排序及应用

一 目的

1、进一步掌握线性表顺序存储结构的定义及应用；

2、能够基于顺序表实现各种简单和先进排序方法。

3、能够应用排序算法解决实际问题。

二 内容

1、设一个含有 10 个任意整型数据元素的顺序线性表，分别使用简单选择排序、冒泡以及快速排序算法对其进行排序，并对结果进行测试。

2、从键盘输入或从文件中读取有限个任意同类型的数据生成无序线性表，并用直接插入排序方法、选择排序和堆排序方法对其进行排序；排序后，再从键盘输入一个同类型的数据，插入后使线性表仍然有序，并对结果进行测试。

3、华清电脑销售公司是一家专门销售 ThinkPad 系列电脑的公司，每月均要向总公司汇报不同型号电脑的销售情况，请运用相关知识实现对该公司不同型号电脑销售情况进行排序，并形成汇总表。

4*、编写算法实现单链表上的选择排序，并对结果进行测试。

三 设计说明

1、冒泡排序：冒泡排序是一种交换排序，其的基本思想是：两两比较相邻记录的关键字，如果反序则交换，直到没有反序的记录为止。

2、选择排序：简单选择排序法（Simple Selection Sort）就是通过 n-i 次关键字间的比较，从 n-i+1 个记录中选出关键字最小的记录，并和第 i(1≤i≤n) 个记录交换之。

3、插入排序：直接插入排序（Straight Insertion Sort）的基本操作是将一个记录插入到已经排好序的有序表中，从而得到一个新的、记录数增 1 的有序表。

4、希尔排序：我们将原本有大量记录数的记录进行分组。分割成若干个子序列，此时每个子序列待排序的记录个数就比较少了，然后在这些子序列内分别进行直接插入排序，当整个序列都基本有序时，注意只是基本有序时，再对全体记录进行一次直接插入排序。

5、堆排序：堆排序（Heap Sort）就是利用堆（假设利用大顶堆）进行排序的方法。它的基本思想是，将待排序的序列构造成一个大顶堆。此时，整个序列的最大值就是堆顶的根结点。将它与堆数组的末尾元素进行交换，此时末尾元素就是最大值，然后将剩余的 n-1 个序列重新构造成一个堆，这样就会得到 n 个元素中的次小值。如此反复执行，便能得到一个有序序列了。（大顶堆：在完全二叉

树中，每个结点的值都大于或等于其左右孩子结点的值）

6、归并排序：归并排序（Merging Sort）就是利用归并的思想实现的排序方法。它的原理是假设初始序列含有 n 个记录，则可以看成是 n 个有序的子序列，每个子序列的长度为 1，然后两两归 并，得到⌈n/2⌉（⌈x⌉表示不小于 x 的最小整数）个长度为 2 的有序子序列；再两两归并，…… 如此重复，直至得到一个长度为 n 的有序序列为止，这 种排序方法称为 2 路归并排序。

7、快速排序：快速排序（Quick Sort）的基本思想是通过一趟排序将待排记录分割成独立的两部分，其中一部分记录的关键字均比另一部分记录的关键字小，则可分别对这两部分记录继续进行排序，以达到整个序列有序的目的。

四 功能说明

```
public int size();        //返回表长度

public void clear();      //重置为空表

public boolean isEmpty();  //若为空表返回 true,否则返回 false

public int get(int i);     //返回第 i 个数据元素(返回类型可不同)

public int indexOf(int obj);  //第 1 个与 obj 满足关系 equals()的数据元素位序,这

样的数据元素不存在,

public int getPre(int obj);   //若 obj 是表中的元素,返回它的前驱

public int getNext(int obj);  //若 obj 是表中的元素,返回它的后继

public void insertElementAt(int obj,int i); //在第 i 个位置之前插入新的数据元素

obj,表长度加 1

public int remove(int i);   //删除第 i 个数据元素,并返回其值,表长减 1

public void printAll();

public void insertsort();  //直接插入排序

public void insertsort1(int x);  //一趟插入排序将 x 插入线性表仍然有序

public  int binarysearch(int x);  //对有序表进行两分查找在下表 0 到 n 之间

public void bubblesort();//改进后的冒泡排序
```

```
}
package sj;
class SeqList implements ListIntf {

final int MAXSIZE=20;          //线性表最大长度

public  int[]  elem;           //存放线性表元素的数组

private int len;                //线性表的长度
```

五 调试分析

对于排序我对它的理解为：
1.把需要排序的数分成两份，第一份是有序的，第二份无序。
2.最开始可以把第一个数看成是有序，每次从第二份无序数中按顺序取一个数。
3.把取得数跟有序数中的最后一个比较，如果前者小于后者，找合适位置插入。
4.循环步骤三，则排序结束。

六 测试结果

```
==============================Begin==============================
----构造顺序表----
表长:0
请输入线性表中的元素:1
85
65
32
41
表长:5
>>表中元素>>
 1  85  65  32  41
请选择排序方法：
1-直接插入排序
2-希尔排序
3-冒泡排序
4-快速排序
5-堆排序
6-简单选择排序
请输入数字1 表示要进行排序，0结束排序
1
请输入1-6 的数字选择不同的排序方法：
1
请输入线性表中的元素建立线性表:1
85
65
32
41
98
直接插入方法排序前：
>>表中元素>>
 1  85  65  32  41  98
直接插入方法排序后：
>>表中元素>>
 1  32  41  65  85  98
请输入1-6 的数字选择不同的排序方法：
2
请输入线性表中的元素建立线性表:32
54
65
32
14
35
```

```
希尔排序后：
>>表中元素>>
 14   32   32   35   54   65
请输入1-6 的数字选择不同的排序方法：
4
请输入线性表中的元素建立线性表:36
54
69
48
97
11
快速排序前：
>>表中元素>>
 36   54   69   48   97   11
快速排序后：
>>表中元素>>
 11   36   48   54   69   97
请输入1-6 的数字选择不同的排序方法：
```

七 带注释的源程序

```
package sj;

interface ListIntf{

public int size();        //返回表长度

public void clear();       //重置为空表

public boolean isEmpty();  //若为空表返回 true,否则返回 false

public int get(int i);      //返回第 i 个数据元素(返回类型可不同)

public int indexOf(int obj);  //第 1 个与 obj 满足关系 equals()的数据元素位序,这

样的数据元素不存在,

public int getPre(int obj);    //若 obj 是表中的元素,返回它的前驱

public int getNext(int obj);  //若 obj 是表中的元素,返回它的后继

public void insertElementAt(int obj,int i); //在第 i 个位置之前插入新的数据元素

obj,表长度加 1

public int remove(int i);    //删除第 i 个数据元素,并返回其值,表长减 1

public void printAll();

public void insertsort();  //直接插入排序
```

```
public void insertsort1(int x);  //一趟插入排序将 x 插入线性表仍然有序

public  int binarysearch(int x);  //对有序表进行两分查找在下表 0 到 n 之间

public void bubblesort();//改进后的冒泡排序

}
package sj;
class SeqList implements ListIntf {

final int MAXSIZE=20;          //线性表最大长度

public  int[]  elem;           //存放线性表元素的数组

private int len;                //线性表的长度



public SeqList(){
len=0;
elem= new int[MAXSIZE];}

public SeqList(int n){
if(n<MAXSIZE)  n=MAXSIZE;
len=0;
elem= new int[MAXSIZE];}


//取得表最大长度

public int getMAXSIZE(){
return MAXSIZE;
}
/**

* 实现接口方法

*/

//@Override 返回表长度

public void bubblesort()//改进后的冒泡排序

{ int i,j,t;
int lastexchangeindex;
 i=len-1;
while(i>0)
 {lastexchangeindex=0;
```

```
  for(j=0;j<i;j++)
  if(elem[j+1]<elem[j])
  {t=elem[j];elem[j]=elem[j+1];elem[j+1]=t;
  lastexchangeindex=j; }
  i=lastexchangeindex;}
}



public void insertsort()     //直接插入排序
{ int j;
 for(int i=1;i<len;i++)
{ int x=elem[i];
  for(j=i-1;j>=0;j--)
  {if(x<elem[j]) elem[j+1]=elem[j]; else break;}
  elem[j+1]=x;
 }
}


public void insertsort1(int x){   //有序表尾添加元素 X 使其仍然有序
    int j;
    for( j=len-1;j>=0;j--)
  {if(x<elem[j]) elem[j+1]=elem[j]; else break;}   //寻找插入的位置，然后将元
素插入
  elem[j+1]=x;
  len++;
}

public int size()  {
     return len;
}


public  int binarysearch (int x) {   //对有序表进行两分查找
int  l=0,h=len-1;
while(l<=h)
 {int mid=(l+h)/2;
  if(elem[mid]==x) return mid;   //查找成功
  else
```

```
    if (x<elem[mid]) h=mid-1 ; else l=mid+1;
 }
return -1;   //查找失败
}


public int partition(int low, int high)  //快速排序的第一次划分
{ int  pivotkey;
  pivotkey=elem[low];  //枢轴记录
 while(low<high)
 {while(low<high && elem[high]>=pivotkey)
  high--;
  if (low<high)  {elem[low]=elem[high]; low++;}
  while(low<high  && elem[low]<=pivotkey)
  low++;
  if(low<high) {elem[high]=elem[low]; high--;}
  }
  elem[low]= pivotkey;
  return (low);
}


void qsort(int s,int t)  //快速排序真正实现
{int pivotloc;
if(s<t)
{ pivotloc=partition(s,t);
 qsort(s,pivotloc-1);
 qsort(pivotloc+1,t);
 }
}

void shellsort()            /*  shellsort   */
{ int i,j,k,x;
 k=len/2;
 while(k>=1)
 {for(i=k;i<=len-1;i++)
  {x=elem[i]; j=i-k;
  while(((j>=0) && elem[j]>=x) )
  {elem[j+k]=elem[j]; j=j-k;}
   elem[j+k]=x;
```

```
  }
  k=k/2;
 }
  }

void selectsort()    //简单选择排序

 { int i,j,k; int  t;
  for(i=0;i<len;i++)
  { k=i;
  for(j=i+1;j<len;j++)
  {if (elem[j]<elem[k]) k=j;}
  if (k!=i) {t=elem[i];elem[i]=elem[k];elem[k]=t; }
 }
}



void heap(int i,int m)          /*  调整成堆   */

{int x;
int j;
x=elem[i]; j=2*i+1;
while(j<=m)
{if (j<m) if (elem[j]>elem[j+1]) j++;
if (elem[j]<x){elem[i]=elem[j];i=j;j=2*i+1;}
else j=m+1; }
elem[i]=x;

}        /* 调整结束 */


void heapsort()
{int i,v,w;
for(i=(len-1)/2; i>=0; i--)

heap(i,len-1);                  /* 将所有元素调整成堆*/

//printAll();
for(v=len-1;v>=0;v--)
{ w=elem[0];
  elem[0]=elem[v];
  elem[v]=w;
  heap(0,v-1);
 }
// ll.elem[l.length-v+1]=l.elem[1];
// ll.length =l.length ;
  }
```

```
public void clear(){
    System.out.println("清除数据了！");
}

//@Override 返回第 i 个数据元素的值(返回类型可不同)
public int get(int i) {
if(len==0)     return 0;        //表空,查找失败
if(i<1||i>len) return -1;     //查找位置不合法,查找失败
return elem[i-1];     //查找成功
}

//@Override 若 obj 是表中的元素,返回它的后继
public int getNext(int obj) {
int i=indexOf(obj);
if(i==-1)     return -1;            // 指定元素不存在
else if(i==len)      return 1;      // 指定元素位于最后一位,不存在后继元素
return elem[i];      //查找后继成功,返回后继
}

//@Override 若 obj 是表中的元,返回它的前驱
public int getPre(int obj) {
int i=indexOf(obj);
if(i==-1)     return -1;           //指定元素不存在
else
     if(i==1)      return 1;   //指定元素位于第一位,不存在前驱元素
return elem[i-1];  //查找前驱成功,返回前驱
}
```

```
//@Override 第 1 个与 obj 满足关系 equals()的数据元素的位序,若这样的数据元素不存在,
则返回值为-1(obj 的类型根据实际不同)

public int indexOf(int obj) {
for(int i=0;i<len;i++){
  if( elem[i]==obj)
   return i+1;                 //查找成功,返回位
     }
return -1;         //查找失败
}

//@Override 在第 i 个位置之前插入新的数据元素 obj,表长度加 1
public void insertElementAt(int obj, int i) {
 if(len==MAXSIZE){                    //判断线性表的存储空间是否已满
System.out.println("--溢出--");
  return;
 }
else if(i<1||i>len+1){
  System.out.println("--插入位置非法--");
return;                             //插入位置非法
  }
for(int j=len-1;j>=i-1;j--)
 elem[j+1]=elem[j];
 elem[i-1]=obj;           //插入元素
 len++;                   //表长加 1
 return;
}

//@Override 若为空表,则返回 true,否则返回 false

public boolean isEmpty() {
```

```
if(len==0)      return true;   //表空,返回 true

return false;                 //表非空,返回 false

}

//@Override 删除第 i 个数据元素,并返回其值,表长度减 1

public int remove(int i) {
int obj;
if(i<1||i>len){

System.out.println("--删除位置非法--");

return -1;    }

 obj=elem[i-1];                      //被删除的值

for(int j=i-1;j<len-1;j++){
 elem[j]=elem[j+1];    }
 len--;

return obj;          //返回被删除的值

}



//打印表中元素

public void printAll(){
if(len==0){

  System.out.println("--表空--");

 return;    }

 System.out.println(">>表中元素>>");

for(int i=0;i<len;i++)
System.out.print(" "+elem[i]+" ");
System.out.println();
}
}
package sj;

import java.util.*;

//线性表的抽象数据类型定义

public class SortTest {
```

```
/**
* @param args
*/

public static void main(String[] args) {

System.out.println("==============================Begin==================
==========");

System.out.println("----构造顺序表----");

int  obj;   int  n, k;
Scanner input=new  Scanner(System.in);
SeqList sl=new SeqList();

System.out.println("表长:"+sl.size());

System.out.print("请输入线性表中的元素:");

for(int i=0;i<5;i++){                    //构造 5 个元素的顺序表


obj=input.nextInt();
sl.insertElementAt(obj, sl.size()+1);
}

System.out.println("表长:"+sl.size());

sl.printAll();
/*
sl.heapsort();
sl.printAll();

sl.heapsort();
sl.insertsort();

System.out.print("堆排序后的结果为：");

sl.printAll();
sl.shellsort();

System.out.print("希尔排序后的结果为：");

sl.printAll();
 sl.insertsort();

 System.out.print("直接插入排序后的结果为：");

sl.printAll();
sl.qsort(0, sl.size()-1);
```

```
System.out.print("快速排序后的结果为：");

sl.printAll();
sl.insertsort1(-21);
sl.bubblesort();

System.out.print("排序后的结果为：");

sl.printAll();

System.out.print("输入元素进行两分查找：");

obj=input.nextInt();
k=sl.binarysearch(obj);

System.out.println("进行两分查找的结果为："+k);

System.out.println("表长"+sl.size());


sl.printAll();   */

System.out.println("请选择排序方法：");

System.out.println("1-直接插入排序");

System.out.println("2-希尔排序");

System.out.println("3-冒泡排序");

System.out.println("4-快速排序");

System.out.println("5-堆排序");

System.out.println("6-简单选择排序");

Scanner s = new Scanner(System.in);

System.out.println("请输入数字 1 表示要进行排序，0 结束排序");

int flag = s.nextInt();
while(flag!=0){

    System.out.println("请输入 1-6 的数字选择不同的排序方法：");

    int xz = s.nextInt();

    SeqList L = new SeqList(); //建立顺序表

 switch (xz) {
       case 1:
```

```
        System.out.print("请输入线性表中的元素建立线性表:");
        for (int i = 0; i < 6; i++) {
          k = s.nextInt();
             L.insertElementAt(k, L.size()+1);
             }
             System.out.println("直接插入方法排序前：");
             L.printAll();
          L.insertsort();
             System.out.println("直接插入方法排序后：");
          L.printAll();
              break;            //直接插入排序
case 2:
     System.out.print("请输入线性表中的元素建立线性表:");
     for (int i = 0; i < 6; i++) {
          k = s.nextInt();
         L.insertElementAt(k, L.size()+1);}
        System.out.println("希尔排序前：");
        L.printAll();
      L.shellsort();
        System.out.println("希尔排序后：");
      L.printAll();
          break;            //直接插入排序
case 3:
     System.out.print("请输入线性表中的元素建立线性表:");
     for (int i = 0; i < 6; i++) {
         k = s.nextInt();
         L.insertElementAt(k, L.size()+1);;}
        System.out.println("冒泡排序前：");
        L.printAll();
      L.bubblesort();
        System.out.println("冒泡排序后：");
      L.printAll();
          break;            //直接插入排序冒泡排序
```

```
case 4:
  System.out.print("请输入线性表中的元素建立线性表:");
  for (int i = 0; i < 6; i++) {
        k = s.nextInt();
        L.insertElementAt(k, L.size()+1);}
       System.out.println("快速排序前：");
       L.printAll();
     L.qsort(0, L.size()-1);
       System.out.println("快速排序后：");
    L.printAll();
        break;            //直接插入排序
case 5:
  System.out.print("请输入线性表中的元素建立线性表:");
  for (int i = 0; i < 6; i++) {
        k = s.nextInt();
        L.insertElementAt(k, L.size()+1);}
       System.out.println("堆排序前：");
       L.printAll();
     L.heapsort();
       System.out.println("堆排序后：");
     L.printAll();
        break;            //直接插入排序
case 6:
  System.out.print("请输入线性表中的元素建立线性表:");
  for (int i = 0; i < 6; i++) {
        k = s.nextInt();
         L.insertElementAt(k, L.size()+1);}
       System.out.println("简单选择排序前：");
       L.printAll();
     L.selectsort();
       System.out.println("简单选择排序后：");
     L.printAll();
        break;            //直接插入排序
```

```
        }
//System.out.println("==============================end==================
==========");
}
}
}
```