常用编程算法

算法 排序算法

算法一：快速排序算法

快速排序是由东尼·霍尔所发展的一种排序算法。在平均状况下，排序 n 个项目要O(n log n)次比较。在最坏状况下则需要O(n2)次比较，但这种状况并不常见。事实上，快速排序通常明显比其他O(n log n) 算法更快，因为它的内部循环（inner loop）可以在大部分的架构上很有效率地被实现出来。
快速排序使用分治法（Divide and conquer）策略来把一个串行（list）分为两个子串行（sub-lists）。

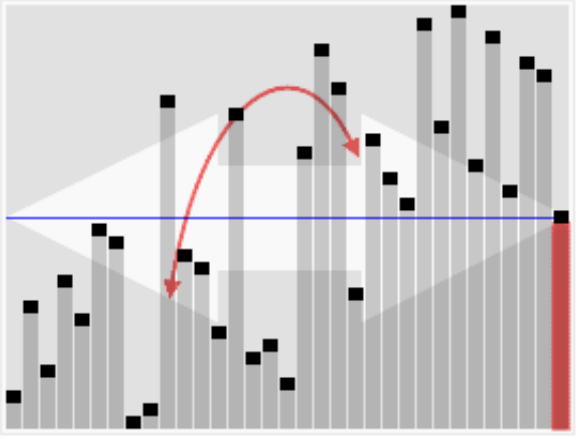
算法思想

基本思想：选择一个基准元素,通常选择第一个元素或者最后一个元素,通过一趟扫描，将待排序列分成两部分,一部分比基准元素小,一部分大于等于基准元素,此时基准元素在其排好序后的正确位置,然后再用同样的方法递归地排序划分的两部分。

算法步骤：

1 .从数列中挑出一个元素，称为 “基准”（pivot），
2 .重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作。
3 .递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

递归的最底部情形，是数列的大小是零或一，也就是永远都已经被排序好了。虽然一直递归下去，但是这个算法总会退出，因为在每次的迭代（iteration）中，它至少会把一个元素摆到它最后的位置去。



代码实现

```
/*
*快速排序
*/
public class quickSort {
  inta[]=
{49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,3
4,15,35,25,53,51};
public quickSort(){
    quick(a);
    for(int i=0;i<a.length;i++){
       System.out.println(a[i]);
    }
}
public int getMiddle(int[] list, int low, int high) {
            int tmp =list[low];    //数组的第一个作为中轴
            while (low < high){
                while (low < high&& list[high] >= tmp) {
                   high--;
                }
                list[low] =list[high];   //比中轴小的记录移到低端
                while (low < high&& list[low] <= tmp) {
                    low++;
                }
                list[high] =list[low];   //比中轴大的记录移到高端
            }
           list[low] = tmp;              //中轴记录到尾
            return low;                  //返回中轴的位置
}

public void quickSort(int[] list, int low, int high) {
            if (low < high){
               int middle =getMiddle(list, low, high);  //将list数
组进行一分为二
                quickSort(list, low, middle - 1);        //对低字表进
行递归排序
                quickSort(list,middle + 1, high);        //对高字表进
行递归排序
            }
}

public void quick(int[] a2) {
            if (a2.length > 0) {    //查看数组是否为空
                quickSort(a2,0, a2.length - 1);
            }
}
}
```

算法二：堆排序算法

堆排序（Heapsort）是指利用堆这种数据结构所设计的一种排序算法。堆积是一个近似完全二叉树的结构，并同时满足堆积的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。
堆排序的平均时间复杂度为O(nlogn)。

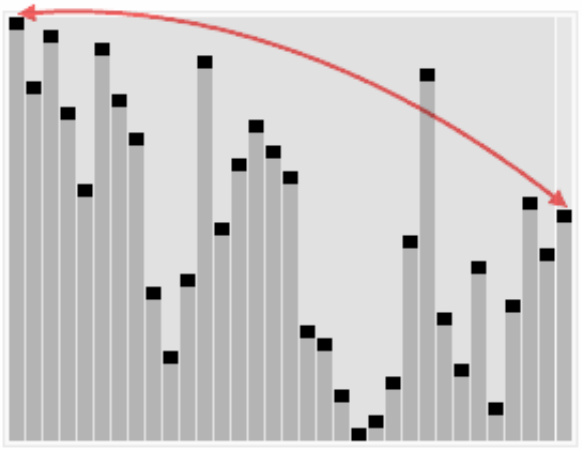
算法思想

基本思想：堆排序是一种树形选择排序，是对直接选择排序的有效改进。

堆的定义如下：具有n个元素的序列（h1,h2,…,hn),当且仅当满足（hi>=h2i,hi>=2i+1）或（hi<=h2i,hi<=2i+1）(i=1,2,…,n/2)时称之为堆。在这里只讨论满足前者条件的堆。由堆的定义可以看出，堆顶元素（即第一个元素）必为最大项（大顶堆）。完全二叉树可以很直观地表示堆的结构。堆顶为根，其它为左子树、右子树。初始时把要排序的数的序列看作是一棵顺序存储的二叉树，调整它们的存储序，使之成为一个堆，这时堆的根节点的数最大。然后将根节点与堆的最后一个节点交换。然后对前面(n-1)个数重新调整使之成为堆。依此类推，直到只有两个节点的堆，并对它们作交换，最后得到有n个节点的有序序列。从算法描述来看，堆排序需要两个过程，一是建立堆，二是堆顶与堆的最后一个元素交换位置。所以堆排序有两个函数组成。一是建堆的渗透函数，二是反复调用渗透函数实现排序的函数

算法步骤：

1.创建一个堆H[0..n-1]
2.把堆首（最大值）和堆尾互换
3.把堆的尺寸缩小1，并调用shift_down(0),目的是把新的数组顶端数据调整到相应位置
4.重复步骤2，直到堆的尺寸为1



代码实现

```
import java.util.Arrays;

public class HeapSort {
    inta[]=
{49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,3
4,15,35,25,53,51};
```

```
    public  HeapSort(){
        heapSort(a);
    }

    public void heapSort(int[] a){
        System.out.println("开始排序");
        int arrayLength=a.length;
        //循环建堆
        for(int i=0;i<arrayLength-1;i++){
            //建堆
            buildMaxHeap(a,arrayLength-1-i);
            //交换堆顶和最后一个元素
            swap(a,0,arrayLength-1-i);
            System.out.println(Arrays.toString(a));
        }
    }

    private  void swap(int[] data, int i, int j) {
        // TODO Auto-generated method stub
        int tmp=data[i];
        data[i]=data[j];
        data[j]=tmp;
    }

    //对data数组从0到lastIndex建大顶堆
    private void buildMaxHeap(int[] data, int lastIndex) {
        // TODO Auto-generated method stub
        //从lastIndex处节点（最后一个节点）的父节点开始
        for(int i=(lastIndex-1)/2;i>=0;i--){
            //k保存正在判断的节点
            int k=i;
            //如果当前k节点的子节点存在
            while(k*2+1<=lastIndex){
                //k节点的左子节点的索引
                int biggerIndex=2*k+1;
                //如果biggerIndex小于lastIndex，即biggerIndex+1代表的
k节点的右子节点存在
                if(biggerIndex<lastIndex){
                    //若果右子节点的值较大
                    if(data[biggerIndex]<data[biggerIndex+1]){
                        //biggerIndex总是记录较大子节点的索引
                        biggerIndex++;
                    }
                }

                //如果k节点的值小于其较大的子节点的值
                if(data[k]<data[biggerIndex]){
                    //交换他们
                    swap(data,k,biggerIndex);
```

```
                    //将biggerIndex赋予k，开始while循环的下一次循环，重新保证k节点的值大于其左右子节点的值
                    k=biggerIndex;
                }else{
                    break;
                }
            }
        }
    }
}
```

算法三：归并排序

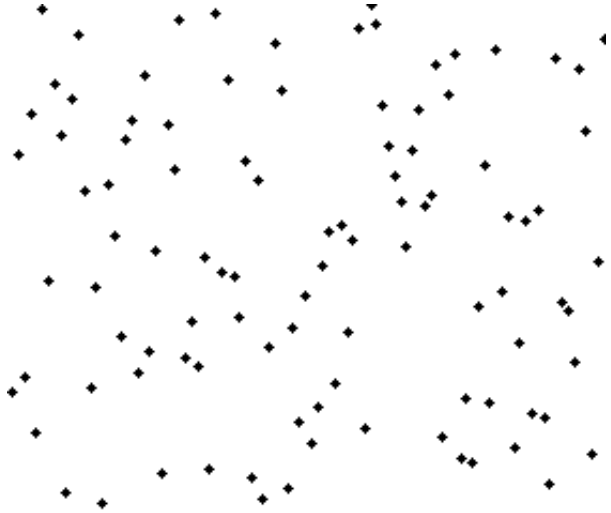
归并排序（Merge sort，台湾译作：合并排序）是建立在归并操作上的一种有效的排序算法。该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。

算法思想

基本思想：归并（Merge）排序法是将两个（或两个以上）有序表合并成一个新的有序表，即把待排序序列分为若干个子序列，每个子序列是有序的。然后再把有序子序列合并为整体有序序列

算法步骤

1. 申请空间，使其大小为两个已经排序序列之和，该空间用来存放合并后的序列
2. 设定两个指针，最初位置分别为两个已经排序序列的起始位置
3. 比较两个指针所指向的元素，选择相对小的元素放入到合并空间，并移动指针到下一位置
4. 重复步骤3直到某一指针达到序列尾
5. 将另一序列剩下的所有元素直接复制到合并序列尾



实现代码

```
import java.util.Arrays;
```

```
public class mergingSort {

inta[]=
{49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,3
4,15,35,25,53,51};

public mergingSort(){
    sort(a,0,a.length-1);
    for(int i=0;i<a.length;i++)
       System.out.println(a[i]);
}

public void sort(int[] data, int left, int right) {
    // TODO Auto-generatedmethod stub
    if(left<right){
        //找出中间索引
        int center=(left+right)/2;
        //对左边数组进行递归
        sort(data,left,center);
        //对右边数组进行递归
        sort(data,center+1,right);
        //合并
        merge(data,left,center,right);
    }

}

public void merge(int[] data, int left, int center, int right) {
    // TODO Auto-generatedmethod stub
    int [] tmpArr=newint[data.length];
    int mid=center+1;
    //third记录中间数组的索引
    int third=left;
    int tmp=left;
    while(left<=center&&mid<=right){
        //从两个数组中取出最小的放入中间数组
        if(data[left]<=data[mid]){
            tmpArr[third++]=data[left++];
        }else{
            tmpArr[third++]=data[mid++];
        }
    }
    //剩余部分依次放入中间数组
    while(mid<=right){
        tmpArr[third++]=data[mid++];
    }
    while(left<=center){
        tmpArr[third++]=data[left++];
    }
```

```
        //将中间数组中的内容复制回原数组
        while(tmp<=right){
            data[tmp]=tmpArr[tmp++];
        }
        System.out.println(Arrays.toString(data));
    }
    }
```

算法四：插入排序

算法思想

基本思想：在要排序的一组数中，假设前面(n-1)[n>=2] 个数已经是排好顺序的，现在要把第n个数插到前面的有序数中，使得这n个数也是排好顺序的。如此反复循环，直到全部排好顺序。

代码实现

```
package com.suanfa;

public class insertSort {

public insertSort(){
    inta[]=
{49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,3
4,15,35,25,53,51};
    int temp=0;
    for(int i=1;i<a.length;i++){
       int j=i-1;
       temp=a[i];
       for(;j>=0&&temp<a[j];j--){
           a[j+1]=a[j];  //将大于temp的值整体后移一个单位
       }
       a[j+1]=temp;
    }
    for(int i=0;i<a.length;i++){
       System.out.println(a[i]);
    }
}
```

算法五：希尔排序（最小增量排序）

算法思想

基本思想：算法先将要排序的一组数按某个增量d（n/2,n为要排序数的个数）分成若干组，每组中记录的下标相差d.对每组中全部元素进行直接插入排序，然后再用一个较小的增量（d/2）对它进行分组，在每组中再进行直接插入排序。当增量减到1时，进行直接插入排序后，排序完成。

算法实现

```
public class shellSort {

public shellSort(){

    int a[]={1,54,6,3,78,34,12,45,56,100};
    double d1=a.length;
    int temp=0;

    while(true){
       d1= Math.ceil(d1/2);
       int d=(int) d1;
       for(int x=0;x<d;x++){

           for(int i=x+d;i<a.length;i+=d){
              int j=i-d;
              temp=a[i];
              for(;j>=0&&temp<a[j];j-=d){
                   a[j+d]=a[j];
              }
              a[j+d]=temp;
           }
       }

       if(d==1){
           break;
       }

    for(int i=0;i<a.length;i++){
       System.out.println(a[i]);
    }
}
```

算法六：冒泡排序

算法思想

基本思想：在要排序的一组数中，对当前还未排好序的范围内的全部数，自上而下对相邻的两个数依次进行比较和调整，让较大的数往下沉，较小的往上冒。即：每当两相邻的数比较后发现它们的排序与排序要求相反时，就将它们互换。

代码实现

```
public class bubbleSort {

public bubbleSort(){
     inta[]=
{49,38,65,97,76,13,27,49,78,34,12,64,5,4,62,99,98,54,56,17,18,23,3
4,15,35,25,53,51};
    int temp=0;
    for(int i=0;i<a.length-1;i++){
       for(int j=0;j<a.length-1-i;j++){
         if(a[j]>a[j+1]){
           temp=a[j];
           a[j]=a[j+1];
           a[j+1]=temp;
         }
       }
    }

    for(int i=0;i<a.length;i++){
       System.out.println(a[i]);
   }
}
```

算法七：简单选择排序

算法思想

基本思想：在要排序的一组数中，选出最小的一个数与第一个位置的数交换；然后在剩下的数当中再找最小的与第二个位置的数交换，如此循环到倒数第二个数和最后一个数比较为止。

代码实现

```
public class selectSort {
    public selectSort(){
        int a[]={1,54,6,3,78,34,12,45};
        int position=0;
        for(int i=0;i<a.length;i++){
            int j=i+1;
            position=i;
            int temp=a[i];
            for(;j<a.length;j++){
                if(a[j]<temp){
                    temp=a[j];
                    position=j;
                }
            }
            a[position]=a[i];
            a[i]=temp;
        }

        for(int i=0;i<a.length;i++)
            System.out.println(a[i]);
    }
}
```