SegTree & BITS

- MegaInversion (BIT):

```
const int maxn = 1e7 + 5;
int N;
ll ftA[maxn + 1], ftB[maxn + 1];

int lso(int i){
  int least_significant_one = i & (-i);
  return least_significant_one;
}

void ajusteA(int pos, int val){
  if (pos == 0) return;
  ftA[pos] += val;
  for (pos = pos+lso(pos); pos <= maxn; pos += lso(pos))
      ftA[pos] += val;
}

ll acumuladoA(int pos){
  ll ans = ftA[pos];
  for (pos = pos-lso(pos); pos > 0; pos -= lso(pos))
      ans += ftA[pos];
  return ans;
}

ll intervaloA(int ini, int fim){
  ll res = acumuladoA(fim) - acumuladoA(ini-1);
  return res;
}
//MAIN
  for (int i = 0; i <= maxn; i++)
      ftA[i] = 0; ftB[i] = 0;
  ll ans = 0;
  for (int i = 0; i < N; i++){
      ll numA = nums[N-1-i];
      ansA[N-1-i] = intervaloA(1, numA-1);
      ajusteA(numA, 1);

      ll numB = nums[i];
      ansB[i] = intervaloB(numB+1, maxn);
      ajusteB(numB, 1);
  }

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

//ios::sync_with_stdio(false), cin.tie(0);
#include <bits/stdc++.h>
#define ii pair<int, int>
#define vi vector<int>
#define vs vector<string>
#define si set<int>
#define ll long long
#define endl "\n"
#define pb(a) push_back(a)
//g++-9 -std=c++11 -Wall -Wextra -pedantic-errors main.cpp -o main
```

- MegaInversion (Segtree):

```
vi treeA(4*maxn, 0), treeB(4*maxn, 0);
void updateA(int pos, int i, int j, int x, int value){
  int esq = 2*pos;
  int dir = 2*pos + 1;
  int mid = (i+j)/2;

  if (i == j)
      treeA[pos] += value;
  else{
      treeA[pos] += value;
      if (x <= mid) updateA(esq, i, mid, x, value);
      else updateA(dir, mid+1, j, x, value);
  }
}
ll queryA(int pos, int i, int j, int l, int r){
  int esq = 2*pos;
  int dir = 2*pos + 1;
  int mid = (i+j)/2;
  ll ret;

  if (j < l || i > r)
      ret = 0;
  else if (i >= l && j <= r)
      ret = treeA[pos];
  else{
      ret = 0;
      ret += queryA(esq, i, mid, l, r);
      ret += queryA(dir, mid+1, j, l, r);
  }
  return ret;
}

  for (int i = 0; i < N; i++){
      ll numA = nums[N-1-i];
      answerA[N-1-i] = queryA(1, 0, maxn, 1, numA-1);
      updateA(1, 0, maxn, numA, 1);

      ll numB = nums[i];
      answerB[i] = queryB(1, 0, maxn, numB+1, maxn);
      updateB(1, 0, maxn, numB, 1);
  }
  for (int i = 0; i < N; i++)
      ans += answerA[i] * answerB[i];
  cout << ans << endl;
```

- Segtree do slide:

```
vi cell(maxn);
vi tree(4*maxn);
int N, Q;

void build(int pos, int i, int j){
  int esq = 2*pos;
  int dir = 2*pos + 1;
  int mid = (i+j)/2;

  if (i == j)
      tree[pos] = cell[i];
  else{
      tree[pos] = 0;
      build(esq,i,mid);
      build(dir,mid+1,j);

      tree[pos] = tree[esq] + tree[dir];
  }
}

void update(int pos, int i, int j, int x, int value){//x eh onde vai e value eh quanto

  int esq = 2*pos;
  int dir = 2*pos + 1;
  int mid = (i+j)/2;

  if (i == j)
      tree[pos] += value;
  else{
      tree[pos] += value;
      if (x <= mid) update(esq, i, mid, x, value);
      else update(dir, mid+1, j, x, value);
  }

}

int query(int pos, int i, int j, int l, int r){

  int esq = 2*pos;
  int dir = 2*pos + 1;
  int mid = (i+j)/2;
  int ret;

  if (j < l || i > r)
      ret = 0;
  else if (i >= l && j <= r)
      ret = tree[pos];
  else{
      ret = 0;

      ret += query(esq, i, mid, l, r);
      ret += query(dir, mid+1, j, l, r);
  }

  return ret;
}
```

- Ada and the Tree:

```
vector<vi> cell(maxn), tree(4*maxn);
void merge(int a, int b, int pos){
  //Merge sorting
  int i = 0, j = 0;
  while(i < int(tree[a].size()) && j < int(tree[b].size()))
      (tree[a][i] < tree[b][j]) ? tree[pos].pb(tree[a][i++]) : tree[pos].pb(tree[b][j++]);
  //Colocar o resto de a ou de b
  while(i < int(tree[a].size()))
      tree[pos].pb(tree[a][i++]);
  while (j < int(tree[b].size()))
      tree[pos].pb(tree[b][j++]);
}
void build(int pos, int i, int j){
  int mid = (i+j)/2;
  int esq = pos*2;
  int dir = pos*2+1;
  if (i == j){//se chegou numa folha, atualiza
      tree[pos] = cell[i];
      return;
  }
  //manda pros filhos
  build(esq,i,mid);
  build(dir,mid+1,j);
  //atualiza de acordo com os filhos
  merge(esq, dir, pos);
}
int query(int pos, int i, int j, int l, int r, int lim){
  int esq = 2*pos;
  int dir = 2*pos + 1;
  int mid = (i+j)/2;
  //se extrapolar os limites
  if (j < l || i > r)
      return 0;
  //se estiver dentro dos limites
  if (i >= l && j <= r){
      int ans = --lower_bound(tree[pos].begin(), tree[pos].end(), lim+1) - tree[pos].begin();
      if (tree[pos][ans] > lim)
            return 0;
      return tree[pos][ans];
  }
  //se nao
  int leftNode = query(esq, i, mid, l, r, lim),//pega uma resposta deste vetor
      rightNode = query(dir, mid+1, j, l, r, lim);//pega uma resposta deste vetor
  return max(leftNode, rightNode);// quero o maximo entre elas
}
int main(){
  ios::sync_with_stdio(false), cin.tie(0);
  int N, Q; cin >> N >> Q;
  for (int i = 1; i <= N; i++){
      int aux; cin >> aux;
      cell[i] = vi(1, aux);
  }
  build(1, 1, N);
  while(Q--){
      int i, j; cin >> i >> j; i++; j++;
      int lim; cin >> lim;
      cout << query(1, 1, N, i, j, lim) << endl;
  }
  return 0;
}
```

- Xenia and Bit Operations:

```
int N;
vi cell(maxn), tree(4*maxn, 0);
void build(int pos, int i, int j, bool op){
  int esq = 2*pos;
  int dir = 2*pos + 1;
  int mid = (i+j)/2;

  if (i == j)
      tree[pos] = cell[i];
  else{
      tree[pos] = 0;
      build(esq,i,mid, !op);
      build(dir,mid+1,j, !op);

      if (op)
            tree[pos] = tree[esq] | tree[dir];
      else
            tree[pos] = tree[esq] ^ tree[dir];
  }
}
void update(int pos, int i, int j, int x, int value, bool op){
  int esq = 2*pos;
  int dir = 2*pos + 1;
  int mid = (i+j)/2;
  if (i == j)
      tree[pos] = value;
  else{
      if (x <= mid) update(esq, i, mid, x, value, !op);
      else update(dir, mid+1, j, x, value, !op);

      if (op)
            tree[pos] = tree[esq] | tree[dir];
      else
            tree[pos] = tree[esq] ^ tree[dir];
  }
}
  int n, q; cin >> n >> q;
  int init = 0;
  if (n%2 == 1) init = 1;
  N = pow(2, n);
  for (int i = 1; i <= N; i++)
      cin >> cell[i];
  build(1, 0, N, init);
  while(q--){
      int p,b; cin >> p >> b;
      update(1, 0, N, p, b, init);
      cout << tree[1] << endl;
  }
```

- SUM and REPLACE:

```
struct node{
  int esq, dir;
  ll sum;
  bool v;
};
vi cell(maxn); vector<node> tree(4*maxn);
int countDivisors(int n){
  int c = 0;
  for (int i = 1; i <= sqrt(n); i++){
      if(n%i==0){
          c++;
          if(n/i != i) c++;
      }
  }
  return c;
}
void build(int pos, int i, int j){
  tree[pos].esq = i; tree[pos].dir = j;
  int mid = (i+j)/2;
  int esq = pos*2;
  int dir = pos*2+1;
  if (i == j){//se chegou numa folha, atualiza
      tree[pos].sum = cell[i];
      if (cell[i] <= 2)
          tree[pos].v = 1;
      return;
  }
  //manda pros filhos
  build(esq,i,mid);
  build(dir,mid+1,j);
  //atualiza de acordo com os filhos
  tree[pos].sum = tree[esq].sum + tree[dir].sum;
  tree[pos].v = tree[esq].v & tree[dir].v;
}
void update(int pos, int i, int j){
  if (tree[pos].v)//se for 1 ou 2 nao precisa fatorar
      return;
  int esq = pos*2;
  int dir = pos*2+1;
  int L = tree[pos].esq, R = tree[pos].dir;
  if (i > R || L > j)//se extrapolar os limites
      return;
  if (L==R){//se for uma folha, atualiza
      tree[pos].sum = countDivisors(tree[pos].sum);
      if (tree[pos].sum <= 2)
          tree[pos].v = 1;
      return;
  }
  //manda pros filhos
  update(esq, i, j);
  update(dir, i, j);
  //atualiza de acordo com os filhos
  tree[pos].sum = tree[esq].sum + tree[dir].sum;
  tree[pos].v = tree[esq].v & tree[dir].v;
}
```

```
ll query(int pos, int l, int r){
  int esq = 2*pos;
  int dir = 2*pos + 1;
  int L = tree[pos].esq, R = tree[pos].dir;
  if (l > R || L > r)//se extrapolar os limites
      return 0;
  if (L >= l && R <= r)//se estiver dentro do intervalo
      return tree[pos].sum;
  //busca prox intervalo
  ll ret = 0;
  ret += query(esq, l, r);
  ret += query(dir, l, r);
  return ret;
}
  int N, Q; cin >> N >> Q;
  for (int i = 1; i <= N; i++)
      cin >> cell[i];
  build(1, 1, N);
  while(Q--){
      int cmd; cin >> cmd;
      if (cmd == 1){
          int a, b; cin >> a >> b;
          update(1, a, b);
      }else{
          int a, b; cin >> a >> b;
          cout << query(1, a, b) << endl;
      }
  }
```

- Sereja and Brackets:

```
struct node{
  int l, o, c;
};
vector<node> cell(maxn), tree(4*maxn);
void build(int pos, int i, int j){
  int mid = (i+j)/2;
  int esq = pos*2;
  int dir = pos*2+1;
  if (i == j){//se chegou numa folha, atualiza
       tree[pos] = cell[i];
       return;
  }
  //manda pros filhos
  build(esq,i,mid);
  build(dir,mid+1,j);
  //atualiza de acordo com os filhos
  int t = min(tree[esq].o, tree[dir].c);
  tree[pos].l = tree[esq].l + tree[dir].l + t*2;
  tree[pos].o = tree[esq].o + tree[dir].o - t;
  tree[pos].c = tree[esq].c + tree[dir].c - t;
}
node query(int pos, int i, int j, int l, int r){
  int esq = 2*pos;
  int dir = 2*pos + 1;
  int mid = (i+j)/2;
  //se extrapolar os limites
  if (j < l || i > r)
       return {0, 0, 0};
  //se estiver dentro dos limites
  if (i >= l && j <= r)
       return tree[pos];
  //se nao
  node ret = {0, 0, 0},
        leftNode = query(esq, i, mid, l, r),
        rightNode = query(dir, mid+1, j, l, r);
  //combina os dois elementos
  int t = min(leftNode.o, rightNode.c);
  ret.l = leftNode.l + rightNode.l + t*2;
  ret.o = leftNode.o + rightNode.o - t;
  ret.c = leftNode.c + rightNode.c - t;

  return ret;
}
  for (int i = 0; i < int(in.size()); i++)
       (in[i] == '(') ? cell[i+1].o = 1 : cell[i+1].c = 1;
  build(1, 1, in.size());
  while(Q--){
       int i, j; cin >> i >> j;
       cout << query(1, 1, in.size(), i, j).l << endl;
  }
```