**BINARY SEARCH**

int binsearch(int a[],int sz,int x){

int lo=0, hi=sz-1;

while(lo<=hi){

int mid=lo+(hi-lo)/2;

if (x==a[mid])

return mid;

else if (x<a[mid])

hi=mid-1;

else

lo=mid+1;

}

return -1;

}

**BINARY SEARCH TO FIND FIRST OCCURANCE**

int binsearch(int a[],int sz,int x){

int lo=0, hi=sz-1, ret=-1;

while(lo<=hi){

int mid=lo+(hi-lo)/2;

if (x==a[mid]){

ret=mid;

hi=mid-1;

}

else if (x<a[mid])

hi=mid-1;

else

lo=mid+1;

}

return ret;

}

**BINARY SEARCH FOR FINDING LAST OCCURANCE**

int binsearch(int a[],int sz,int x){

int lo=0, hi=sz-1, ret=-1;

while(lo<=hi){

int mid=lo+(hi-lo)/2;

if (x==a[mid]){

ret=mid;

lo=mid+1;

}

else if (x<a[mid])

hi=mid-1;

else

lo=mid+1;

}

return ret;

}

**SIEVE**

void sieve(){

for (int i = 4; i <= N; i += 2){

num [i] = 1;

}

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

if ( !num[i] ){

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

num[k] = 1;

}

}

}

prime[0] = 2;

int j = 1;

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

if (!num[i]){

prime[j++] = i;

}

}

return;

}

**COUTING FACTORS**

int factors (int n){

if (n < N){

if ( !num[n] ){

return 2;

}

}

int cnt = 1, s = 0, val = sqrt(n)+1;

for (int i = 0; prime[i] < val; i++){

if ( !(n%prime[i]) ){

s = 1;

while ( !(n%prime[i]) ){

n /= prime[i];

s++;

}

cnt \*= s;

}

}

if ( n > 1 ){

cnt \*= 2;

}

return cnt;

}

**SUM OF FACTORS**

ll factor\_sum (ll n){

ll sum = 1, p = 0, val = sqrt(n)+1, s = 0;

for (int i = 0; prime[i] < val; i++){

if ( !(n%prime[i]) ){

p = prime[i];

while ( !(n%prime[i]) ){

p \*= prime[i];

n /= prime[i];

}

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

sum \*= s;

}

}

if (n > 1){

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

}

return sum;

}

**SEGMENT TREE**

int arr[mx];

int tree[mx \* 3];

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

if (b == e) {

tree[node] = arr[b];

return;

}

int Left = node \* 2;

int Right = node \* 2 + 1;

int mid = (b + e) / 2;

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; //বাইরে চলে গিয়েছে

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

return tree[node]; //রিলেভেন্ট সেগমেন্ট

int Left = node \* 2; //আরো ভাঙতে হবে

int Right = node \* 2 + 1;

int mid = (b + e) / 2;

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

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

return p1 + p2; //বাম এবং ডান পাশের যোগফল

}

void update(int node, int b, int e, int i, int newvalue){

if (i > e || i < b)

return; //বাইরে চলে গিয়েছে

if (b >= i && e <= i) { //রিলেভেন্ট সেগমেন্ট

tree[node] = newvalue;

return;

}

int Left = node \* 2; //আরো ভাঙতে হবে

int Right = node \* 2 + 1;

int mid = (b + e) / 2;

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

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

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

}

int main(){

READ("in");

int n;

cin >> n;

repl(i, n)

cin >> arr[i];

init(1, 1, n);

update(1, 1, n, 2, 0);

cout << query(1, 1, n, 1, 3) << endl;

update(1, 1, n, 2, 2);

cout << query(1, 1, n, 2, 2) << endl;

return 0;

}

**BFS (BREADTH FIRST SEARCH) <1D>**

1 procedure BFS(G,source):

2 Q=queue(), level[]=infinity

3 Q.enqueue(source)

4 level[source]=0

5 while Q is not empty

6 u ← Q.pop()

7 for all edges from u to v in G.adjacentEdges(v) do

8 if level[v] = infinity:

9 level[u]=level[v]+1;

10 Q.enqueue(v)

11 end if

12 end for

13 end while

14. Return distance;

**BFS (BREADTH FIRST SEARCH) <2D GRID>**

#define pii pair<int,int>

int fx[]={1,-1,0,0}; //ডিরেকশন অ্যারে

int fy[]={0,0,1,-1};

int cell[100][100]; //cell[x][y] যদি -১ হয় তাহলে সেলটা ব্লক

int d[100][100],vis[100][100]; //d means destination from source.

int row,col;

void bfs(int sx,int sy) //Source node is in [sx][sy] cell.{

memset(vis,0,sizeof vis);

vis[sx][sy]=1;

queue<pii>q; //A queue containing STL pairs

q.push(pii(sx,sy));

while(!q.empty()){

pii top=q.front(); q.pop();

for(int k=0;k<4;k++){

int tx=top.uu+fx[k];

int ty=top.vv+fy[k]; //Neighbor cell [tx][ty]

if(tx>=0 and tx<row and ty>=0 and ty<col and cell[tx][ty]!=-1 and vis[tx][ty]==0){ //Check if the neighbor is valid and not visited before.

vis[tx][ty]=1;

d[tx][ty]=d[top.uu][top.vv]+1;

q.push(pii(tx,ty)); //Pushing a new pair in the queue

}

}

}

}

**BIG MOD**

ll bigmod (ll base, ll pow, ll mod){

if (pow == 0){

return 1;

}

else if (pow%2){

return ((base%mod)\*bigmod(base,pow-1,mod))%mod;

}

else{

ll y = bigmod(base,pow/2,mod);

return (y\*y)%mod;

}

}