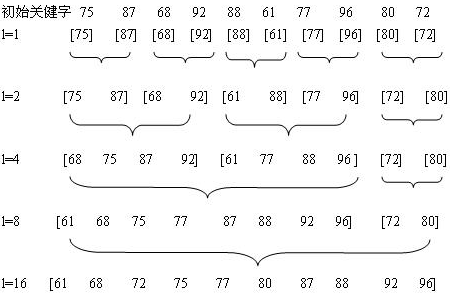
# 归并排序和基数排序

## 实验简介

这章讲解两个经典排序算法，归并排序和基数排序。归并排序是建立在归并操作上的一种有效的排序算法，时间复杂度是O(nlogn)。基数排序不需要进行数据元素间的比较，时间复杂度为O(kn)。

## 一、归并排序

归并排序是建立在归并操作上的一种有效的排序算法，它过程为：比较a[i]和a[j]的大小，若a[i]≤a[j]，则将第一个有序表中的元素a[i]复制到r[k]中，并令i和k分别加上1；否则将第二个有序表中的元素a[j]复制到r[k]中，并令j和k分别加上1，如此循环下去，直到其中一个有序表取完，然后再将另一个有序表中剩余的元素复制到r中从下标k到下标t的单元，如下图所示。



见归并排序演示

归并排序的代码实现：

#include <stdio.h>#include <stdlib.h>

int n;

/\*

\* 合并

\*/void Merge(int \*source, int \*target, int i, int m, int n){

int j, k;

for (j = m + 1, k = i; i <= m && j <= n; k++)

{

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

{

target[k] = source[i++];

}

else

{

target[k] = source[j++];

}

}

while (i <= m)

{

target[k++] = source[i++];

}

while (j <= n)

{

target[k++] = source[j++];

}

}

/\*

\* 归并排序

\*/

void MergeSort(int \*source, int \*target, int s, int t)

{

int m, \*temp;

if (s == t)

{

target[s] = source[s];

}

else

{

temp = (int\*) malloc(sizeof(int) \* (t - s + 1));

m = (s + t) / 2;

MergeSort(source, temp, s, m);

MergeSort(source, temp, m + 1, t);

Merge(temp, target, s, m, t);

}

}

int main()

{

int i;

int \*array;

printf("请输入数组的大小：");

scanf("%d", &n);

array = (int\*) malloc(sizeof(int) \* n);

printf("请输入数据（用空格分隔）：");

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

{

scanf("%d", &array[i]);

}

MergeSort(array, array, 0, n - 1);

printf("排序后为：");

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

{

printf("%d ", array[i]);

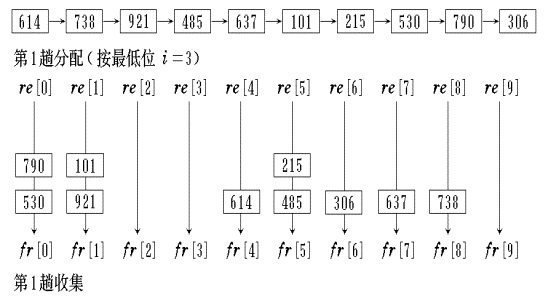
}

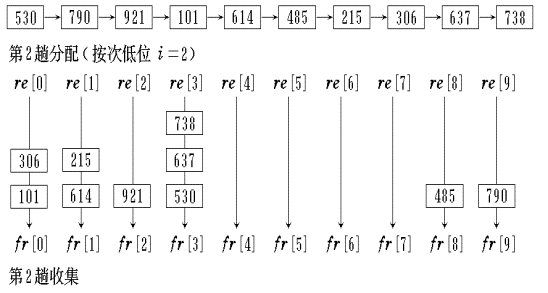
printf("\n");

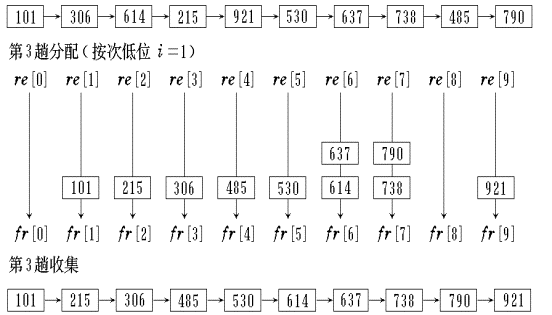
}

## 二、基数排序

基数排序是跟前面的几种排序算法完全不一样的排序算法，前面的排序算法主要通过关键字之间的比较和移动来实现，而基数排序不需要进行关键字之间的比较，它是借助多关键字的思想来实现的。对于数字，每一位上的数字就是一个关键字，每一位的数字范围就是关键字范围，它的主要过程为：将所有待比较数值（正整数）统一为同样的数位长度，数位较短的数前面补零。然后，从最低位开始，依次进行一次排序。这样从最低位排序一直到最高位排序完成以后,数列就变成一个有序序列，如下图所示。类似从低位到高位比较，就是从次关键字到主关键字比较，这种称为最低位优先（LSD），反之称为最高位优先（MSD）。







基数排序的代码实现：

#include <stdio.h>#include <stdlib.h>

int n; //元素个数int bit\_num; //最大数字位数

/\*

\* 获取相应位置上的数（从右到左）

\*/int GetNumInPos(int num, int pos){

int i, temp = 1;

for (i = 0; i < pos - 1; i++)

{

temp \*= 10;

}

return (num / temp) % 10;

}

/\*

\* 基数排序（LSD）

\*/void RadixSort(int \*array){

int radix = 10;

int \*count, \*bucket, i, j, k;

count = (int\*) malloc(sizeof(int) \* radix);

bucket = (int\*) malloc(sizeof(int) \* n);

for (k = 1; k <= bit\_num; k++)

{

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

{

count[i] = 0;

}

//统计各个桶中所盛数据个数

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

{

count[GetNumInPos(array[i], k)]++;

}

//count[i]表示第i个桶的右边界索引

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

{

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

}

//分配

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

{

j = GetNumInPos(array[i], k);

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

count[j]--;

}

//收集

for (i = 0, j = 0; i < n; i++, j++)

{

array[i] = bucket[j];

}

}

}

int main(){

int i;

int \*array;

printf("请输入最大数字的位数：");

scanf("%d", &bit\_num);

printf("请输入数组的大小：");

scanf("%d", &n);

array = (int\*) malloc(sizeof(int) \* n);

printf("请输入数据（用空格分隔）：");

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

{

scanf("%d", &array[i]);

}

RadixSort(array);

printf("排序后为：");

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

{

printf("%d ", array[i]);

}

printf("\n");

}

## 三、小结

这一章讲解了归并排序和基数排序，它们都是稳定的排序算法。归并排序是建立在归并操作上的一种有效的排序算法，它的时间复杂度是O(nlogn)。基数排序不需要进行数据元素间的比较，它是一种借助多关键字的思想对单逻辑关键字进行排序的方法，它分为最低位优先（LSD）和最高位优先（MSD），它的时间复杂度为O(kn)。

## 作业

使用非递归方式实现归并排序。