//二分查找

int getData(int\*array,int data,int length)

{

int start=0;

int end=length-1;

while (start<=end)

{

int middle=(start+end)/2;

if(data>array[middle])

{

start=middle+1;

}

else if(data<array[middle])

{

end=middle-1;

}

else

return middle;

}

return -1;

}

//冒泡排序

void bubblesort(int \*array,int length)

{

for(int i=length;i>0;i--)

{

bool flag=true;

for(int j=1;j<i;j++)

{

if(array[j-1]>array[j])

{

int temp=array[j-1];

array[j-1]=array[j];

array[j]=temp;

flag=false;

}

}

if(flag)

{

break;

}

}

}

void print(int \*array,int length)

{

for(int i=0;i<length;i++)

{

cout<<array[i]<<" ";

}

cout<<endl;

}

//选择排序

void chooseSort(int \*array,int length)

{

for(int i=length;i>0;i--)

{

int k=0;

for(int j=1;j<i;j++)

{

if(array[k]<array[j])

k=j;

}

if(k!=i-1)

{

int temp=array[k];

array[k]=array[i-1];

array[i-1]=temp;

}

}

}

//插入排序

void inserSort(int \*array,int length)

{

int temp;

for(int i=1;i<length;i++)

{

int j=i-1;

temp=array[i];

while(j>=0)

{

if(temp<array[j])

j--;

else

break;

}

j++;

for(int k=i;k>j;k--)

array[k]=array[k-1];

array[j]=temp;

}

}

//希尔排序

void shellSort(int \*array,int length)

{

int d=length;

while(d>1)

{

d=d>>1;

for(int i=0;i+d<length;i=i+1)

{

int mark=i+d;

int temp=array[mark];

while((mark-d>=0)&&temp<array[mark-d])

{

array[mark]=array[mark-d];

mark-=d;

}

array[mark]=temp;

}

cout<<"d="<<d<<" ";

print(array,10);

}

}

//快速排序

void quickSort(int \*array,int start,int end)

{

int i=start;

int j=end;

int temp=array[start];

if(i>=j)

return;

while(i<j)

{

while(i<j&&array[j]>temp)

j--;

array[i]=array[j];

while(i<j&&array[i]<=temp)

i++;

array[j]=array[i];

}

array[i]=temp;

quickSort(array,start,i-1);

quickSort(array,i+1,end);

}

//二路归并排序

void mergeSortByREC(int array[],int start,int end)

{

if(start<end)

{

int dis=(end-start)/2;

mergeSortByREC(array,start,start+dis);

mergeSortByREC(array,start+dis+1,end);

merge(array,start,start+dis,start+dis+1,end);

print(array,5);

}

}

void merge(int \*array,int start1,int end1,int start2,int end2)

{

cout<<"start1="<<start1<<"end1="<<end1<<"start2="<<start2<<"end2="<<end2<<" ";

int \*temp=new int[end1-start1+end2-start2+2];

int k=0;

int i1=start1;

int i2=start2;

while(i1<=end1&&i2<=end2)

{

if(array[i1]<array[i2])

temp[k++]=array[i1++];

else

temp[k++]=array[i2++];

}

while(i1<=end1)

temp[k++]=array[i1++];

while(i2<=end2)

temp[k++]=array[i2++];

for(int i=start1,mark=0;mark<k;i++)

array[i]=temp[mark++];

}

//堆排序

void heapSort(int \*array,int length)

{

for(int i=length/2;i>=0;i--)

{

heapAdjust(array,i,length);

}

for(int i=length-1;i>0;i--)

{

swap(array[0],array[i]);

heapAdjust(array,0,i);

}

}

void heapAdjust(int \*array,int i,int length)

{

while(2\*i+1<length)

{

int left=2\*i+1;

if((left+1)<length&&array[left]<array[left+1])

left++;

if(array[i]<array[left])

{

swap(array[i],array[left]);

i=left;

}

else

break;

}

}

//KMP算法

void getNext(char \*p,int \*next)

{

int j,k;

next[0]=-1;

j=0;

k=-1;

while(j<strlen(p)-1)

{

if(k==-1||p[j]==p[k]) //匹配的情况下,p[j]==p[k]

{

j++;

k++;

next[j]=k;

}

else //p[j]!=p[k]

k=next[k];

}

}

int KMPMatch(char \*s,char \*p)

{

int next[100];

int i,j;

i=0;

j=0;

getNext(p,next);

while(i<strlen(s))

{

if(j==-1||s[i]==p[j])

{

i++;

j++;

}

else

{

j=next[j]; //消除了指针i的回溯

}

if(j==strlen(p))

return i-strlen(p);

}

return -1;

}