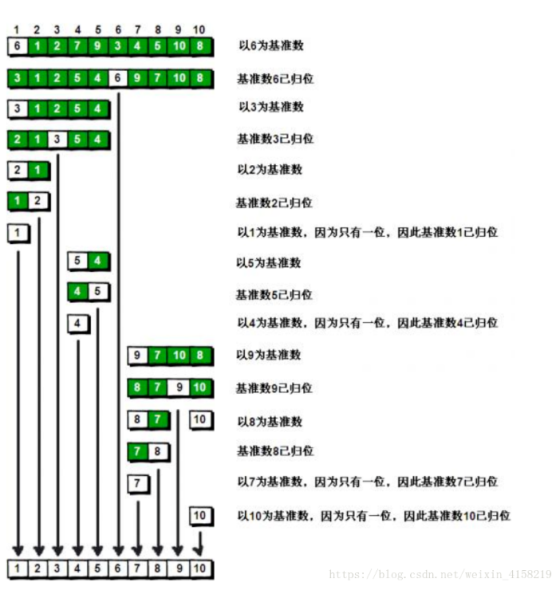
算法分析与设计实验二

一．实验要求：用递归调用实现快速排序和合并排序算法，语言不限，并随机生成测试算例，统计比较两种算法占用的时间。

二．实验知识补充：

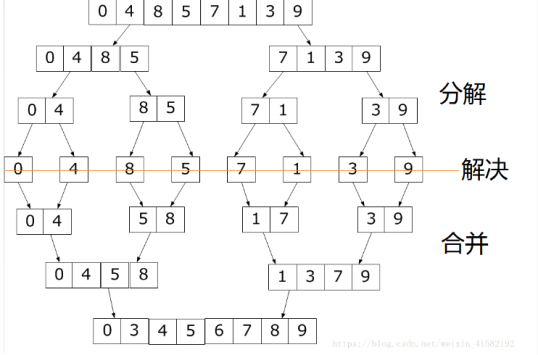
1.快速排序的定义：快速排序同样是一种分治思想的排序方法。它将一个数组分为连个数组，将两部分独立排序。快速排序和归并排序是互补的：归并排序将数组分成两个子数组分别排序，并将有序的子数组归并后整个排序，而快速排序将数组排序的方式则是当两个自数据都有序是整个数组也就自然有序了。

当我们面对一个要排序的数组时，我们首先要找一个基准数（这个基准数没有特殊限定，一般去数组第一个数），然后我们想办法将数组中大于这个基准数的数据放在基准数的一端，小于基准数的数据放在另一端。现在我们就得到了两个子数组，虽然这两个子数组并不是排序完成的，但我们能确定，其中一个子数组内所有的数都小于另一个，也就是说，当我们将这两个子数组排序完成时，整个数组自然就有序了。那么现在就是使用分治思想的地方，分开这两个数组后，再利用上述思想，同样找基准数，将数据按基准数分开两边，这时，我们就拥有了四个子数组。以此下去，当我们每个子数组都是有序的，那么我们就排序完成了。通过递归我们可以实现这一步骤。



2.合并算法的定义：

归并排序时，我们先将要进行排序的数组分为两部分，我们叫做Left和Right，如果我们将这两部分都进行排序完成后，即子数组Left和Right都是有序数组。那么我们将这两个数组进行合并，合并的方式为，首先创建一个与原数组容量相同的数组用来存放合并时的数据，然后比较Left和Right中的数组，如果Left[0]<Right[0]，将Left[0]放入新数组的0索引处，然后比较Left[1]和Right[0]，依次类推按照升序或降序的方式便能将Left和Right中所有数组按照一定顺序拷贝进入新数组，此时就完成了数组排序。归并排序的模型中并不是将一个数组只分为两块，而是分为数组最小单元，再对每个最小单元进行治的处理，这种处理方式要通过递归思想来进行实现。



三.实验代码：

合并算法的实现：

#include <stdlib.h>

#include <string.h>

#include "sort.h"

/\*merge 合并两个有序数据集\*/

static int merge(void \*data, int esize, int i, int j, int k, int (\*compare)(const void \*key1,const void \*key2))

{

char \*a = data,

\*m;

int ipos ,jpos,mpos;

/\*初始化用于合并过程中的计数器\*/

ipos = i;

jpos = j+1;

mpos = 0;

/\*首先，为要合并的元素集分配空间\*/

if((m = (char \*)malloc(esize \* ((k-i)+1))) == NULL)

return -1;

/\*接着，只要任一有序集有元素需要合并，就执行合并操作\*/

while(ipos <= j || jpos <=k)

{

if(ipos > j)

{

/\*左集中没有元素要合并，就将右集中的元素放入目标集（合并集）\*/

while(jpos <= k)

{

memcpy(&m[mpos \* esize],&a[jpos \* esize],esize);

jpos++;

mpos++;

}

continue;

}

else if(jpos > k)

{

/\*右集没有要合并的元素，就将左集中的元素放入目标集（合并集）\*/

while(ipos <= j)

{

memcpy(&m[mpos \* esize],&a[ipos \*esize],esize);

ipos++;

mpos++;

}

continue;

}

/\*追加下一个有序元素到合并集中\*/

if(compare(&a[ipos \* esize],\*a[jpos \*esize])<0)

{

memcpy(&m[mpos \* esize],&a[ipos \* esize],esize);

ipos++;

mpos++;

}

else

{

memccpy(&m[mpos \* esize],&a[jpos \* esize],esize);

jpos++;

mpos++;

}

}

/\*将已经排序的数据集拷贝到原数组中\*/  
 memcpy(&a[i \* esize],m,esize \* ((k-i)+1));  
  
 /\*释放为排序分配的存储空间\*/  
 free(m);  
 return 0;

}  
  
/\*mgsort 归并排序（递归调用）\*/  
int mgsort(void \*data, int size, int esize, int i, int k, int(\*compare)(const void \*key1,const void \*key2))  
{  
 int j;  
  
 /\*递归调用mgsort持续分割，直到没有可以再分割的数据集\*/  
 if(i < k)  
 {  
 /\*计算对半分割的位置下标\*/  
 j = (int)(((i+k-1)) / 2);  
  
 /\*递归排序两边的集合\*/  
 if(mgsort(data, size, esize, i, j, compare) < 0)  
 return -1;  
 if(mgsort(data, size, esize, j+1, k, compare) <0)  
 return -1;  
  
 /\*将两个有序数据集合并成一个有序数据集\*/  
 if(meger(data, esize, i, j, k compare) < 0)  
 return -1;  
 }  
return 0;  
}

快速排序的实现：

#include <stdio.h>

#include <stdlib.h>

void quick\_sort(int \*a, int left, int right)

{

if(left >= right)/\*如果左边索引大于或者等于右边的索引就代表已经整理完成一个组了\*/

{

return ;

}

int i = left;

int j = right;

int key = a[left];

while(i < j)

{

/\*而寻找结束的条件就是，1，找到一个小于或者大于key的数（大于或小于取决于你想升

序还是降序）2，没有符合条件1的，并且i与j的大小没有反转\*/

while(i < j && key <= a[j])j--;

/\*找到一个这样的数后就把它赋给前面的被拿走的i的值（如果第一次循环且key是

a[left]，那么就是给key）\*/

a[i] = a[j];

/\*这是i在当组内向前寻找，同上，不过注意与key的大小关系停止循环和上面相反，

因为排序思想是把数往两边扔，所以左右两边的数大小与key的关系相反\*/

while(i < j && key >= a[i])i++

a[j] = a[i];

}

a[i] = key;//基准值到达最终位置

quick\_sort(a, left, i - 1);//对左半段排序

quick\_sort(a, i + 1, right);//对右半段排序

}

int main(void)

{

int N;

scanf("%d",&N);

int a[N], i;

printf("Please enter %d number of sorting: ", N);

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

{

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

}

quick\_sort(a, 0, N - 1);

printf("In sorted order: ");

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

{

printf("%d \n", a[i]);

}

return 0;

}