```c
/* sort.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <math.h>
#include "LinkStack.h"

#define LONG_LONG_MIN ((unsigned long long)1 << (sizeof(long long) * 8 - 1))
#define qsSwitch 0

//a，b不能指向同一元素
#define SWAP(a, b) (a ^= b, b ^= a, a ^= b)

void bubbleSort(long long *a, int n)
{
    int i, j;
    for (i = 0; i < n; i ++)
    {
        int done = 1;
        for (j = 1; j < n - i; j ++)
        {
            if (a[j - 1] > a[j])
            {
                SWAP(a[j - 1], a[j]);
                done = 0;
            }
        }
        if (done)
        {
            break;
        }
    }
}

void selectionSort(long long *a, int n)
{
    int rank;
    int i, j;
    for (i = 0; i < n; i ++)
    {
        rank = 0;
        for (j = 1; j < n - i; j ++)
        {
            if (*(a + rank) < *(a + j))
            {
                rank = j;
            }
        }
        //带交换的两个变量指向同一个地址时不能调用SWAP宏，会出错
        if (rank != n - i - 1)
        {
            SWAP(a[rank], a[n - i - 1]);
        }
    }
}

void maxHeapDown(long long *a, int start, int end)
{
    int cur = start;//current node position
    int left = 2 * cur + 1; //left child position
    int tmp = a[cur];//current node value
    for (; left <= end; cur = left, left = 2 * left + 1)
    {
        if (left < end && a[left] < a[left + 1])
        {
            left ++;
        }
        if (tmp >= a[left])
        {
            break;
        }
        else
        {
            a[cur] = a[left];
            a[left] = tmp;
        }
    }
}

void heapSort(long long *a, int n)
{
    int i;
    //get maximum heap, Floyd算法--自下而上的下虑，时间复杂度O(n)
    for (i = n / 2 - 1; i >= 0; i --)
    {
        maxHeapDown(a, i, n - 1);
    }
    //sort--put current maximum data in the current last position
    for (i = n - 1; i > 0; i --)
    {
        SWAP(a[0], a[i]);
        maxHeapDown(a, 0, i - 1);
    }
}

//合并操作
void merge(long long *a, int lo, int mid, int hi)
{
    int len1 = mid - lo + 1;
    int len2 = hi - mid;
    long long *left = (long long *)malloc(len1 * sizeof(long long));
    long long *right = (long long *)malloc(len2 * sizeof(long long));
    int i, j, k;
    for (i = 0; i < len1; i ++)
    {
        left[i] = a[lo + i];
    }
    for (i = 0; i < len2; i ++)
    {
        right[i] = a[mid + 1 + i];
    }
    i = 0, j = 0, k = lo;
    while(i < len1 && j < len2)
    {
        if (left[i] <= right[j])
        {
            a[k ++] = left[i ++];
        }
        else
        {
            a[k ++] = right[j ++];
        }
    }
    while (i < len1)
    {
        a[k ++] = left[i ++];
    }
    while (j < len2)
    {
        a[k ++] = right[j ++];
    }
    free(left);
    free(right);
}

//[lo, hi]
void mergeSort2Way(long long *a, int lo, int hi)
{
    if (lo < hi)
    {
        int mid = (lo + hi) >> 1;
        mergeSort2Way(a, lo, mid);
        mergeSort2Way(a, mid + 1, hi);
        merge(a, lo, mid, hi);
    }
```

```
}

void mergeSort(long long *a, int n)
{
    mergeSort2Way(a, 0, n - 1);
}

void insertionSort(long long *a, int n)
{
    int i, j;
    long long tmp;
    for (i = 1; i < n; i ++)
    {
        tmp = *(a + i);
        j = i - 1;
        while (j >= 0 && tmp < *(a + j))
        {
            *(a + j + 1) = *(a + j);
            j --;
        }
        *(a + j + 1) = tmp;
    }
}

//勤于拓展，懒于交换
static int partitionA(long long *a, int lo, int hi)
{
    //任选一个元素和首元素进行交换
    int tmp = lo + rand() % (hi - lo + 1);
    //待交换的两个变量指向同一个地址时不能调用SWAP宏，会出错
    if (lo != tmp)
    {
        SWAP(a[lo], a[tmp]);
    }
    long long pivot = a[lo];
    while (lo < hi)
    {
        while (lo < hi && a[hi] >= pivot)
        {
            hi --;
        }
        a[lo] = a[hi];
        while (lo < hi && a[lo] <= pivot)
        {
            lo ++;
        }
        a[hi] = a[lo];
    }
    a[lo] = pivot;
    return lo;
}

//懒于拓展，勤于交换
static int partitionB(long long *a, int lo, int hi)
{
    int tmp = lo + rand() % (hi - lo + 1);
    //待交换的两个变量指向同一个地址时不能调用SWAP宏，会出错
    if (lo != tmp)
    {
        SWAP(a[lo], a[tmp]);
    }
    long long pivot = a[lo];
    while (lo < hi)
    {
        while (lo < hi)
        {
            if (pivot < a[hi])
            {
                hi --;
            }
            else
            {
                a[lo ++] = a[hi];
```

```c
                break;
            }
        }
        while (lo < hi)
        {
            if (a[lo] < pivot)
            {
                lo ++;
            }
            else
            {
                a[hi --] = a[lo];
                break;
            }
        }

    }
    a[lo] = pivot;
    return lo;
}

static int partitionC(long long *a, int lo, int hi)
{
    int tmp = lo + rand() % (hi - lo + 1);
    //待交换的两个变量指向同一个地址时不能调用SWAP宏，会出错
    if (hi != tmp)
    {
        SWAP(a[hi], a[tmp]);
    }
    long long pivot = a[hi];
    int i = lo - 1, j = lo;
    for (; j < hi; j ++)
    {
        if (a[j] <= pivot)
        {
            i ++;
            if (a[i] != a[j])
            {
                SWAP(a[i], a[j]);
            }
        }
    }
    i ++;
    if (a[i] != a[hi])
    {
        SWAP(a[i], a[hi]);
    }
    return i;
}

void quickSortRecur(long long *a, int lo, int hi)
{
    if (lo >= hi)
    {
        return ;
    }
    int index = partitionC(a, lo, hi);
    quickSortRecur(a, lo, index - 1);
    quickSortRecur(a, index + 1, hi);
}

void quickSortNonRecur(long long *a, int lo, int hi)
{
    if (lo >= hi)
    {
        return ;
    }
    STACK s;
    StackNew(&s, sizeof(int), NULL);
    StackPush(&s, &hi);
    StackPush(&s, &lo);
    while (!StackEmpty(&s))
    {
        int l, r;
        long long tmp;
        StackPop(&s, &l);
        StackPop(&s, &r);
        int index = partitionC(a, l, r);
        if (l < index - 1)
        {
            tmp = index - 1;
            StackPush(&s, &tmp);
            StackPush(&s, &l);
        }
        if (r > index + 1)
        {
            tmp = index + 1;
            StackPush(&s, &r);
            StackPush(&s, &tmp);
        }
    }
    StackDispose(&s);
}

void quickSort(long long *a, int n)
{
    if (0 == qsSwitch)
    {
        quickSortRecur(a, 0, n - 1);
    }
    else
    {
        quickSortNonRecur(a, 0, n - 1);
    }
}

//此处要求a[i] >= 0
void countSort(long long *a, int n)
{
    long long *b = (long long *)malloc(sizeof(long long) * n);
    memcpy(b, a, sizeof(long long) * n);
    long long max = LONG_LONG_MIN;
    int i = 0;
    for (; i < n; i ++)
    {
        if (max < a[i])
        {
            max = a[i];
        }
    }
    max ++;
    long long *c = (long long *)malloc(sizeof(long long) * max);
    memset(c, 0, sizeof(long long) * max);
    //记录数据在每个桶中的数量
    for (i = 0; i < n; i ++)
    {
        c[a[i]] ++;
    }
    for (i = 1; i < max; i ++)
    {
        c[i] += c[i - 1];
    }
    for (i = n - 1; i >= 0; i --)
    {
        a[c[b[i]] - 1] = b[i];
        c[b[i]] --;
    }
    free(b);
    free(c);
}

//此处要求a[i] >= 0
void radixSort(long long *a, int n)
{
    long long max = LONG_LONG_MIN;
    int i = 0;
```

```c
    for (; i < n; i ++)
    {
        if (max < a[i])
        {
            max = a[i];
        }
    }
    int d = 0;
    while (max > 0)
    {
        d ++;
        max /= 10;
    }
    long long c[10];
    long long *b = (long long *)malloc(sizeof(long long) * n);
    for (i = 0; i < d; i ++)
    {
        memset(c, 0, sizeof(long long) * 10);
        memcpy(b, a, sizeof(long long) * n);
        //计数排序
        long long base1 = (long long)powl(10, i);
        long long base2 = (long long)powl(10, i + 1);
        int j, rank;
        for (j = 0; j < n; j ++)
        {
            rank = (b[j] % base2) / base1;
            c[rank] ++;
        }
        for (j = 1; j < 10; j ++)
        {
            c[j] += c[j - 1];
        }
        for (j = n - 1; j >=0; j --)
        {
            rank = (b[j] % base2) / base1;
            a[c[rank] - 1] = b[j];
            c[rank] --;
        }
    }
    free(b);
}

int main()
{
    int n;
    scanf("%d", &n);
    long long a[n];
    int i;

    for (i = 0; i < n; i ++)
    {
        scanf("%lld", &a[i]);
    }

    //前5个算法的思想是减而治之
    //bubbleSort(a, n);
    //insertionSort(a, n);
    //selectionSort(a, n);
    //heapSort(a, n);
    //quickSort(a, n);
    //归并排序的思想是分而治之
    //mergeSort(a, n);
    //countSort(a, n);
    radixSort(a, n);

    for (i = 0; i < n; i ++)
    {
        printf("%lld ", a[i]);
    }
    printf("\n");
    return 0;
}
```