```
    }
    while (curr_r> r) {
      --curr_r;
      curr_ans -= a[curr_r];
    }
    ans[q[i].idx] = curr_ans;
  }
  for (int i = 0; i< query; i++) {
    cout<<ans[i] <<endl;
  }
  return 0;
} // end
```

Disjoint Set Union(DSU): // =>O(1)

Applications:1) Cycle detection. 2) Connected Components in graph. 3) MST(Minimum Spanning Tree).

```
const int N = 1e5 + 10;
int par[N];
int Size[N];

// returns the representative of the set that
contains the element v
int Find(int v) {
  if (par[v] == v) return v;
  return par[v] = Find(par[v]);
  // Path Compression
}

// merges the two specified sets(u & v)
void Union(int u, int v) {
  int repU = Find(u);
  int repV = Find(v);
  if (repU != repV)
  {
    // Union by size
    if (Size[repU] < Size[repV]) swap(repU, repV);
    par[repV] = repU;
    Size[repU] += Size[repV];
  }
}

int get_size(int i) {
  return Size[Find(i)];
}

int numberOfConnectedComponents(int n) {
  int ct = 0;
  for (int i = 1; i <= n; ++i)
  {
    if (Find(i) == i) ++ct;
  }
  return ct;
}

void build(int n) {
  for (int i = 0; i < n; i++) {
    par[i] = i;
    Size[i] = 1;
  }
}

int main() {
  int u, v, tc, n, k;
  cin >> n >> k;

  build(N); // Create a new set

  bool cycle = 0;
  for (int i = 1; i <= k; i++) {
    cin >> u >> v;
    /* //Finding Cycle
    if(Find(u)==Find(v)) cycle=1;   //Cycle is
                                        Found;
    else Union(u, v); */
    Union(u, v);
  }
  // if(cycle) cout<<"Found Cycle";

  cout << numberOfConnectedComponents(n) <<
endl; // Count Connected Components

  return 0;
}
```

Lowest Common Ancistor(LCA):

```
const int N = 1e5 + 10;
vector<int> g[N];
int table[N + 1][22];
int level[N];
int tin[N], tout[N];
int minLen[N + 1][22], maxLen[N + 1][22];
// maximum ans minimun weight of a tree
int n, lg, Time = 0, INF = 1e9 + 10;

void dfs(int v, int par = -1, int dep = 0, int mn=INF,
int mx=-1)
{
  tin[v] = ++Time;    //for find is_ancestor
  table[v][0] = par;
  level[v] = dep;
  minLen[v][0] = mn, maxLen[v][0] = mx;
  for (int i = 1; i<= lg; i++)
  {
    if (table[v][i - 1] != -1)
    {
      table[v][i] = table[table[v][i - 1]][i - 1];
      //minLen[v][i] = min(minLen[v][i - 1],
```