#include<iostream>

#include<cstring>

#include<algorithm>

#include<cstdlib>

#include<vector>

#include<iterator>

#include<sstream>

#include<utility>

using namespace std;

template<class ElemType>

void print(const vector<ElemType>& v) {

if (v.size() == 0) {

return;

}

for (typename vector<ElemType>::const\_iterator it = v.begin(); it != v.end(); ++it) {

cout << \*it << ' ';

}

}

//一趟归并排序算法: (两路有序并为一路)

template<class ElemType>

void merge(vector<ElemType>& v, int low, int mid, int high) { //low为第1有序区的第1个元素，

//mid为第1有序区的最后1个元素，high为第二有序区的最后一个元素

while (high > low) {

//冒泡排序归并两个序列

for (int i = 0; i < high - low + 1; ++i) {

bool flag = false;

for (int j = low; j < high - i; ++j) {

if (v[j] > v[j + 1]) {

ElemType tmp = v[j];

v[j] = v[j + 1];

v[j + 1] = tmp;

flag = true;

}

}

if (!flag) {

break;

}

}

if (high == v.size() - 1) {//high为整个容器内元素的边界

break;

}

int dlt = high - low;//dlt为high与low相差的量，用于辅助high和low的移动

low = high + 1;

high = low + dlt;//参与归并的两个相邻组的边界标记的移动

if (low > v.size() - 1) {//low移出了容器元素的边界，其实这一步比较多余，因为前面已经判断了

break;//high是否等于v.size()-1，如果可以执行到这一步则之前的high一定不等于v.size()-1，

}//那么加1后得到的新的low值也一定不会出界

else {

if (high > v.size() - 1) {

high = v.size() - 1;//若high出界，则让high等于标号的最大值

}

if (high == low) {//表明此时两组重合且只有一个元素，不排序

break;

}

}

mid = (high + low) / 2;

}

}

//非递归形式的两路归并排序算法

template<class ElemType>

void mergeSort(vector<ElemType>& v) {

if (v.size() > 1) {

int low = 0;

int high = 1;

int mid = (low + high) / 2;//mid没有怎么使用

while (high - low < v.size() - 1) {//当high-low==v.size()-1时，就是两个半大序列的终极合并

merge(v, low, mid, high);

print(v);

cout << endl;

high = 2 \* (high - low + 1) - 1;

if (high >= v.size()) {

high = v.size() - 1;

}

mid = (high + low) / 2;

}

merge(v, low, mid, high);

print(v);

}

else {

print(v);

}

}

int main(){

//数据类型标记（0：int，1：double，2：char，3：string）

int flag;

cin >> flag;

cin.ignore();

switch (flag) {

case 0: {

vector<int>v;

int tmp;

while (cin>>tmp) {//最好采用这种输入方式判断 ，而不要采用cin.peek()!='\n'的方式判断输入是否结束，否则的话会比较耗时

v.push\_back(tmp);

if (cin.get() == '\n')break;

}

mergeSort(v);

}

break;

case 1: {

vector<double>v;

double tmp;

while (cin >> tmp) {

v.push\_back(tmp);

if (cin.get() == '\n')break;

}

mergeSort(v);

}

break;

case 2: {

vector<char>v;

char tmp;

while (cin >> tmp) {

v.push\_back(tmp);

if (cin.get() == '\n')break;

}

mergeSort(v);

}

break;

case 3:{

vector<string>v;

string tmp;

getline(cin, tmp);

string s = "";

for (typename string::const\_iterator it = tmp.begin(); it != tmp.end(); ++it) {

if (\*it != ' ') {

s += \*it;

if (it == tmp.end() - 1) {

v.push\_back(s);

break;

}

}

else if (it == tmp.end() - 1) {

if (\*it != ' ') {

s += \*it;

}

v.push\_back(s);

}

else {

v.push\_back(s);

s = "";

}

}

mergeSort(v);

}

break;

default: cout << "err";

}

return 0;

}