算法设计与分析实验三

一.实验要求：设计算法从给定的n个数中找出最小的k个数。采用递归调用实现算法。

二．实验代码：

方法一：先直接排序，再取排序后数据的前k个数。

排序算法用最快的堆排序，复杂度也会达到O(N\*logN).

void filterDown(int\* disorder, int pos, int size){

int temppos=pos,temp=0;

while(temppos<size/2){

if(2\*temppos+2<size){

if(disorder[2\*temppos+1]>disorder[2\*temppos+2]){

if(disorder[temppos]<disorder[2\*temppos+1]){

temp=disorder[temppos];

disorder[temppos]=disorder[2\*temppos+1];

disorder[2\*temppos+1]=temp;

temppos=2\*temppos+1;

}

else{

break;

}

}

else{

if(disorder[temppos]<disorder[2\*temppos+2]){

temp=disorder[temppos];

disorder[temppos]=disorder[2\*temppos+2];

disorder[2\*temppos+2]=temp;

temppos=2\*temppos+2;

}

else{

break;

}

}

}

else if(disorder[temppos]<disorder[2\*temppos+1]){

temp=disorder[temppos];

disorder[temppos]=disorder[2\*temppos+1];

disorder[2\*temppos+1]=temp;

temppos=2\*temppos+1;

}

else{

break;

}

}

}

void heapSort(int\* disorder, int size){

int bottomRowSize=2;

while(bottomRowSize<size){

bottomRowSize\*=2;

}

int temp=0,i=0;

for(int j=size/2-1;j>=0;j--){

filterDown(disorder, j, size);

}

for(int j=size-1;j>0;j--){

temp=disorder[0];

disorder[0]=disorder[j];

disorder[j]=temp;

filterDown(disorder,0,j);

}

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

const int size=200;

const int maxnum = 10000;

const int k=10;

int\* disorder=new int[size];

srand((unsigned) time(NULL));

for(int i=0;i<size;i++){

disorder[i]=rand()%maxnum;

}

heapSort(disorder,size);

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

cout<<disorder[i]<<endl;

}

return 0;

}

 当k接近于N时，可以用这种算法。

方法二：先排序前k个数，对于后面N-k个数，依次进行插入。

时间复杂度为O(k\*n)

void insertSort(int\* disorder, int size){

int temp=0,i=0;

for(int j=1;j<size;j++){

temp=disorder[j];

for(i=j;i>0;i--){

if(temp<disorder[i-1]){

disorder[i] = disorder[i-1];

}

else{

break;

}

}

disorder[i]=temp;

}

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

const int size=200;

const int maxnum = 10000;

const int k=10;

int\* disorder=new int[size];

srand((unsigned) time(NULL));

int i=0,temp;

for(;i<size;i++){

disorder[i]=rand()%maxnum;

}

insertSort(disorder,k);

for(int j=k;j<size;j++){

temp=disorder[j];

for(i=k-1;i>=0;i--){

if(temp<disorder[i]){

disorder[i+1] = disorder[i];

}

else{

break;

}

}

disorder[i+1]=temp;

}

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

cout<<disorder[i]<<endl;

}

return 0;

}

当k很小时，可以用这种算法

方法三：对前k个数，建立最大堆，对于后面N-k个数，依次和最大堆的最大数比较，如果小于最大数，则替换最大数，并重新建立最大堆。

时间复杂度为O(N\*logk)

void filterDown(int\* disorder, int pos, int size){

int temppos=pos,temp=0;

while(temppos<size/2){

if(2\*temppos+2<size){

if(disorder[2\*temppos+1]>disorder[2\*temppos+2]){

if(disorder[temppos]<disorder[2\*temppos+1]){

temp=disorder[temppos];

disorder[temppos]=disorder[2\*temppos+1];

disorder[2\*temppos+1]=temp;

temppos=2\*temppos+1;

}

else{

break;

}

}

else{

if(disorder[temppos]<disorder[2\*temppos+2]){

temp=disorder[temppos];

disorder[temppos]=disorder[2\*temppos+2];

disorder[2\*temppos+2]=temp;

temppos=2\*temppos+2;

}

else{

break;

}

}

}

else if(disorder[temppos]<disorder[2\*temppos+1]){

temp=disorder[temppos];

disorder[temppos]=disorder[2\*temppos+1];

disorder[2\*temppos+1]=temp;

temppos=2\*temppos+1;

}

else{

break;

}

}

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

const int size=200;

const int maxnum = 10000;

const int k=10;

int\* disorder=new int[size];

srand((unsigned) time(NULL));

int i=0,temp;

for(;i<size;i++){

disorder[i]=rand()%maxnum;

}

for(int j=k/2-1;j>=0;j--){

filterDown(disorder, j, k);

}

for(int j=k;j<size;j++){

if(disorder[j]<disorder[0]){

disorder[0]=disorder[j];

filterDown(disorder,0,k);

}

}

for(int j=k-1;j>0;j--){

temp=disorder[0];

disorder[0]=disorder[j];

disorder[j]=temp;

filterDown(disorder,0,j);

}

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

cout<<disorder[i]<<endl;

}

return 0;

}

实验结果：