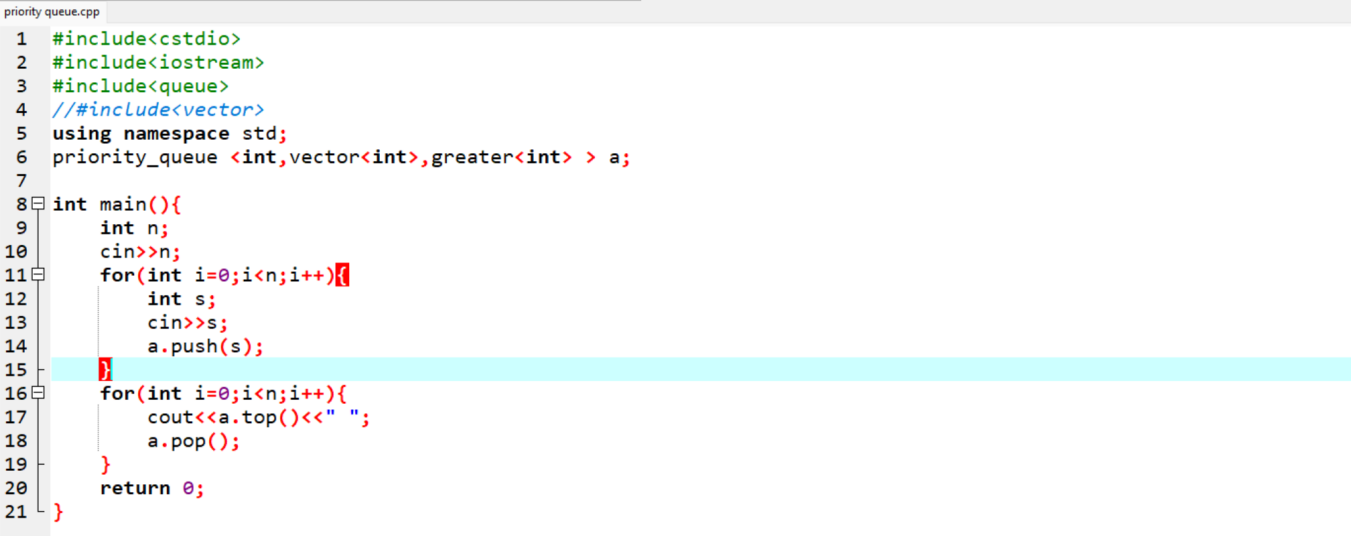
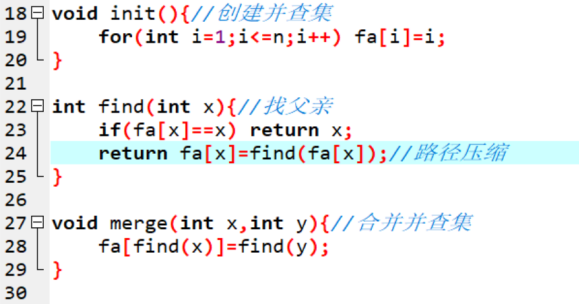
#### STL

Priority queue



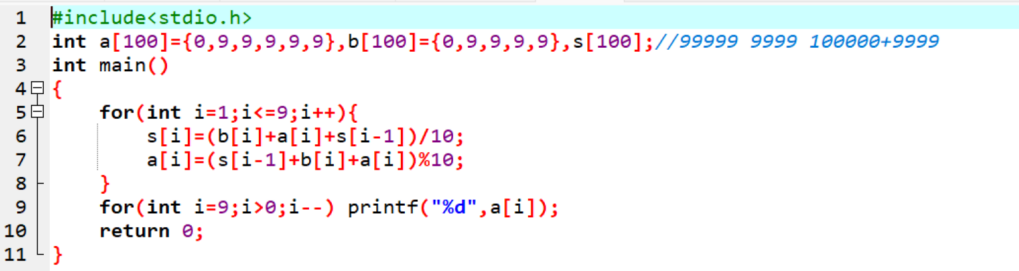
#### 数据结构

并查集

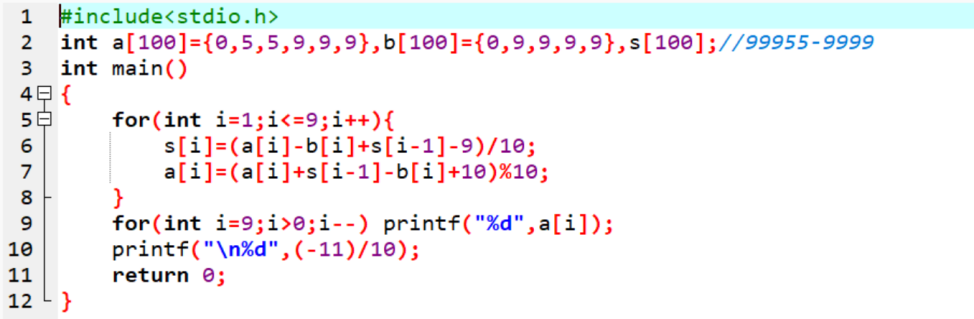


#### 高精度

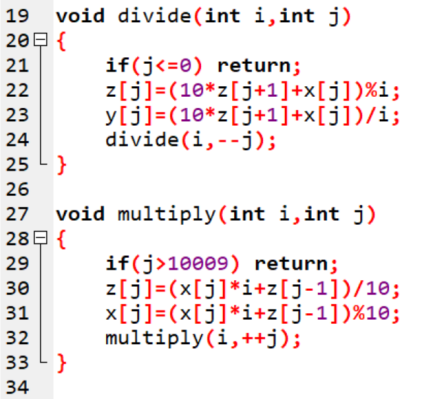
加



减

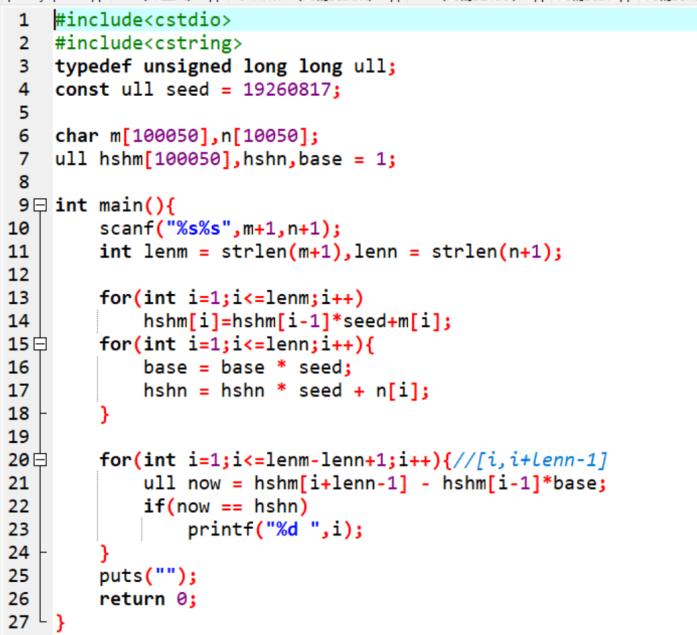


乘除



#### 字符串

哈希

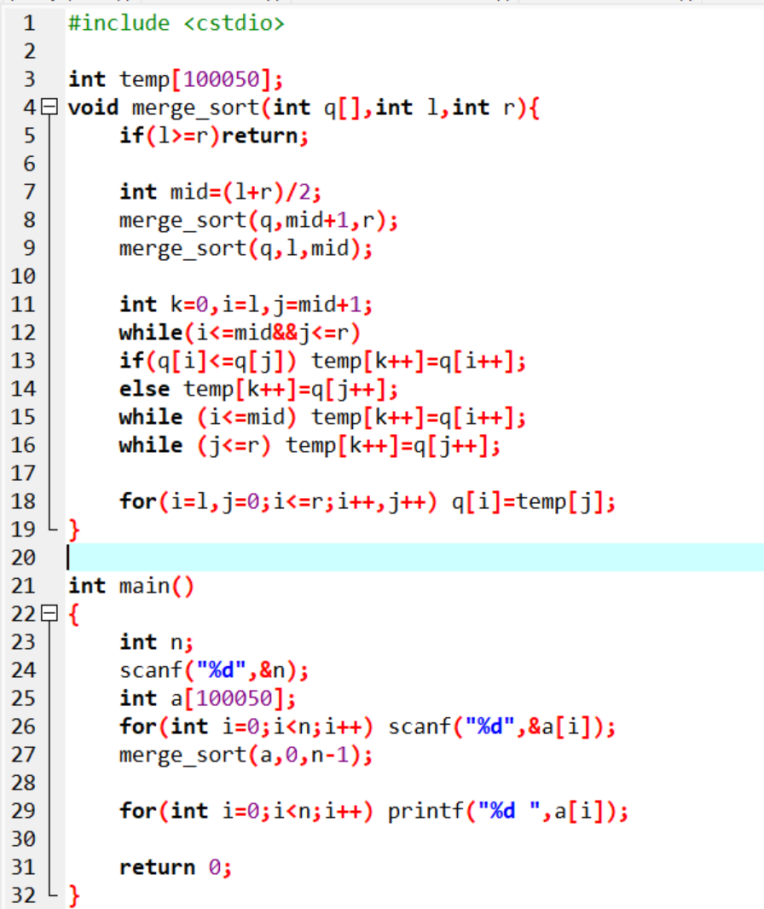


KMP

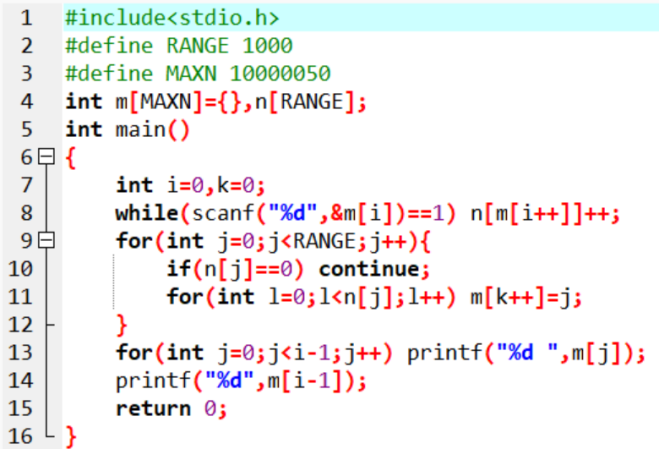


#### 排序

归并排序

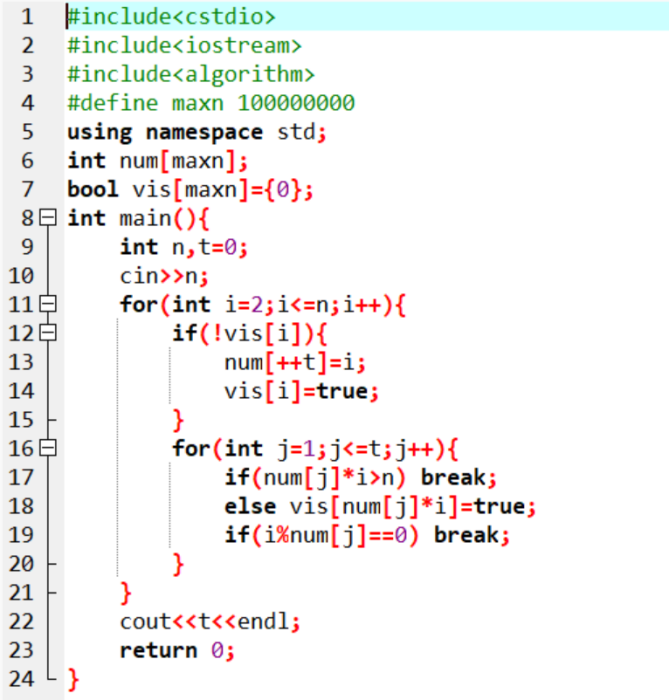


计数排序

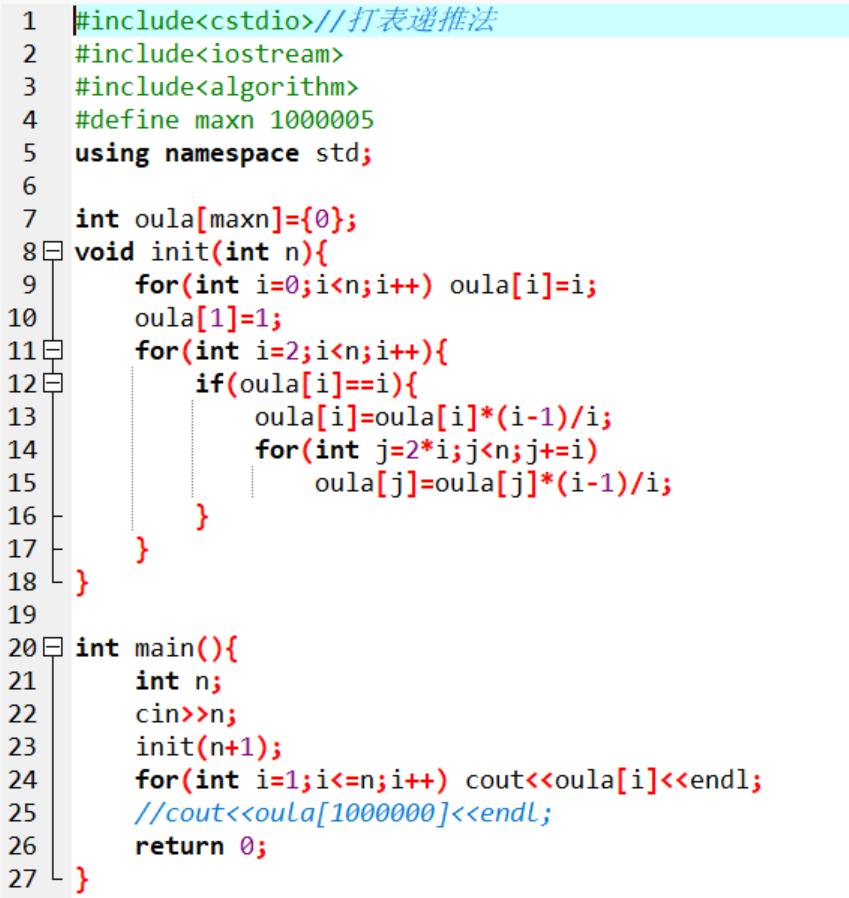


#### 数论

欧拉筛

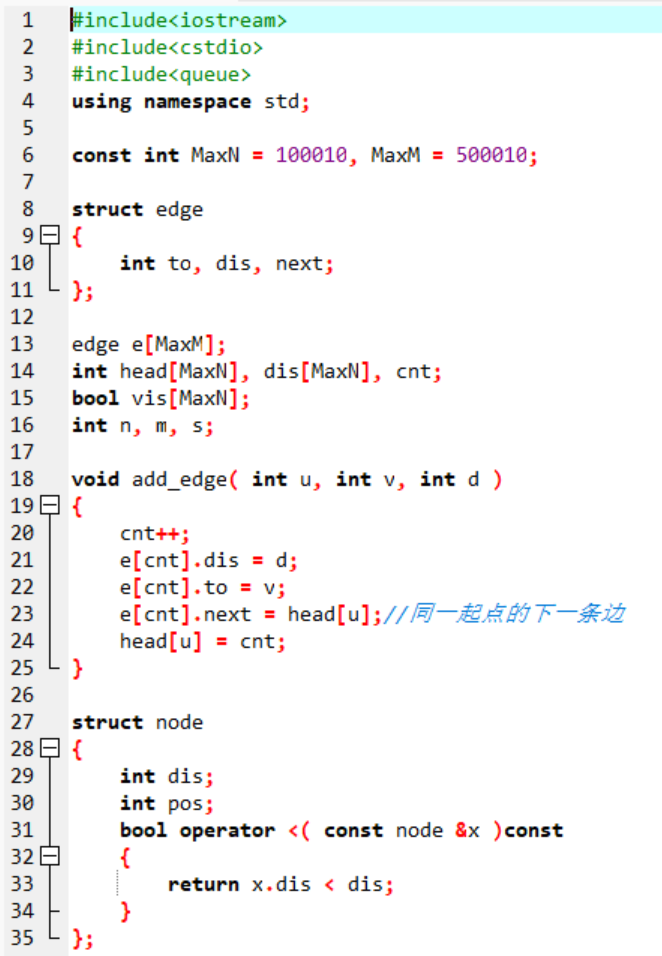
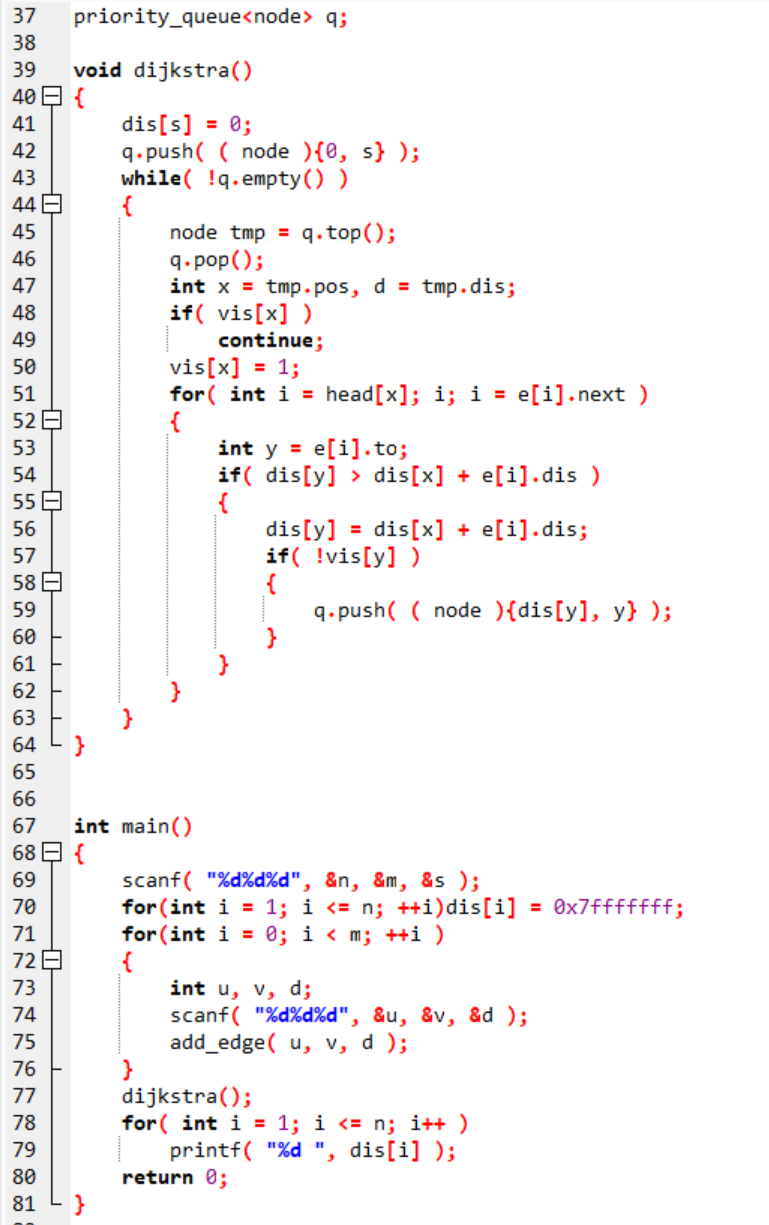


欧拉函数

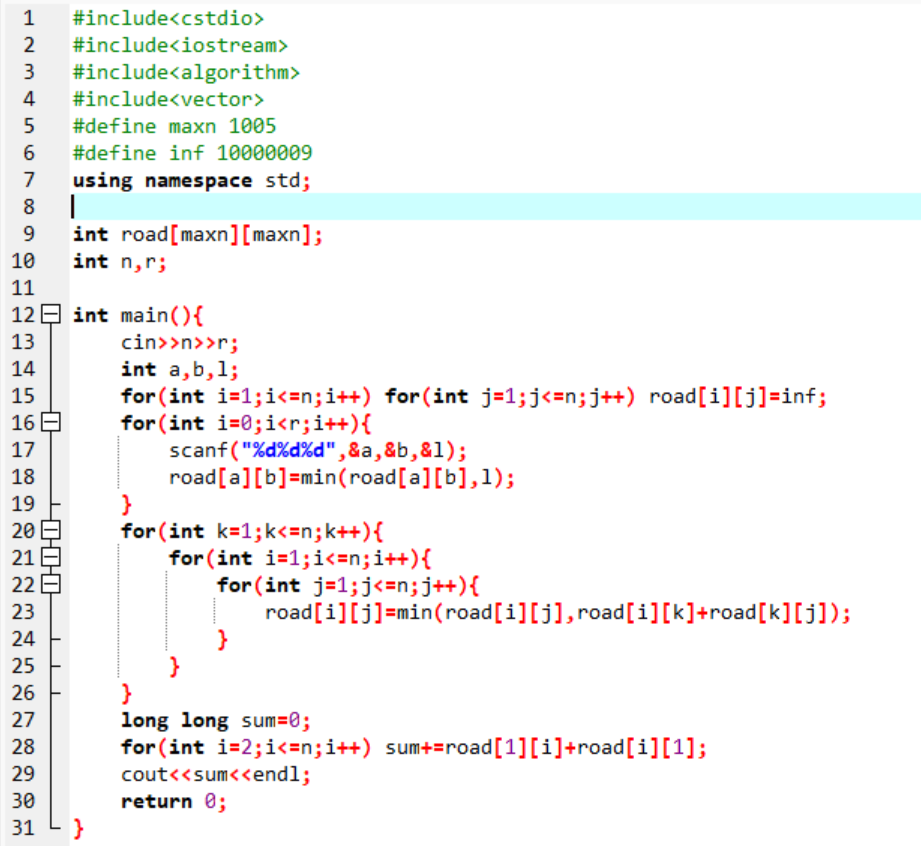


#### 最短路

迪杰斯特拉



FLOYD



#### 树状数组

//单点修改

int tree[MAXN];

inline void update(int i, int x)

{

for (int pos = i; pos < MAXN; pos += lowbit(pos))

tree[pos] += x;

}

//求前n项和

inline int query(int n)

{

int ans = 0;

for (int pos = n; pos; pos -= lowbit(pos))

ans += tree[pos];

return ans;

}

//区间查询

inline int query(int a, int b)

{

return query(b) - query(a - 1);

}

#include <cstdio>

#include <cctype>

#include <algorithm>

#define lowbit(x) ((x) & (-x))

#define MAXN 500010

using namespace std;

typedef long long ll;

ll read() //快速读入，不是这篇文章的重点

{

ll ans = 0;

char c = getchar();

while (!isdigit(c))

c = getchar();

while (isdigit(c))

{

ans = ans \* 10 + c - '0';

c = getchar();

}

return ans;

}

ll tree[MAXN];

inline void update(ll i, ll x)

{

for (ll pos = i; pos < MAXN; pos += lowbit(pos))

tree[pos] += x;

}

inline ll query(int n)

{

ll ans = 0;

for (ll pos = n; pos; pos -= lowbit(pos))

ans += tree[pos];

return ans;

}

inline ll query(ll x, ll y)

{

return query(y) - query(x - 1);

}

int A[MAXN]; //离散化后的数组

typedef struct

{

ll value, id;

} mypair;

mypair B[MAXN]; //原始数组（同时存储id）

bool cmp(mypair x, mypair y)

{

if (x.value < y.value)

return true;

else if (x.value == y.value && x.id < y.id)

return true;

return false;

}

int main()

{

ll n = read(), sum = 0;

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

{

B[i].value = read();

B[i].id = i;

}

sort(B + 1, B + n + 1, cmp);

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

A[B[i].id] = i;

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

{

sum += query(A[i]);

update(A[i], 1);

}

sum = n \* (n - 1) / 2 - sum;

printf("%lld\n", sum);

return 0;

}

///线段树

//题目描述

//如题，已知一个数列，你需要进行下面两种操作：

//1.将某区间每一个数加上x

//2.求出某区间每一个数的和

//输入格式

//第一行包含两个整数N、M，分别表示该数列数字的个数和操作的总个数。

//第二行包含N个用空格分隔的整数，其中第i个数字表示数列第i项的初始值。

//接下来M行每行包含3或4个整数，表示一个操作，具体如下：

//操作1： 格式：1 x y k 含义：将区间[x,y]内每个数加上k

//操作2： 格式：2 x y 含义：输出区间[x,y]内每个数的和

//输出格式

//输出包含若干行整数，即为所有操作2的结果。

#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const int MAXN = 1e5 + 5;

ll tree[MAXN << 2], mark[MAXN << 2], n, m, A[MAXN];

void push\_down(int p, int len)

{

if (len <= 1) return;

tree[p << 1] += mark[p] \* (len - len / 2);

mark[p << 1] += mark[p];

tree[p << 1 | 1] += mark[p] \* (len / 2);

mark[p << 1 | 1] += mark[p];

mark[p] = 0;

}

void build(int p = 1, int cl = 1, int cr = n)

{

if (cl == cr) return void(tree[p] = A[cl]);

int mid = (cl + cr) >> 1;

build(p << 1, cl, mid);

build(p << 1 | 1, mid + 1, cr);

tree[p] = tree[p << 1] + tree[p << 1 | 1];

}

ll query(int l, int r, int p = 1, int cl = 1, int cr = n)

{

if (cl >= l && cr <= r) return tree[p];

push\_down(p, cr - cl + 1);

ll mid = (cl + cr) >> 1, ans = 0;

if (mid >= l) ans += query(l, r, p << 1, cl, mid);

if (mid < r) ans += query(l, r, p << 1 | 1, mid + 1, cr);

return ans;

}

void update(int l, int r, int d, int p = 1, int cl = 1, int cr = n)

{

if (cl >= l && cr <= r) return void(tree[p] += d \* (cr - cl + 1), mark[p] += d);

push\_down(p, cr - cl + 1);

int mid = (cl + cr) >> 1;

if (mid >= l) update(l, r, d, p << 1, cl, mid);

if (mid < r) update(l, r, d, p << 1 | 1, mid + 1, cr);

tree[p] = tree[p << 1] + tree[p << 1 | 1];

}

int main()

{

ios::sync\_with\_stdio(false);

cin >> n >> m;

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

cin >> A[i];

build();

while (m--)

{

int o, l, r, d;

cin >> o >> l >> r;

if (o == 1)

cin >> d, update(l, r, d);

else

cout << query(l, r) << '\n';

}

return 0;

}

//最近祖先

int Log2[MAXN], fa[MAXN][20], dep[MAXN]; // fa的第二维大小不应小于log2(MAXN)

bool vis[MAXN];

void dfs(int cur, int fath = 0)

{

if (vis[cur])

return;

vis[cur] = true;

dep[cur] = dep[fath] + 1;

fa[cur][0] = fath;

for (int i = 1; i <= Log2[dep[cur]]; ++i)

fa[cur][i] = fa[fa[cur][i - 1]][i - 1];

for (int eg = head[cur]; eg != 0; eg = edges[eg].next)

dfs(edges[eg].to, cur);

}

int lca(int a, int b)

{

if (dep[a] > dep[b])

swap(a, b);

while (dep[a] != dep[b])

b = fa[b][Log2[dep[b] - dep[a]]];

if (a == b)

return a;

for (int k = Log2[dep[a]]; k >= 0; k--)

if (fa[a][k] != fa[b][k])

a = fa[a][k], b = fa[b][k];

return fa[a][0];

}

int main()

{

// ...

for (int i = 2; i <= n; ++i)

Log2[i] = Log2[i / 2] + 1;

// ...

dfs(s); // 无根树可以随意选一点为根

// ...

return 0;

}

//离散化

int C[MAXN], L[MAXN];

// 在main函数中...

memcpy(C, A, sizeof(A)); // 复制

sort(C, C + n); // 排序

int l = unique(C, C + n) - C; // 去重

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

L[i] = lower\_bound(C, C + l, A[i]) - C + 1; // 查找

#### 解几与线代

点

