**排序算法总结**

[](https://www.jianshu.com/u/6e7c82da2a5e)

[梦工厂](https://www.jianshu.com/u/6e7c82da2a5e) 关注

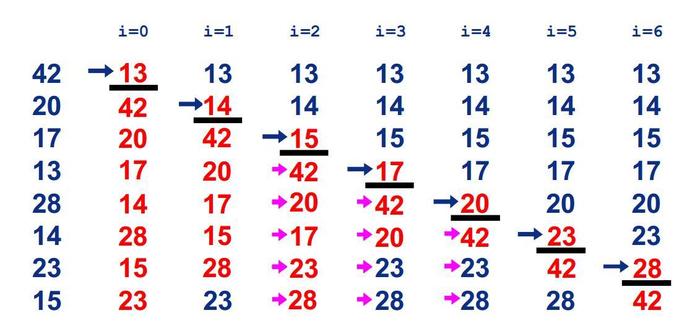
2015.08.31 17:50\* 字数 2109 阅读 43501评论 33喜欢 765赞赏 2

| **排序算法** | **平均时间复杂度** |
| --- | --- |
| 冒泡排序 | O(n2) |
| 选择排序 | O(n2) |
| 插入排序 | O(n2) |
| 希尔排序 | O(n1.5) |
| 快速排序 | O(N\*logN) |
| 归并排序 | O(N\*logN) |
| 堆排序 | O(N\*logN) |
| 基数排序 | O(d(n+r)) |

**一. 冒泡排序(BubbleSort)**

1. **基本思想：**两个数比较大小，较大的数下沉，较小的数冒起来。
2. **过程：**

* 比较相邻的两个数据，如果第二个数小，就交换位置。
* 从后向前两两比较，一直到比较最前两个数据。最终最小数被交换到起始的位置，这样第一个最小数的位置就排好了。
* 继续重复上述过程，依次将第2.3...n-1个最小数排好位置。



冒泡排序

1. **平均时间复杂度：**O(n2)
2. **java代码实现：**

public static void BubbleSort(int [] arr){

int temp;//临时变量

for(int i=0; i<arr.length-1; i++){ //表示趟数，一共arr.length-1次。

for(int j=arr.length-1; j>i; j--){

if(arr[j] < arr[j-1]){

temp = arr[j];

arr[j] = arr[j-1];

arr[j-1] = temp;

}

}

}

}

1. **优化：**

* **针对问题：**  
  数据的顺序排好之后，冒泡算法仍然会继续进行下一轮的比较，直到arr.length-1次，后面的比较没有意义的。
* **方案：**  
  设置标志位flag，如果发生了交换flag设置为true；如果没有交换就设置为false。  
  这样当一轮比较结束后如果flag仍为false，即：这一轮没有发生交换，说明数据的顺序已经排好，没有必要继续进行下去。

public static void BubbleSort1(int [] arr){

int temp;//临时变量

boolean flag;//是否交换的标志

for(int i=0; i<arr.length-1; i++){ //表示趟数，一共arr.length-1次。

flag = false;

for(int j=arr.length-1; j>i; j--){

if(arr[j] < arr[j-1]){

temp = arr[j];

arr[j] = arr[j-1];

arr[j-1] = temp;

flag = true;

}

}

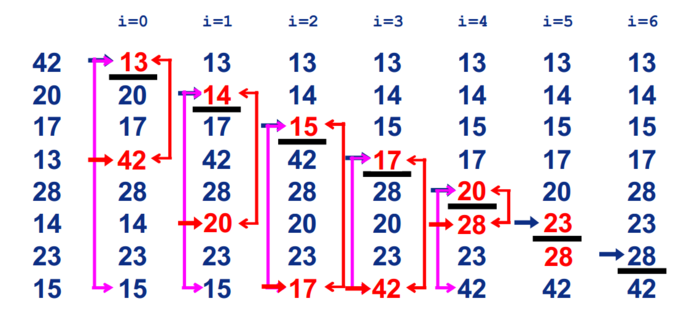
if(!flag) break;

}

}

**二. 选择排序(SelctionSort)**

1. **基本思想：**  
   在长度为N的无序数组中，第一次遍历n-1个数，找到最小的数值与第一个元素交换；  
   第二次遍历n-2个数，找到最小的数值与第二个元素交换；  
   。。。  
   第n-1次遍历，找到最小的数值与第n-1个元素交换，排序完成。
2. **过程：**



选择排序

1. **平均时间复杂度：**O(n2)
2. **java代码实现：**

public static void select\_sort(int array[],int lenth){

for(int i=0;i<lenth-1;i++){

int minIndex = i;

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

if(array[j]<array[minIndex]){

minIndex = j;

}

}

if(minIndex != i){

int temp = array[i];

array[i] = array[minIndex];

array[minIndex] = temp;

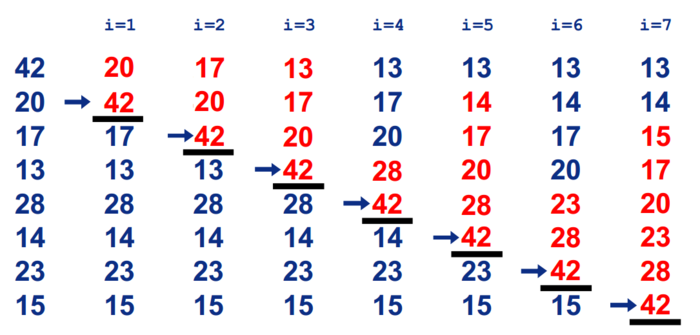
}

}

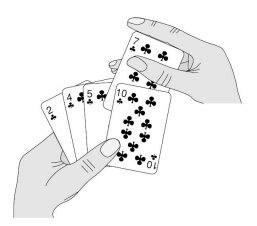
}

**三. 插入排序(Insertion Sort)**

1. **基本思想：**  
   在要排序的一组数中，假定前n-1个数已经排好序，现在将第n个数插到前面的有序数列中，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。
2. **过程：**



插入排序



相同的场景

1. **平均时间复杂度：**O(n2)
2. **java代码实现：**

public static void insert\_sort(int array[],int lenth){

int temp;

for(int i=0;i<lenth-1;i++){

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

if(array[j] < array[j-1]){

temp = array[j-1];

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

array[j] = temp;

}else{ //不需要交换

break;

}

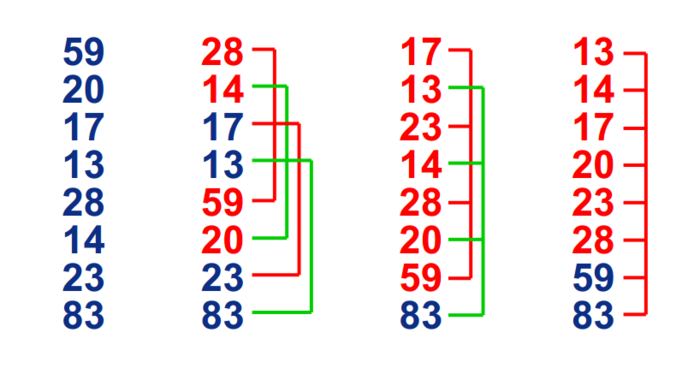
}

}

}

**四. 希尔排序(Shell Sort)**

1. **前言：**  
   数据序列1： 13-17-20-42-28 利用插入排序，13-17-20-28-42. Number of swap:1;  
   数据序列2： 13-17-20-42-14 利用插入排序，13-14-17-20-42. Number of swap:3;  
   如果数据序列基本有序，使用插入排序会更加高效。
2. **基本思想：**  
   在要排序的一组数中，根据某一增量分为若干子序列，并对子序列分别进行插入排序。  
   然后逐渐将增量减小,并重复上述过程。直至增量为1,此时数据序列基本有序,最后进行插入排序。
3. **过程：**



希尔排序

1. **平均时间复杂度：**
2. **java代码实现：**

public static void shell\_sort(int array[],int lenth){

int temp = 0;

int incre = lenth;

while(true){

incre = incre/2;

for(int k = 0;k<incre;k++){ //根据增量分为若干子序列

for(int i=k+incre;i<lenth;i+=incre){

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

if(array[j]<array[j-incre]){

temp = array[j-incre];

array[j-incre] = array[j];

array[j] = temp;

}else{

break;

}

}

}

}

if(incre == 1){

break;

}

}

}

**五. 快速排序(Quicksort)**

1. **基本思想：（分治）**

* 先从数列中取出一个数作为key值；
* 将比这个数小的数全部放在它的左边，大于或等于它的数全部放在它的右边；
* 对左右两个小数列重复第二步，直至各区间只有1个数。

1. **辅助理解：**[**挖坑填数**](https://link.jianshu.com/?t=http://blog.csdn.net/morewindows/article/details/6684558)

* 初始时 i = 0; j = 9; key=72  
  由于已经将a[0]中的数保存到key中，可以理解成在数组a[0]上挖了个坑，可以将其它数据填充到这来。  
  从j开始向前找一个比key小的数。当j=8，符合条件，**a[0] = a[8] ; i++ ;** 将a[8]挖出再填到上一个坑a[0]中。  
  这样一个坑a[0]就被搞定了，但又形成了一个新坑a[8]，这怎么办了？简单，再找数字来填a[8]这个坑。  
  这次从i开始向后找一个大于key的数，当i=3，符合条件，**a[8] = a[3] ; j-- ;** 将a[3]挖出再填到上一个坑中。

数组：72 - 6 - 57 - 88 - 60 - 42 - 83 - 73 - 48 - 85

0 1 2 3 4 5 6 7 8 9

* 此时 i = 3; j = 7; key=72  
  再重复上面的步骤，先从后向前找，再从前向后找。  
  从j开始向前找，当j=5，符合条件，将a[5]挖出填到上一个坑中，**a[3] = a[5]; i++;**  
  从i开始向后找，当i=5时，由于i==j退出。  
  此时，i = j = 5，而a[5]刚好又是上次挖的坑，因此将key填入a[5]。

数组：48 - 6 - 57 - 88 - 60 - 42 - 83 - 73 - 88 - 85

0 1 2 3 4 5 6 7 8 9

* 可以看出a[5]前面的数字都小于它，a[5]后面的数字都大于它。因此再对a[0…4]和a[6…9]这二个子区间重复上述步骤就可以了。

数组：48 - 6 - 57 - 42 - 60 - 72 - 83 - 73 - 88 - 85

0 1 2 3 4 5 6 7 8 9

1. **平均时间复杂度：**O(N\*logN)
2. **代码实现：**

public static void quickSort(int a[],int l,int r){

if(l>=r)

return;

int i = l; int j = r; int key = a[l];//选择第一个数为key

while(i<j){

while(i<j && a[j]>=key)//从右向左找第一个小于key的值

j--;

if(i<j){

a[i] = a[j];

i++;

}

while(i<j && a[i]<key)//从左向右找第一个大于key的值

i++;

if(i<j){

a[j] = a[i];

j--;

}

}

//i == j

a[i] = key;

quickSort(a, l, i-1);//递归调用

quickSort(a, i+1, r);//递归调用

}

key值的选取可以有多种形式，例如中间数或者随机数，分别会对算法的复杂度产生不同的影响。

**六. 归并排序(Merge Sort)**

1. **基本思想：**[参考](https://link.jianshu.com/?t=http://blog.csdn.net/morewindows/article/details/6678165" \t "_blank)  
   归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的一个非常典型的应用。  
   首先考虑下如何将2个有序数列合并。这个非常简单，只要从比较2个数列的第一个数，谁小就先取谁，取了后就在对应数列中删除这个数。然后再进行比较，如果有数列为空，那直接将另一个数列的数据依次取出即可。

//将有序数组a[]和b[]合并到c[]中

void MemeryArray(int a[], int n, int b[], int m, int c[])

{

int i, j, k;

i = j = k = 0;

while (i < n && j < m)

{

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

c[k++] = a[i++];

else

c[k++] = b[j++];

}

while (i < n)

c[k++] = a[i++];

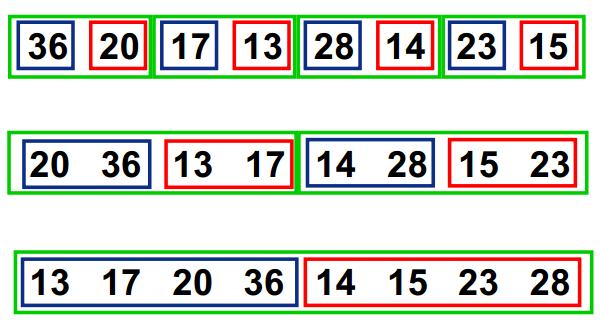
while (j < m)

c[k++] = b[j++];

}

解决了上面的合并有序数列问题，再来看归并排序，其的基本思路就是将数组分成2组A，B，如果这2组组内的数据都是有序的，那么就可以很方便的将这2组数据进行排序。如何让这2组组内数据有序了？  
可以将A，B组各自再分成2组。依次类推，当分出来的小组只有1个数据时，可以认为这个小组组内已经达到了有序，然后再合并相邻的2个小组就可以了。这样通过**先递归的分解数列**，**再合并数列**就完成了归并排序。

1. **过程：**



归并排序

1. **平均时间复杂度：**O(NlogN)  
   归并排序的效率是比较高的，设数列长为N，将数列分开成小数列一共要logN步，每步都是一个合并有序数列的过程，时间复杂度可以记为O(N)，故一共为O(N\*logN)。
2. **代码实现：**

public static void merge\_sort(int a[],int first,int last,int temp[]){

if(first < last){

int middle = (first + last)/2;

merge\_sort(a,first,middle,temp);//左半部分排好序

merge\_sort(a,middle+1,last,temp);//右半部分排好序

mergeArray(a,first,middle,last,temp); //合并左右部分

}

}

//合并 ：将两个序列a[first-middle],a[middle+1-end]合并

public static void mergeArray(int a[],int first,int middle,int end,int temp[]){

int i = first;

int m = middle;

int j = middle+1;

int n = end;

int k = 0;

while(i<=m && j<=n){

if(a[i] <= a[j]){

temp[k] = a[i];

k++;

i++;

}else{

temp[k] = a[j];

k++;

j++;

}

}

while(i<=m){

temp[k] = a[i];

k++;

i++;

}

while(j<=n){

temp[k] = a[j];

k++;

j++;

}

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

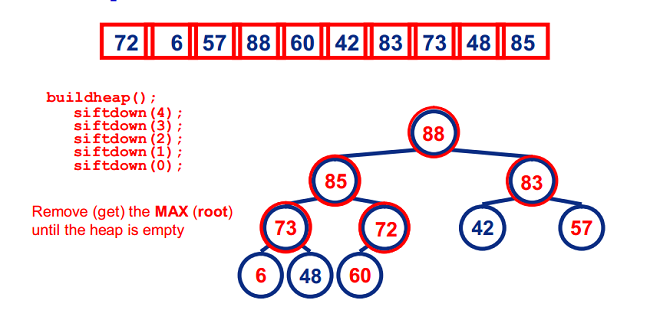
a[first + ii] = temp[ii];

}

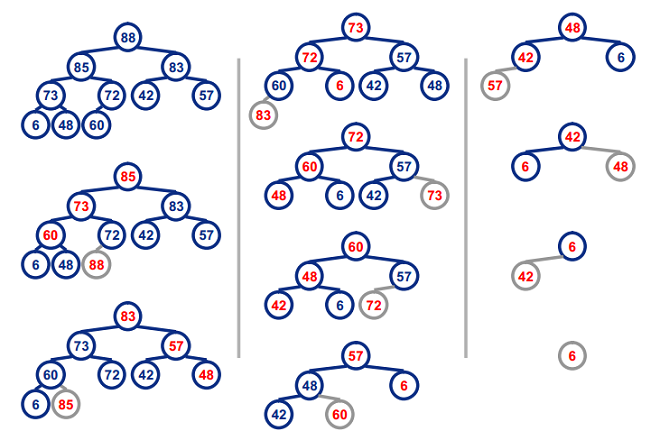
}

**七. 堆排序(HeapSort)**

1. **基本思想：**



1. \*\* 图示：\*\* （88,85,83,73,72,60,57,48,42,6）



Heap Sort

1. **平均时间复杂度：**O(NlogN)  
   由于每次重新恢复堆的时间复杂度为O(logN)，共N - 1次重新恢复堆操作，再加上前面建立堆时N / 2次向下调整，每次调整时间复杂度也为O(logN)。二次操作时间相加还是O(N \* logN)。
2. **java代码实现：**

//构建最小堆

public static void MakeMinHeap(int a[], int n){

for(int i=(n-1)/2 ; i>=0 ; i--){

MinHeapFixdown(a,i,n);

}

}

//从i节点开始调整,n为节点总数 从0开始计算 i节点的子节点为 2\*i+1, 2\*i+2

public static void MinHeapFixdown(int a[],int i,int n){

int j = 2\*i+1; //子节点

int temp = 0;

while(j<n){

//在左右子节点中寻找最小的

if(j+1<n && a[j+1]<a[j]){

j++;

}

if(a[i] <= a[j])

break;

//较大节点下移

temp = a[i];

a[i] = a[j];

a[j] = temp;

i = j;

j = 2\*i+1;

}

}

public static void MinHeap\_Sort(int a[],int n){

int temp = 0;

MakeMinHeap(a,n);

for(int i=n-1;i>0;i--){

temp = a[0];

a[0] = a[i];

a[i] = temp;

MinHeapFixdown(a,0,i);

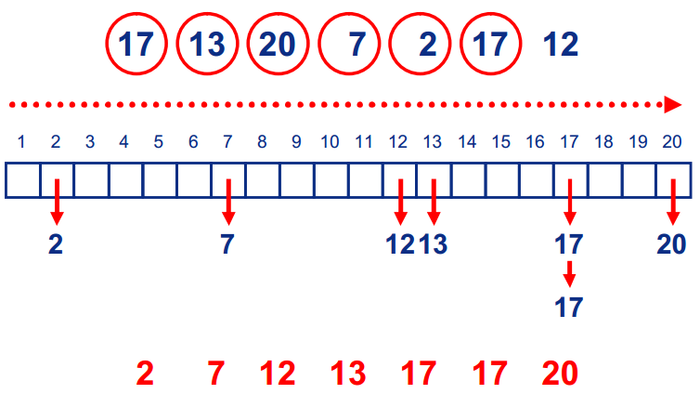
}

}

**八. 基数排序(RadixSort)**

**BinSort**

1. **基本思想：**  
   BinSort想法非常简单，首先创建数组A[MaxValue]；然后将每个数放到相应的位置上（例如17放在下标17的数组位置）；最后遍历数组，即为排序后的结果。
2. \*\* 图示：\*\*

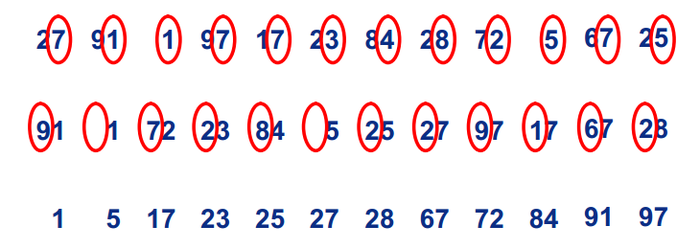


BinSort

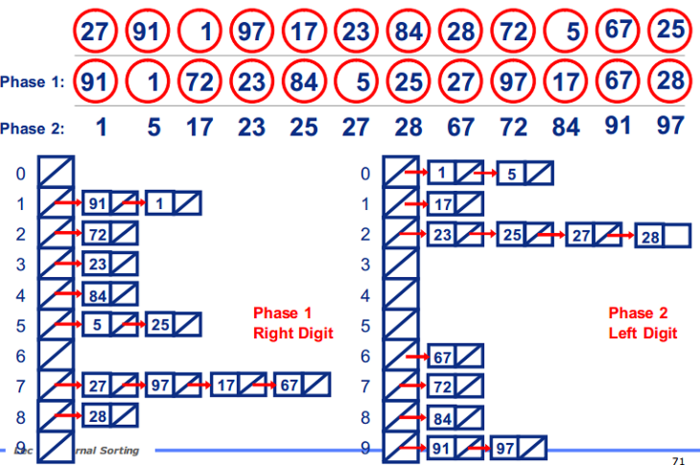
1. \*\* 问题：\*\*  
   当序列中存在较大值时，BinSort 的排序方法会浪费大量的空间开销。

**RadixSort**

1. **基本思想：**  
   基数排序是在BinSort的基础上，通过基数的限制来减少空间的开销。
2. **过程：**



过程1



过程2

（1）首先确定基数为10，数组的长度也就是10.每个数34都会在这10个数中寻找自己的位置。  
（2）不同于BinSort会直接将数34放在数组的下标34处，基数排序是将34分开为3和4，第一轮排序根据最末位放在数组的下标4处，第二轮排序根据倒数第二位放在数组的下标3处，然后遍历数组即可。

1. **java代码实现：**

public static void RadixSort(int A[],int temp[],int n,int k,int r,int cnt[]){

//A:原数组

//temp:临时数组

//n:序列的数字个数

//k:最大的位数2

//r:基数10

//cnt:存储bin[i]的个数

for(int i=0 , rtok=1; i<k ; i++ ,rtok = rtok\*r){

//初始化

for(int j=0;j<r;j++){

cnt[j] = 0;

}

//计算每个箱子的数字个数

for(int j=0;j<n;j++){

cnt[(A[j]/rtok)%r]++;

}

//cnt[j]的个数修改为前j个箱子一共有几个数字

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

cnt[j] = cnt[j-1] + cnt[j];

}

for(int j = n-1;j>=0;j--){ //重点理解

cnt[(A[j]/rtok)%r]--;

temp[cnt[(A[j]/rtok)%r]] = A[j];

}

for(int j=0;j<n;j++){

A[j] = temp[j];

}

}

}

[2015-08-31]