#include<iostream>

#include<algorithm>

using namespace std;

void BubbleSort1(int \*a, int n);

void BubbleSort2(int \*a, int n);

void BubbleSort3(int \*a, int n);

void SelectSort(int \*a, int n);

void InsertSort(int \*a, int n);

void ShellSort(int \*a, int n);

void HeapSort(int \*a, int n);

void HeapAdjust(int\* a, int i, int n);

void mergeSort(int \*a, int n);

void QuickSort(int \*a, int n);

void LSDSort(int \*a, int n);

void MSDSort(int \*a, int n);

int main()

{

int n = 10;

int a[] = { 20,80,90,589,998,965,852,123,456,789 };

//BubbleSort1(a, n);

//BubbleSort2(a, n);

//BubbleSort3(a, n);

//SelectSort(a, n);

//InsertSort(a, n);

// ShellSort(a, n);

//HeapSort(a, n);

//mergeSort(a, n);

//QuickSort(a, n);

// LSDSort(a, n);

MSDSort(a,n);

for (int i = 0; i < n-1; i++)

cout << a[i] << ' ';

cout << a[n - 1] << endl;

system("pause");

return 0;

}

//冒泡排序初级版（这不是真正的冒泡排序），每次交换使得a[i]得到目前的最小值；

void BubbleSort1(int \*a, int n)

{

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

{

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

{

if (a[i]>a[j])

swap(a[i], a[j]);

}

}

}

//真正的冒泡排序

void BubbleSort2(int \*a, int n)

{

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

{

for (int j = n - 2; j >= i; j--)

{

if (a[j]>a[j + 1])

swap(a[j], a[j + 1]);

}

}

}

//冒泡排序优化版本

void BubbleSort3(int \*a, int n)

{

bool flag = true;

for (int i = 0; i < n&&flag==true; i++)

{

flag = false;

for (int j = n - 2; j >= i; j--)

{

if (a[j]>a[j + 1])

{

swap(a[j], a[j + 1]);

flag = true;

}

}

}

}

//简单选择排序

void SelectSort(int \*a, int n)

{

int min;

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

{

min = i;

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

{

if (a[min]>a[j])

min = j;

}

if (min != i)

swap(a[i], a[min]);

}

}

//简单插入排序

void InsertSort(int \*a, int n)

{

int temp, j;

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

{

if (a[i] < a[i - 1])

{

temp = a[i];

for (j = i - 1; a[j]>temp&&j>=0; j--)

a[j + 1] = a[j];

a[j + 1] = temp;

}

}

}

//希尔排序，复杂度为N的1.5次方,跳跃式移动，不稳定；

void ShellSort(int \*a, int n)

{

int i, j, temp;

int increment = n;//increment=9;

do{

increment = increment/3+1;

for (i = increment; i<n; i++)

{

if (a[i]<a[i - increment])

{

temp = a[i];

for (j = i - increment; j >=0 && temp<a[j]; j = j - increment)

a[j + increment] = a[j];

a[j + increment] = temp;

}

}

}

while (increment>1);

}

//堆排序，不稳定，复杂度为NlogN

void HeapSort(int \*a, int n)

{

for (int i = (n - 1) / 2; i >= 0; i--)

HeapAdjust(a, i, n);

for (int i = n - 1; i >=0; i--)

{

swap(a[0], a[i]);

HeapAdjust(a, 0, i);

}

}

void HeapAdjust(int\* a, int i, int n)

{

int leftIndex = 2 \* i + 1;

int rightIndex = 2 \* i + 2;

int max = i;

if (i <= (n - 1) / 2)

{

if (leftIndex <n&&a[leftIndex]>a[max])

max = leftIndex;

if (rightIndex<n&&a[rightIndex]>a[max])

max = rightIndex;

if (max != i)

{

swap(a[i], a[max]);

HeapAdjust(a, max, n);

}

}

}

//归并排序，空间复杂度为O（N），时间复杂度为NlogN；是一种稳定的排序；

void mergeArray(int \*a, int first, int mid, int last, int\* temp)

{

int i = first;

int j = mid + 1;

int p = mid;

int q = last;

int k = 0;

while (i <= p&&j <= q)

{

if (a[i] <= a[j])

temp[k++] = a[i++];

else

temp[k++] = a[j++];

}

while (i <= p)

temp[k++] = a[i++];

while (j <= q)

temp[k++] = a[j++];

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

a[first + i] = temp[i];

}

void MergeSort(int\* a, int first, int last, int\* temp)

{

if (first < last)

{

int mid = (first + last) / 2;

MergeSort(a, first, mid, temp);//左边有序

MergeSort(a, mid + 1, last, temp);//右边有序

mergeArray(a, first, mid, last, temp);//将两个序列合并

}

}

void mergeSort(int \*a, int n)

{

int\* p = new int[n];

MergeSort(a, 0,n-1, p);

delete[] p;

}

//快速排序，复杂度NlogN,不稳定

int Partion(int \*a, int first, int last)

{

int val = a[first];

while (first < last)

{

while (first < last&&a[last] >= val)

last--;

swap(a[first], a[last]);

while (first < last&&a[first] <= val)

first++;

swap(a[first], a[last]);

}

return first;

}

void quickSort(int \*a, int first, int last)

{

int part;

if (first < last)

{

part = Partion(a, first, last);

quickSort(a, first, part - 1);

quickSort(a, part + 1, last);

}

}

void QuickSort(int \*a, int n)

{

quickSort(a, 0, n - 1);

}

//基数排序，最低位优先法，稳定，时间复杂度为O（d\*(n+r))(d是最大位数，r是进制数，一般为10）

int getdigit(int number, int d)

{

int c[] = { 1, 1, 10, 100 };

return (number / c[d]) % 10;

}

void LsdSort(int \*a, int begin, int end, int d)

{

const int radix = 10;

int count[radix], i, j;

int \*bucket = new int[end - begin + 1];

for (int k = 0; k <= d; ++k)

{

for (i = 0; i < radix; i++)

count[i] = 0;

for (i = begin; i <= end; i++)

count[getdigit(a[i], k)]++;

for (i = 1; i < radix; i++)

count[i] += count[i - 1];

for (i = end; i >= begin; i--)

{

j = getdigit(a[i], k);

bucket[count[j] - 1] = a[i];

--count[j];

}

for (i = begin, j = 0; i <= end; i++, j++)

a[i] = bucket[j];

}

delete[] bucket;

}

void LSDSort(int \*a, int n)

{

LsdSort(a, 0, n - 1, 3);

}

//基数排序，最高位优先法，稳定，时间复杂度为O（d\*(n+r))(d是最大位数，r是进制数，一般为10）

void MsdSort(int \*a, int begin, int end, int d)

{

const int radix = 10;

int count[radix], i, j;

for (i = 0; i < radix; i++)

count[i] = 0;

int \*bucket = new int[end - begin + 1];

for (i = begin; i <= end; i++)

count[getdigit(a[i], d)]++;

for (i = 1; i < radix; i++)

count[i] += count[i - 1];

int count1[radix];

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

count1[i] = count[i];

for (i = end; i >= begin; i--)

{

j = getdigit(a[i], d);

bucket[count[j] - 1] = a[i];

--count[j];

}

for (i = begin, j = 0; i <= end; i++, j++)

a[i] = bucket[j];

delete[] bucket;

for (i = 0; i < radix; i++)

{

int p1 = begin + count1[i];

int p2 = begin + count1[i + 1] - 1;

if (p1<p2&&d>1)

MsdSort(a, p1, p2, d - 1);

}

}

void MSDSort(int \*a, int n)

{

MsdSort(a, 0, n - 1, 3);

}