ via ],mat[via][to] ));**

**}**

**}**

**}**

**}**

**/// finding a path between two nodes that minimizes the maximum cost**

**/// eg: max load a truck can carry from one node to another node where**

**/// the paths have weight limit**

**void maxi\_min( vvi &mat ) {**

**int V = mat.size();**

**for( int via=0; via; via<V; via++ ) {**

**for( int from=0; from<V; from++ ) {**

**for( int to=0; to<V; to++ ) {**

**mat[from][to] = max( mat[ from ][ to ],**

**min(mat[from][via], mat[via][to]) );**

**}**

**}**

**}**

**}**

* 1. **Bellman Ford**

**struct Edge {**

**int u, v, w;**

**};**

**vector < Edge > vs;**

**int n, dis[205];**

**void bell() {**

**dis[1] = 0;**

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

**for(int j = 0; j < vs.size(); j++) {**

**if(dis[vs[j].v] > dis[vs[j].u] + vs[j].w ) {**

**dis[vs[j].v] = dis[vs[j].u] + vs[j].w;**

**}**

**}**

**}**

**}**

**1.06 Topological Sort**

**int n;**

**vector<vector<int>> adj;**

**vector<bool> visited;**

**vector<int> ans;**

**void dfs(int v) {**

**visited[v] = true;**

**for (int u : adj[v]) {**

**if (!visited[u]) dfs(u);**

**}**

**ans.push\_back(v);**

**}**

**void topological\_sort() {**

**visited.assign(n, false);**

**ans.clear();**

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

**if (!visited[i]) dfs(i);**

**}**

**reverse(ans.begin(), ans.end());**

**}**

**///\*-----------Graph Moves--------------------**

**int fx[]={+1,-1,+0,+0}; //Four Side**

**int fy[]={+0,+0,+1,-1}; //Four Side**

**int fx[]={+0,+0,+1,-1,-1,+1,-1,+1};// Kings Move**

**int fy[]={-1,+1,+0,+0,+1,+1,-1,-1}; // Kings Move**

**int fx[]={-2,-2,-1,-1, 1, 1, 2, 2}; //Knights Move**

**int fy[]={-1, 1,-2, 2,-2, 2,-1, 1}; //Knights Move**

**int fx[]={+0,+0,+1,-1,-1,+1,-1,+1}; //Eight Side**

**int fy[]={-1,+1,+0,+0,+1,+1,-1,-1}; //Eight Side**

**///\*----------------------------------------**

**2 Data Structure**

**2.01 Segment Tree**

**int arr[mx],tree[mx \* 3];**

**void init(int node, int b, int e) {**

**if (b == e) {**

**tree[node] = arr[b]; return;**

**}**

**int Left = node << 1;**

**int Right = Left | 1;**

**int mid = (b + e) >> 1;**

**init(Left, b, mid);**

**init(Right, mid + 1, e);**

**tree[node] = tree[Left]+tree[Right];**

**}**

**int query(int node, int b, int e, int i, int j) {**

**if (i > e || j < b)**

**return 0;**

**if (b >= i && e <= j)**

**return tree[node];**

**int Left = node << 1;**

**int Right = Left | 1;**

**int mid = (b + e) >> 1;**

**int p1 = query(Left, b, mid, i, j);**

**int p2 = query(Right, mid + 1, e, i, j);**

**return p1+p2;**

**}**

**void update(int node, int b, int e, int i, int**

**newvalue) {**

**if (i > e || i < b) return;**

**if (b >= i && e <= i) {**

**tree[node] = newvalue; return;**

**}**

**int Left = node << 1;**

**int Right = Left | 1;**

**int mid = (b + e) >> 1;**

**update(Left, b, mid, i, newvalue);**

**update(Right, mid + 1, e, i, newvalue);**

**tree[node] = tree[Left] + tree[Right];**

**}**

**2.02 Segment Tree with Lazy**

**struct exo { int pro,sum; };**

**exo tree[mx \* 3];**

**int arr[mx];**

**void init(int node, int b, int e) {**

**if (b == e) {**

**tree[node].sum = arr[b]; return;**

**}**

**int Left = node << 1;**

**int Right = Left | 1;**

**int mid = (b + e) >> 1;**

**init(Left, b, mid);**

**init(Right, mid + 1, e);**

**tree[node].sum =**

**tree[Left].sum+tree[Right].sum;**

**}**

**int query(int node, int b, int e, int i, int j,int**

**carry=0) {**

**if (i > e || j < b) return 0;**

**if (b >= i && e <= j)**

**return tree[node].sum+carry\*(e-b+1);**

**int Left = node << 1;**

**int Right = Left | 1;**

**int mid = (b + e) >> 1;**

**int p1 = query(Left, b, mid, i,**

**j,carry+tree[node].pro);**

**int p2 = query(Right, mid + 1, e, i,**

**j,carry+tree[node].pro);**

**return p1 + p2;**

**}**

**void update(int node, int b, int e, int i,int j,**

**int newvalue) {**

**if (i > e || j < b) return;**

**if (b >= i && e <= j) {**

**tree[node].sum +=((e-b+1)\* newvalue);**

**tree[node].pro +=newvalue;**

**return;**

**}**

**int Left = node << 1;**

**int Right = Left | 1;**

**int mid = (b + e) >> 1;**

**update(Left, b, mid, i,j, newvalue);**

**update(Right, mid + 1, e, i,j, newvalue);**

**tree[node].sum = tree[Left].sum +**

**tree[Right].sum+(e-b+1)\*tree[node].pro;**

**}**

**2.03 Binary Indexed Tree (BIT) 1D**

**int ar[mx] ,tree[mx];**

**int query(int idx) {**

**int sum = 0;**

**while (idx > 0) {**

**sum += tree[idx];**

**idx -= (idx & -idx);**

**}**

**return sum;**

**}**

**void update(int idx, int val, int n) {**

**while (idx <= n) {**

**tree[idx] += val;**

**idx += (idx & -idx);**

**}**

**}**

**2.04 Binary Indexed Tree(BIT) 2D**

**using vi = vector < int >;**

**using vii = vector < vi >;**

**struct BIT\_2D {**

**int n; vii tree;**

**BIT\_2D () {}**

**BIT\_2D (int \_n):n( \_n),tree( \_n,vi( \_n, 0)) {}**

**~BIT\_2D () {}**

**void update\_y( int x, int y, int v ) {**

**for( ; y<n; y+=(y&-y) ) {**

**tree[x][y] += v;**

**}**

**}**

**void update( int x, int y, int v ) {**

**for( ; x<n; x+=(x&-x) ) {**

**update\_y( x, y, v );**

**}**

**}**

**int query\_y( int x, int y ) {**

**int ret = 0;**

**for( ; y; y-=(y&-y) ) {**

**ret += tree[x][y];**

**}**

**return ret;**

**}**

**int query( int x, int y ) {**

**int ret = 0;**

**for( ; x; x-=(x&-x) ) {**

**ret += query\_y( x, y );**

**}**

**return ret;**

**}**

**int query( int x1, int y1, int x2, int y2 ) {**

**return ( query( x2, y2 ) –**

**query( x2, y1-1 ) -**

**query( x1-1, y2 ) +**

**query( x1-1, y1-1 ) );**

**}**

**}**

**2.05 Merge Sort**

**void merge(int ar[],int temp[],int L,int mid,**

**int R) {**

**int i=L, j=mid, k=L;**

**while( (i<mid) && (j<=R)) {**

**if(ar[i]<=ar[j]) temp[k++]=ar[i++];**

**else temp[k++]=ar[j++];**

**}**

**while(i<mid) temp[k++]=ar[i++];**

**while(j<=R) temp[k++]=ar[j++];**

**while(L<=R) ar[L]=temp[L], L++;**

**}**

**void mergesort(int ar[],int temp[],int L,int R) {**

**if(L<R) {**

**int mid=L+(R-L)/2;**

**mergesort(ar,temp,L,mid);**

**mergesort(ar,temp,mid+1,R);**

**merge(ar,temp,L,mid+1,R);**

**}**

**}**

**2.06 STL order statistics tree**

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

**#include <ext/pb\_ds/assoc\_container.hpp>**

**#include <ext/pb\_ds/tree\_policy.hpp>**

**using namespace \_\_gnu\_pbds;**

**using namespace std;**

**typedef tree< int,null\_type,less<int>,rb\_tree\_tag,**

**tree\_order\_statistics\_node\_update>**

**ordered\_set;**

**int main() {**

**ordered\_set X;**

**X.insert(1); X.insert(2); X.insert(4);**

**X.insert(8); X.insert(16);**

**cout<<\*X.find\_by\_order(1)<<endl; // 2**

**cout<<\*X.find\_by\_order(4)<<endl; // 16**

**cout<<(end(X)==X.find\_by\_order(6))<<endl;**

**cout<<X.order\_of\_key(-5)<<endl; // 0**

**cout<<X.order\_of\_key(4)<<endl; // 2**

**cout<<X.order\_of\_key(400)<<endl; // 5**

**}**

**2.07 MO’s Algorithm**

**///Find sum in a Range**

**int k,n,q,ans[mx],ar[mx],L,R,sum;**

**struct Query{**

**int QueryNo,L,R;**

**bool operator < (const Query &other) const{**

**int block\_own= L/k;**

**int block\_other= other.L/k;**

**if(block\_own==block\_other)**

**return R<other.R;**

**return block\_own < block\_other;**

**}**

**}Q[mx];**

**void Add(int x) { sum+=ar[x]; }**

**void Remove(int x) { sum-=ar[x]; }**

**int main(){**

**cin>>n;**

**k=(int)ceil(sqrt(n\*1.0) );**

**for(int i=0; i<n; i++ )cin>>ar[i];**

**cin>>q;**

**for(int i=0; i<q; i++){**

**cin>>Q[i].L>>Q[i].R;**

**Q[i].L--; Q[i].R--;**

**Q[i].QueryNo=i;**

**}**

**sort(Q,Q+q);**

**int L=0,R=-1;**

**for(int i=0; i<q; i++){**

**while(R<Q[i].R) Add(++R);**

**while(L<Q[i].L) Remove(L++);**

**while(R>Q[i].R) Remove(R--);**

**while(L>Q[i].L) Add(--L);**

**ans[ Q[i].QueryNo ]= sum;**

**}**

**for(int i=0; i<q; i++) cout<<ans[i]<<endl;**

**}**

**//Distinct Value in Sub Array**

**int ans[mx],k,n,q,ar[mx],L,R,curDiff;**

**struct Query { //2.07// } query[mx];**

**int main() {**

**cin>>n;**

**k=(int)sqrt(n);**

**for(int i=0; i<n; i++) cin>>ar[i];**

**cin>>q;**

**for(int i=0; i<q; i++) {**

**cin>>query[i].L>>query[i].R);**

**query[i].L--; query[i].R--;**

**query[i].QueryNo=i;**

**}**

**sort(query,query+q);**

**vector<int> frequency(mx,0);**

**L=0,R=-1,curDiff=0;**

**for(int i=0; i<q; i++) {**

**while(L<query[i].L) {**

**frequency[ ar[L] ]--;**

**curDiff-=(frequency[ ar[L] ]==0);**

**L++;**

**}**

**while(R<query[i].R) {**

**R++;**

**frequency[ ar[R] ]++;**

**curDiff+=(frequency[ ar[R] ]==1);**

**}**

**while(L > query[i].L) {**

**L--;**

**frequency[ ar[L] ]++;**

**curDiff+=(frequency[ ar[L] ]==1);**

**}**

**while(R > query[i].R) {**

**frequency[ ar[R] ]--;**

**curDiff-=(frequency[ ar[R] ]==0);**

**R--;**

**}**

**ans[ query[i].QueryNo ]=curDiff;**

**}**

**for(int i=0; i<q; i++)cin>>ans[i]<endl;**

**}**

**2.08 Disjoint Set**

**void make\_set(int v) {**

**parent[v] = v; rank[v] = 0;**

**}**

**int find\_set(int v) {**

**if (v == parent[v]) return v;**

**return parent[v] = find\_set(parent[v]);**

**}**

**void union\_sets(int a, int b) {**

**a = find\_set(a); b = find\_set(b);**

**if (a != b) {**

**if (rank[a] < rank[b]) swap(a, b);**

**parent[b] = a;**

**if (rank[a] == rank[b]) rank[a]++;**

**}**

**}**

**2.09 Sparse Table**

**const int k = 16 , N = 1e5 ,ZERO = 1e9 + 1;**

**int table[N][k + 1] , Arr[N];**

**int main(){**

**int n, L, R, q;**

**cin >> n;**

**for(int i = 0; i < n; i++) cin >> Arr[i];**

**///Pre-compute sparse table**

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

**table[i][0] = Arr[i];**

**for(int j = 1; j <= k; j++){**

**for(int i = 0; i <= n - (1 << j); i++){**

**table[i][j] = min(table[i][j - 1],**

**table[i + (1 << (j - 1))][j - 1]);//For RMQ**

**table[i][j] = table[i][j - 1] +**

**table[i + (1 << (j - 1))][j - 1]; //For RSQ**

**table[i][j] = \_\_gcd(table[i][j - 1],**

**table[i + (1 << (j - 1))][j 1]); //For GCD**

**}**

**}**

**cin >> q;**

**for(int i = 0; i < q; i++){**

**cin >> L >> R;**

**int answer = ZERO;**

**for(int j = k; j >= 0; j--){**

**if(L + (1 << j) - 1 <= R){**

**answer =min(answer,table[L][j]); //For RMQ**

**answer = answer + table[L][j]; //For RSQ**

**answer=\_\_gcd(answer,table[L][j]);//For GCD**

**L += 1 << j;**

**}**

**}**

**}**

**cout << answer << endl;**

**return 0;**

**}**

**//Num of contiguous sub-arrays with GCD equal to 1**

**long long answer = 0;**

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

**int R = i, g = ZERO;**

**for(int j = k; j >= 0; j--) {**

**if(R + (1 << j) - 1 >= n) continue;**

**if(\_\_gcd(g, table[R][j]) > 1) {**

**g = \_\_gcd(g, table[R][j]);**

**R += 1 << j;**

**}**

**}**

**answer += n - R;**

**}**

**cout << answer << endl;**

**2.10 Merge Sort Tree  
vector <int> tree[5\*N]; int A[N];**

**Void build\_tree( int cur, int l, int r ) {**

**if( l==r ) {**

**tree[cur].push\_back( a[ l ] ); return ;**

**}**

**int mid = l+(r-l)/2;**

**build\_tree(2\*cur+1, l, mid );**

**build\_tree(2\*cur+2, mid+1, r );**

**tree[cur]=merge( tree[2\*cur+1], tree[2\*cur+2]);**

**}**

**int query(int cur,int l,int r,int x,int y,int k) {**

**if( r < x || l > y ) { return 0; }**

**if( x<=l && r<=y ) {**

**//Binary search over the current sorted**

**vector to find elements smaller than K**

**return upper\_bound(tree[cur].begin(),**

**tree[cur].end(),K)-tree[cur].begin();**

**}**

**int mid=l+(r-l)/2;**

**return query(2\*cur+1,l,mid,x,y,k)+**

**query(2\*cur+2,mid+1,r,x,y,k);**

**}**

**2.11 Find K-th number after sorting in ascending Order L to R**

**/\***

**- At first take input and store the input in a pair**

**array where first element of i\_th pair is the value**

**a\_i and second element of i\_th pair is i.**

**- Sort the pair array with with ascending order of a\_i**

**- Build a merge sort tree using the second element of**

**each pair of sorted pair array.**

**- Now for each query i,j,k at first check how many**

**number in range i to j inclusive are present in left**

**subtree of current node in merge sort tree. Let the**

**value is x. So if x<=k then it's sure that the answer**

**is in the left subtree. So we will go to left subtree**

**of current node with k. Otherwise we will go to right**

**subtree of current node with k-x;**

**- In this manner when we reach to a leaf node we can say**

**that this node contains the index of our answer.**

**\*/**

**#define segment\_tree int l=n\*2,r=l+1,mid=(b+e)/2**

**#define mx 100005**

**pii ara[mx]; int arr[mx];**

**vector<int>tree[3\*mx];**

**void init(int n, int b, int e) {**

**if(b==e) {**

**tree[n].push\_back(ara[b].sv);**

**return;**

**}**

**segment\_tree;**

**init(l,b,mid);**

**init(r,mid+1,e);**

**merge(all(tree[l]),all(tree[r]),**

**back\_inserter(tree[n]));**

**}**

**int query(int n, int b, int e, int i,int j,int k){**

**if(b==e) {**

**return tree[n].back();**

**}**

**segment\_tree;**

**int x = upper\_bound(all(tree[l]),j)-**

**lower\_bound(all(tree[l]),i);**

**if(x>=k) {**

**return query(l,b,mid,i,j,k);**

**} else**

**return query(r,mid+1,e,i,j,k-x);**

**}**

**int main() {**

**int n,m; cin>>n>>m;**

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

**cin>>ara[i].fv;**

**ara[i].sv=i;**

**arr[i]=ara[i].fv;**

**}**

**sort(ara+1,ara+n+1);**

**init(1,1,n);**

**while(m--) {**

**cin>>a>>b>>c;**

**int ans=query(1,1,n,a,b,c);**

**printf("%d\n",arr[ans]);**

**}**

**return 0;**

**}**

**2.12 Find sum of number sum(ai)<Cap from L to R**

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

**#include <ext/pb\_ds/assoc\_container.hpp>**

**#include <ext/pb\_ds/tree\_policy.hpp>**

**#include <ext/pb\_ds/detail/standard\_policies.hpp>**

**using namespace \_\_gnu\_pbds;**

**using namespace std;**

**#define mid (b+e)/2**

**#define ll long long**

**const int sz=100005,oo=1<<30;**

**vector<ll>mst[sz\*4] , sum[sz\*4];**

**ll ara[sz],n,q,k=1;**

**void combine(int node,int left,int right) {**

**int i=0,j=0;**

**for(;i<mst[left].size()&&j<mst[right].size(); ){**

**if(mst[left][i]<mst[right][j]) {**

**mst[node].push\_back(mst[left][i]); i++;**

**} else {**

**mst[node].push\_back(mst[right][j]); j++;**

**}**

**}**

**while(i<mst[left].size()) {**

**mst[node].push\_back(mst[left][i]);**

**i++;**

**}**

**while(j<mst[right].size()) {**

**mst[node].push\_back(mst[right][j]);**

**j++;**

**}**

**sum[node].push\_back(mst[node][0]);**

**for(int i=1; i<mst[node].size(); i++) {**

**sum[node].push\_back(**

**sum[node][i-1]+mst[node][i]);**

**}**

**}**

**void build(int node,int b,int e) {**

**if(b==e) {**

**mst[node].push\_back(ara[b]);**

**sum[node].push\_back(ara[b]);**

**return;**

**}**

**build(node\*2,b,mid);**

**build(node\*2+1,mid+1,e);**

**combine(node,node\*2,node\*2+1);**

**}**

**pair<ll,ll> query(int node,int b,int e,ll l,ll r,ll val) {**

**if(b>r || e<l) return {0,0};**

**if(b>=l && e<=r) {**

**int indx=upper\_bound(mst[node].begin(),**

**mst[node].end(),val)-mst[node].begin();**

**if(indx>0) {**

**return {sum[node][indx-1],indx};**

**}**

**return {0,0};**

**}**

**pair<ll,ll>q1=query(node\*2,b,mid,l,r,val);**

**pair<ll,ll>q2=query(node\*2+1,mid+1,e,l,r,val);**

**q1.first+=q2.first;**

**q1.second+=q2.second;**

**return q1;**

**}**

**int main() {**

**scanf("%lld%lld",&n,&q);**

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

**scanf("%lld",&ara[i]);**

**}**

**build(1,1,n);**

**ll l,r,x;**

**while(q--) {**

**scanf("%lld%lld%lld",&l,&r,&x);**

**pair<ll,ll>ans=query(1,1,n,l,r,x);**

**printf("%lld\n",ans.second\*x-ans.first);**

**}**

**}**

**3 Number Theory**

**3.01 Sieve of Eratosthenes**

**void seive() {**

**for(i=2; i<=mx; i++) flag[i]=1;**

**int val=sqrt(mx)+1;**

**for(i=2; i<val; i++){**

**if(flag[i]){**

**for(j=i; j\*i<=mx; j++) flag[i\*j]=0;**

**}**

**}**

**for(i=2; i<=mx; i++)**

**if(flag[i]) primes[index++]=i;**

**}**

**3.02 Bitwise Sieve**

**int primes[mx]; int N=mx;**

**int status[(mx/32)+2];**

**bool Check(int N,int pos) {**

**return (bool)(N & (1<<pos));**

**}**

**int Set(int N,int pos) {**

**return N=N | (1<<pos);**

**}**

**void sieve() {**

**int i, j, sqrtN,ind=0;**

**sqrtN = int( sqrt( N ) );**

**for( i = 3; i <= sqrtN; i += 2 ) {**

**if( Check(status[i>>5],i&31)==0) {**

**for( j = i\*i; j <= N; j += (i<<1) ) {**

**status[j>>5]=Set(status[j>>5],j & 31);**

**}**

**}**

**}**

**primes[ind++]=2;**

**for(i=3; i<=N; i+=2){**

**if( Check(status[i>>5],i&31)==0)**

**primes[ind++]=i;**

**}**

**}**

**3.03 Segmented Sieve**

**int arr[SIZE];**

**int segmentedSieve(int a,int b) {**

**if(a==1) a++;**

**int sqrtn=sqrt(b);**

**memset(arr,0,sizeof arr);**

**for(i=0;i<prime.size()&&prime[i]<=sqrtn; i++) {**

**int p=prime[i];**

**int j=p\*p;**

**if(j<a)**

**j=((a+p-1)/p)\*p;**

**for (; j<=b; j+=p) { arr[j-a]=1; }**

**}**

**int res=0;**

**for(int i=a; i<=b; i++) {**

**if(arr[i-a]==0) res++;**

**}**

**return res;**

**}**

**3.04 Number of Divisor & Sum of Divisor**

**int divisor(int N) {**

**int val=sqrt(N)+1,cnt,p;**

**int sumOfDiv=1 ,totalDivisor=1;**

**for(i=0; primes[i]<val; i++) {**

**if(N%primes[i]==0) {**

**cnt=0; //For Div Count**

**p=1; //For Sum of Div**

**while(N%primes[i]==0) {**

**N/=primes[i];**

**cnt++;**

**p\*=primes[i];**

**}**

**p\*=primes[i];**

**s=(p-1)/(primes[i]-1);**

**sumOfDiv\*=s;**

**totalDivisor\*=(cnt+1);**

**}**

**}**

**if(N>1){**

**p=N\*N;**

**s=(p-1)/(N-1);**

**sumOfDiv\*=s;**

**totalDivisor\*=2;**

**}**

**return sumOfDiv**

**return totalDivisor;**

**}**

**3.05 Totient Phi Function**

**long long totient(long long n) {**

**if (n == 1) return 0;**

**long long ans = n;**

**for (int i = 0;primes[i]\*primes[i]<= n; ++i) {**

**if ((n % primes[i]) == 0) {**

**while ((n % primes[i]) == 0)**

**n /= primes[i];**

**ans -= ans / primes[i];**

**}**

**}**

**if (n > 1) ans -= ans / n;**

**return ans;**

**}**

**3.06 Sieve Phi**

**void sievephi() {**

**mark[1] = 1;**

**for(ll i = 1; i < Mx; i++) {**

**phi[i] = i;**

**if(!(i & 1))**

**mark[i] = 1, phi[i] /= 2;**

**}**

**mark[2] = 0;**

**for(ll i = 3; i < Mx; i+=2) {**

**if(!mark[i]) {**

**phi[i] = phi[i] - 1;**

**for(ll j = 2 \* i; j < Mx; j += i) {**

**mark[j] = 1;**

**phi[j] /= i;**

**phi[j] \*= i - 1;**

**}**

**}**

**}**

**}**

**3.07 How Many Digits of X^Y**

**Digit= floor( log10(x^y))+1= floor(y\*log10(x))+1**

**3.08 How Many Digits Of N!**

**floor[log 10 (1) + log 10 (2) + ………+log 10 (N)]+1**

**3.09 Trailing Zeros of N!**

**int Trailing\_Zeros(int N) {**

**int sum=0;**

**while(N) {**

**sum+=N/5;**

**N=N/5;**

**}**

**return sum;**

**}**

**3.10 Big Mod**

**ll big\_mod( ll b, ll p ) {**

**ll ret = 1;**

**for( ; p; p >>= 1 ) {**

**if( p&1 ) ret = ( ret \* b ) % mod;**

**b = ( b \* b ) % mod;**

**}**

**return ret % mod;**

**}**

**3.11 Number Of Divisor [1 to N]**

**int D[1000010];**

**void DivisorGenerate() {**

**int i,j,val,N,M,count;**

**D[1]=1;**

**for(i=2; i<=1000000; i++) {**

**N=M=i; val=sqrt(N)+1;**

**for(j=0; primes[j]<val; j++) {**

**if(M%primes[j]==0) {**

**count=0;**

**while(M%primes[j]==0) {**

**M/=primes[j]; count++;**

**}**

**D[N]=(count+1)\*D[M];**

**break;**

**}**

**}**

**if(M==N) D[N]=2;**

**}**

**}**

**3.12 Catalan Number**

**unsigned long int catalan(unsigned int n) {**

**if (n <= 1) return 1;**

**unsigned long int res = 0;**

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

**res += catalan(i)\*catalan(n-i-1);**

**return res;**

**}**

**3.13 Number Of Trailing Zeros in N! on Base B**

**int factors\_in\_factorial(int N,int p) {**

**int sum=0;**

**while(N) {**

**sum+=N/p; N/=p;**

**}**

**return sum;**

**}**

**int Trailingzero\_Base\_B(int N,int B) {**

**int freq,power,ans=INF;**

**for(i=0; primes[i]<=B; i++) {**

**if(B%primes[i]==0) {**

**freq=0;**

**while(B%primes[i]==0) {**

**freq++; B/=primes[i];**

**}**

**power=factors\_in\_factorial(N,primes[i]);**

**ans=min(ans,power/freq);**

**}**

**}**

**return ans;**

**}**

**3.14 Pascal Triangle**

**int C[maxn + 5][maxn + 5];**

**int PascalTriangle() {**

**C[0][0] = 1;**

**for (int n = 1; n <= maxn; ++n) {**

**C[n][0] = C[n][n] = 1;**

**for (int k = 1; k < n; ++k) {**

**C[n][k] = C[n - 1][k - 1] + C[n - 1][k];**

**}**

**}**

**}**

**3.15 Cumulative Sum of Divisor**

**Cud(24)=2+3+4+6+8+12**

**long long csod(long long n) {**

**long long ans = 0;**

**for (long long i = 2; i \* i <= n; ++i) {**

**long long j = n / i;**

**ans += (i + j) \* (j - i + 1) / 2;**

**ans += i \* (j - i);**

**}**

**return ans;**

**}**

**3.16 Modular Inverse**

**int gcdExtended(int a, int b, int \*x, int \*y) {**

**if (a == 0) {**

**\*x = 0, \*y = 1;**

**return b;**

**}**

**int x1, y1;**

**int gcd = gcdExtended(b%a, a, &x1, &y1);**

**\*x = y1 - (b/a) \* x1;**

**\*y = x1;**

**return gcd;**

**}**

**int modInverse(int a, int m) {**

**int x, y;**

**int g = gcdExtended(a, m, &x, &y);**

**if (g != 1) return 0;**

**else {**

**return (x%m + m) % m;**

**}**

**}**

**3.17 Lucas Theorem (nCr % p)**

**int nCrModp(int n, int r, int p) {**

**int C[r+1]={0}; C[0] = 1;**

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

**for (int j = min(i, r); j > 0; j--)**

**C[j] = (C[j] + C[j-1])%p;**

**}**

**return C[r];**

**}**

**int nCrModpLucas(int n, int r, int p) {**

**if (r==0) return 1;**

**int ni = n%p, ri = r%p;**

**return**

**(nCrModpLucas(n/p, r/p, p)\***

**nCrModpDP(ni, ri, p)) % p;**

**}**

**3.18 Number Of relative primes in a given interval**

**int CoPrime (int n, int r) {**

**vector<int> p;**

**for (int i=2; i\*i<=n; ++i) {**

**if (n % i == 0) {**

**p.push\_back (i);**

**while (n % i == 0) n /= i;**

**}**

**}**

**if (n > 1) p.push\_back (n);**

**int sum = 0;**

**for (int msk=1; msk<(1<<p.size()); ++msk) {**

**int mult = 1,bits = 0;**

**for (int i=0; i<(int)p.size(); ++i) {**

**if (msk & (1<<i)) {**

**++bits; mult \*= p[i];**

**}**

**}**

**int cur = r / mult;**

**if (bits % 2 == 1) sum += cur;**

**else sum -= cur;**

**}**

**return r - sum;**

**}**

**3.19 Last Non Zero Digit Of N!**

**int** dig[] = {**1**, **1**, **2**, **6**, **4**, **2**, **2**, **4**, **2**, **8**};

**int** **lastNon0Digit**(**int** n) {

**if** (n < **10**) **return** dig[n];

**if** (((n/**10**)%**10**)%**2** == **0**)

**return** (**6**\*lastNon0Digit(n/**5**)\*dig[n%**10**])% **10**;

**else**

**return** (**4**\*lastNon0Digit(n/**5**)\*dig[n%**10**]) % **10**;

}

**4 DP**

**4.01 LCS**

**Time -> O(m \* n), Space -> O(n)**

**/Space optimized C++ implementation of LCS problem**

**// Returns length of LCS for X[0..m-1], Y[0..n-1]**

**int lcs(string &X, string &Y) {**

**// Find lengths of two strings**

**int m = X.length(), n = Y.length();**

**int L[2][n + 1];**

**// Binary index, used to index current row and**

**previous row.**

**bool bi;**

**for (int i = 0; i <= m; i++) {**

**// Compute current binary index**

**bi = i & 1;**

**for (int j = 0; j <= n; j++) {**

**if (i == 0 || j == 0) L[bi][j] = 0;**

**else if (X[i-1] == Y[j-1])**

**L[bi][j] = L[1 - bi][j - 1] + 1;**

**else**

**L[bi][j] = max(L[1 - bi][j],**

**L[bi][j - 1]);**

**}**

**}**

**// Last filled entry contains length of LCS**

**for X[0..n-1] and Y[0..m-1]**

**return L[bi][n];**

**}**

**int main() {**

**string X = "AGGTAB";**

**string Y = "GXTXAYB";**

**printf("Length of LCS is %d\n", lcs(X, Y));**

**return 0;**

**}**

**4.02 LIS**

**T -> THETA (N log N)**

**int LISLength(vector<int>& v) {**

**if (v.size() == 0) return 0;**

**vector<int> tail(v.size(), 0);**

**int length = 1, tail[0] = v[0];**

**for (int i = 1; i < v.size(); i++) {**

**// Do binary search for the element in**

**// the range from begin to begin + length**

**auto b=tail.begin(),**

**e= tail.begin()+length;**

**auto it = lower\_bound(b, e, v[i]);**

**// If !present cng the tail element 2 v[i]**

**if (it == tail.begin() + length)**

**tail[length++] = v[i];**

**else \*it = v[i];**

**}**

**return length;**

**}**

**4.03 n-th Permutation**

**long** **long** fac[**26**]; **long** **long** pos, n; **int** freq[**26**];

**void** **init**() {

fac[**0**] = **1**;

**for**( **int** i=**1**; i<**26**; i++ )

fac[i] = fac[i-**1**] \* i;

}

**long** **long** **koita**( **int** n ) {

**long** **long** ret = fac[n];

**for**(**int** i=**0**; i<**26**; i++ ) ret /= fac[ freq[i] ];

**return** ret;

}

**void** **solve**( **int** sz ) {

**long** **long** upto, now; **bool** found;

**while**( sz ) {

upto = **0**; found = **0**;

**for**( **int** i=**0**; i<**26** && !found; i++ ) {

**if**( freq[i] == **0** ) **continue**;

freq[i]--; now = koita( sz-**1** );

**if**( now + upto >= n ) {

n -= upto; sz--;

putchar( ’a’ + i ); found = **1**;

} **else** {

freq[i]++; upto += now;

}

}

**if**( !found ) **break**; }

putchar( ’\n’ );

}

**4.04 0-1 Knapsack**

**T-> O(n \* w)**

**int knapSack(int W,int wt[], int val[], int n) {**

**int i, w, K[n+1][W+1];**

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

**for (w = 0; w <= W; w++) {**

**if (i==0 || w==0) K[i][w] = 0;**

**else if (wt[i-1] <= w)**

**K[i][w] = max(val[i-1] +**

**K[i-1][w-wt[i-1]], K[i-1][w]);**

**else K[i][w] = K[i-1][w];**

**}**

**}**

**return K[n][W];**

**}**

**int main() {**

**int val[] = {60, 100,120},wt[] = {10, 20, 30};**

**int W = 50;**

**int n = sizeof(val)/sizeof(val[0]);**

**printf("%d", knapSack(W, wt, val, n));**

**return 0;**

**}**