1.1快排

int partition(int \*arr,int left,int right) // 是以最右边（而非最左边）作为枢纽

{

int k,i;//k记录要放入比分割值小的数据的位置

for(i=left,k=left;i<right;i++)

{

if(arr[i]<arr[right])

{

SWAP(arr[k],arr[i]);

k++;

}

}

SWAP(arr[k],arr[right]);

return k;

}

void arr\_quick(int\* arr,int left,int right)

{

int pivot\_pos;

if(left<right)

{

pivot\_pos=partition(arr,left,right);

arr\_quick(arr,left,pivot\_pos-1);

arr\_quick(arr,pivot\_pos+1,right);

}

}

int main()

{

int \*arr=(int\*)malloc(sizeof(int)\*N); // 为数组生成所需全部空间

int i;

time\_t start,end; // ----------生成随机数

srand(time(NULL));

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

{

arr[i]=rand()%100;

}

print(arr);

start=time(NULL); //------------十个随机数生成完毕

arr\_quick(arr,0,N-1); // 传入的arr已经是一个地址

end=time(NULL);

print(arr);

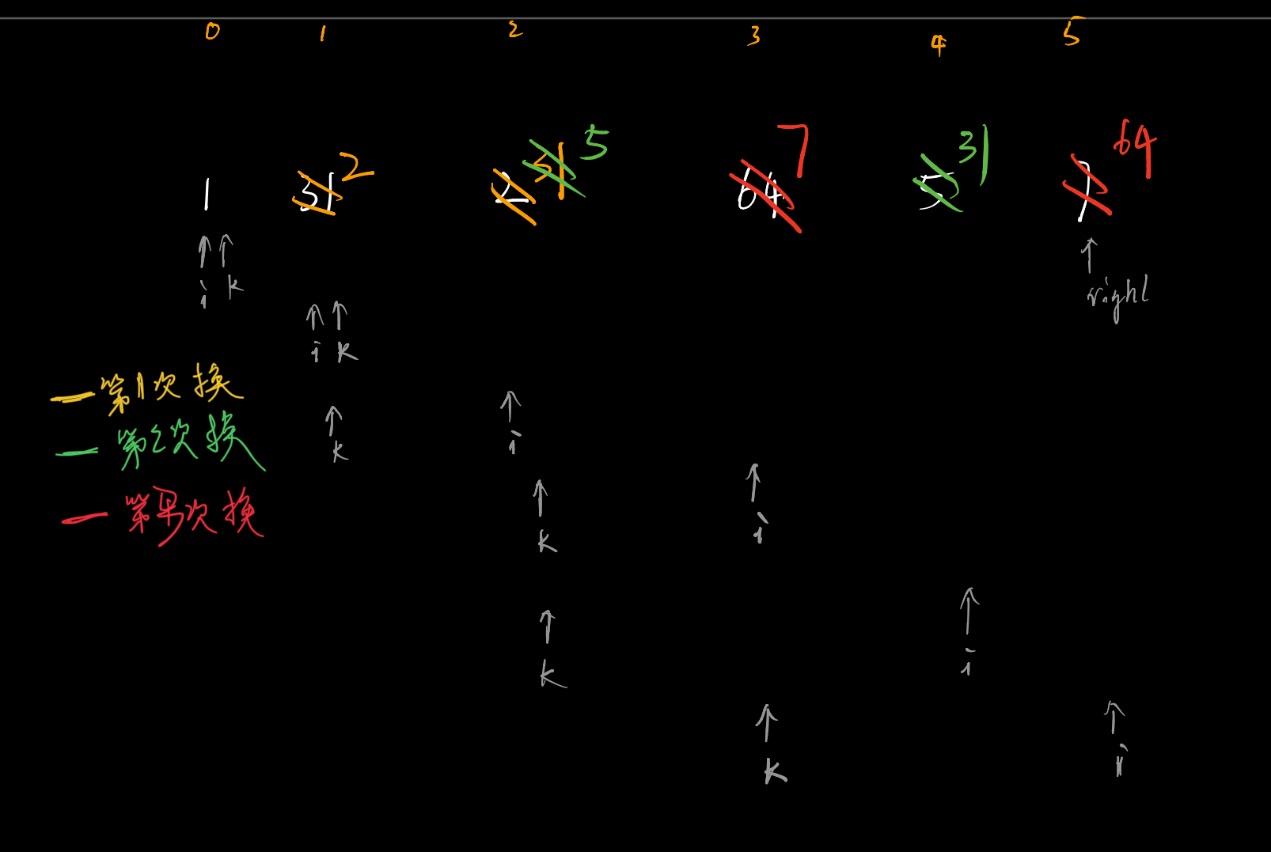
printf("use time=%d\n",end-start);

system("pause");

}



Patition过程



1.2 堆排

void adjust\_max\_heap(int \*arr,int adjust\_pos,int len)

{

int dad=adjust\_pos;

int son=2\*dad+1;

while(son<len) // 本层循环的目的是让子树都调整为大根堆

{

//判断是否有右孩子，以及右孩子是否大于左孩子

if(son+1<len&&arr[son]<arr[son+1])

{

son++; // 确保孩子是较大的那个

}

//如果孩子大于父亲，发生交换

if(arr[son]>arr[dad])

{

SWAP(arr[son],arr[dad]);

dad=son;

son=2\*dad+1;

}else{

break;

}

}

}

void arr\_heap(int \*arr)

{

int i;

//调整为大根堆

for(i=N/2-1;i>=0;i--) //从一半的位置处递减往回遍历

{

adjust\_max\_heap(arr,i,N);

}

//交换根部元素和最后一个元素

SWAP(arr[0],arr[N-1]);

for(i=N-1;i>1;i--)

{

//不断调整根部元素，为大根堆

adjust\_max\_heap(arr,0,i);

//交换根部元素和最后一个元素

SWAP(arr[0],arr[i-1]);

}

int main()

{

int \*arr=(int\*)malloc(sizeof(int)\*N); // 为数组生成所需全部空间

int i;

time\_t start,end; // ----------生成随机数

srand(time(NULL));

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

{

arr[i]=rand()%100;

}

print(arr);

start=time(NULL); //------------十个随机数生成完毕

arr\_heap(arr); // 传入的arr已经是一个地址

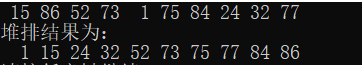
end=time(NULL);

print(arr);

printf("use time=%d\n",end-start);

system("pause");

}



1.3 归并

void merge(int \*arr,int low,int mid,int high)

{

int b[N];

int i,j,k;

for(i=low;i<=high;i++)

{

b[i]=arr[i];

}

for(k=low,i=low,j=mid+1;i<=mid&&j<=high;)

{

if(b[i]<b[j])

{

arr[k]=b[i];

k++;

i++;

}else{

arr[k]=b[j];

k++;

j++;

}

}

while(i<=mid)

{

arr[k]=b[i];

k++;

i++;

}

while(j<=high)

{

arr[k]=b[j];

k++;

j++;

}

}

void arr\_merge(int \*arr,int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

arr\_merge(arr,low,mid);

arr\_merge(arr,mid+1,high);

merge(arr,low,mid,high);

}

}

int main()

{

int \*arr=(int\*)malloc(sizeof(int)\*N); // 为数组生成所需全部空间

int i;

time\_t start,end; // ----------生成随机数

srand(time(NULL));

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

{

arr[i]=rand()%100;

}

print(arr);

printf("归并结果为： \n");

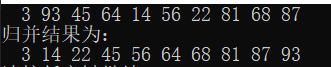
start=time(NULL); //------------十个随机数生成完毕

arr\_merge(arr,0,N-1);

print(arr);

system("pause");

}



1.4 计数排序

void arrCount(int\* arr) {

int count[M] = { 0 };

int i, j, k;

// 遍历数组后，统计数组中每个元素出现次数

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

count[arr[i]]++;

}

//k用来记录数组中哪些元素已经填入值

k = 0;

//将每个数值出现的次数，依次从前到后填入数组arr

for (i = 0; i < M; i++) {

for (j = 0; j < count[i]; j++) {

arr[k++] = i;

}

}

}

int main()

{

int \*arr=(int\*)malloc(sizeof(int)\*N); // 为数组生成所需全部空间

int i;

time\_t start,end; // ----------生成随机数

srand(time(NULL));

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

{

arr[i]=rand()%M;

}

print(arr);

printf("计数排序结果为： \n");

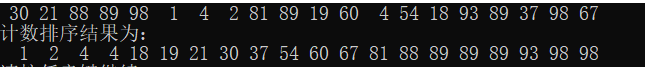
start=time(NULL); //------------十个随机数生成完毕

arrCount(arr);

print(arr);

system("pause");

}



2 完成qsort的熟练使用，排序结构体数组

int compare\_hard(const void \* pleft, const void \*pright) //加const,排序变指针，不变内容

{

pstu\* p1=(pstu\*)pleft;

pstu\* p2=(pstu\*)pright;

if((\*p1)->english>(\*p2)->english) // 用指针对结构体中english这个特征排序

{

return 1;

}else if((\*p1)->english<(\*p2)->english)

{

return -1;

}else{

return 0;

}

}

void use\_qsort()

{

stu sarr[5];

pstu p[5];

int i;

for(i=0;i<5;i++)

{

scanf("%d%s%f%f",&sarr[i].num,sarr[i].name,&sarr[i].english,&sarr[i].math);

p[i]=sarr+i; //把指针传给p数组

}

//qsort(sarr,5,sizeof(stu),compare1); //对总分进行排序 排一般数组，原数组内容改变

//qsort去排指针数组，最终输出有效效果，但是原数组内容打印不变

printf("--------------------------指针排出来的：\n");

qsort(p,5,sizeof(pstu),compare\_hard); //自定义的排序规则为排English

for(i=0;i<5;i++)

{

printf("%d %s %5.2f %5.2f %6.2f\n",p[i]->num,p[i]->name,p[i]->english,p[i]->math,p[i]->english+p[i]->math);

}

printf("--------------------------原数组为：\n");

for(i=0;i<5;i++)

{

printf("%d %s %5.2f %5.2f %6.2f\n",sarr[i].num,sarr[i].name,sarr[i].english,sarr[i].math,sarr[i].english+sarr[i].math);

}

}

int main()

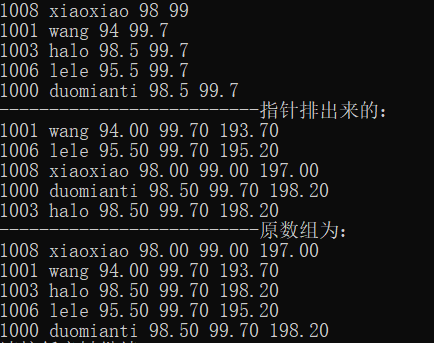
{

use\_qsort();

system("pause");

}

本排序方法不稳定



3 hash

#define MAXKEY 1000

int hash(char\* key) {

int h = 0, g;

while (\*key)

{

h = (h << 4) + \*key++;

g = h & 0x0000000;

if (g) {

h ^= g >> 24;

}

h &= ~g;

}

return h % MAXKEY;

}

int main()

{

char\* pStr[5] = { "xiongda","lele","hanmeimei","wangdao","fenghua" };

char\* hashTable[MAXKEY] = { NULL }; // 全置为NULL而不是0；

int i;

for (i = 0; i < 5; i++) {

printf("%10s hashValue=%d\n", pStr[i], hash(pStr[i]));

hashTable[hash(pStr[i])] = pStr[i];

}

}

