#include<bits/stdc++.h>

using namespace std;

class node{

private:

int data;

node \*next;

public:

node \*head = NULL;

node\* tail = NULL;

int size = 0;

void addlast(int value);

void display();

void addfirst(int value);

void removefirst();

int getfirst();

int getlast();

int getat(int key);

void addat(int key, int value);

void removelast();

void removeat(int key);

node\* getnodeAt(int key);

void reverseDi();

void reversePI();

int kthFromEnd(int key);

int mid();

node merged\_list(node l1, node l2);

node merged\_sort(node l1);

};

// node \*head = NULL;

// node\* tail = NULL;

// int size = 0;

void node::addlast(int value){

node\* temp = new node();

temp->data = value;

temp->next = NULL;

if(head==NULL){

head = temp;

tail = temp;

}

else{

tail->next = temp;

tail = temp;

}

size++;

}

void node::addfirst(int value){

node\* temp = new node();

temp->data = value;

if(head==NULL){

temp->next = NULL;

head = temp;

tail = temp;

}

else{

temp->next = head;

head = temp;

}

size++;

}

void node::removefirst(){

if(size==0){

cout<<"List is empty"<<endl;

}

else if(size==1){

node\* temp = head;

head=NULL;

tail=NULL;

delete temp;

size--;

}

else{

node\* temp = head;

head = head->next;

delete temp;

size--;

}

}

int node::getfirst(){

if(head==NULL){

cout<<"List is empty"<<endl;

return -1;

}

else{

return head->data;

}

}

int node::getlast(){

if(head==NULL){

cout<<"List is empty"<<endl;

}

else{

return tail->data;

}

}

int node::getat(int key){

node\* temp = head;

if(size==0){

cout<<"List is empty"<<endl;

return -1;

}

else if(key>=size){

cout<<"Invalid arguments"<<endl;

return -1;

}

else{

for(int i=0;i<key;i++){

temp = temp->next;

}

return temp->data;

}

}

void node::addat(int key, int value){

if(key==0){

addfirst(value);

}

else if(key>size){

cout<<"Invalid arguments"<<endl;

}

else if(key==size){

addlast(value);

}

else{

node\* new\_node = new node();

new\_node->data = value;

node\* temp = head;

node\* prev;

for(int i=0;i<key;i++){

prev= temp;

temp = temp->next;

}

prev->next = new\_node;

new\_node->next = temp;

size++;

}

}

void node::removelast(){

if(size==0){

cout<<"List is empty"<<endl;

}

else if(size==1){

node\* to\_delete = head;

head = NULL;

tail = NULL;

delete to\_delete;

size--;

}

else{

node\* temp = head;

while(temp->next->next!=NULL){

temp = temp->next;

}

node\* to\_delete = temp->next;

temp->next = NULL;

tail = temp;

delete to\_delete;

size--;

}

}

void node::removeat(int key){

if(size==0){

cout<<"List is empty"<<endl;

}

else if(key==size-1){

removelast();

}

else if(key<0 || key>=size){

cout<<"Invalid arguments"<<endl;

}

else if(key==0){

removefirst();

}

else{

node\* temp = head;

node\* prev;

// node\* following;

for(int i=0;i<key;i++){

prev = temp;

temp = temp->next;

}

// following = temp->next;

prev->next = temp->next;

delete temp;

size--;

}

}

node\* node::getnodeAt(int key){

node\* temp = head;

if(size==0){

cout<<"List is empty"<<endl;

return NULL;

}

else if(key>=size){

cout<<"Invalid arguments"<<endl;

return NULL;

}

else{

for(int i=0;i<key;i++){

temp = temp->next;

}

return temp;

}

}

void node::reverseDi(){

int li = 0;

int ri = size-1;

while(li<ri){

node\* left = getnodeAt(li);

node\* right = getnodeAt(ri);

int temp = left->data;

left->data = right->data;

right->data = temp;

li++;

ri--;

}

}

void node::reversePI(){

node\* curr = head;

node\* prev = NULL;

while(curr!=NULL){

node\* temp = curr->next;

curr->next = prev;

prev = curr;

curr = temp;

}

node\* p = head;

head = tail;

tail = p;

}

int node::kthFromEnd(int key){

node\* slow = head;

node\* fast = head;

for(int i=0;i<key;i++){

fast = fast->next;

}

while(fast->next!=NULL){

slow = slow->next;

fast = fast->next;

}

return slow->data;

}

int node::mid(){

node\* slow = head;

node\* fast = head;

while(fast->next!=NULL && fast->next->next!=NULL){

slow = slow->next;

fast = fast->next->next;

}

return slow->data;

}

node node::merged\_list(node l1, node l2){

node\* h1 = l1.head;

node\* h2 = l2.head;

node merged;

while(h1!=NULL && h2!=NULL){

if(h1->data<=h2->data){

merged.addlast(h1->data);

h1 = h1->next;

}

else{

merged.addlast(h2->data);

h2 = h2->next;

}

}

while(h1!=NULL){

merged.addlast(h1->data);

h1 = h1->next;

}

while(h2!=NULL){

merged.addlast(h2->data);

h2 = h2->next;

}

return merged;

}

void node::display(){

node\* temp = head;

while(temp!=NULL){

cout<<temp->data<<" ";

temp = temp->next;

}

cout<<endl;

}

int main(){

node list1;

node list2;

node sorted;

int n,m,num;

cin>>n;

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

cin>>num;

list1.addlast(num);

}

cin>>m;

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

cin>>num;

list2.addlast(num);

}

sorted = sorted.merged\_list(list1, list2);

sorted.display();

list1.display();

list2.display();

}