原 C语言实现8种排序

2017年02月24日 22:26:01 J_小浩子 阅读数 21353 更多

最近要开始准备春招了，没什么时间学习spring，得忙着刷题。这两天复习排序，综合网上和书上的资料，整理了下面8种排序算法的实现。基于C语言的，java版本很快就出来

具体代码：

```c
#include<stdio.h>
#include<stdlib.h>
//冒泡排序
void bubleSort(int data[], int n);
//快速排序
void quickSort(int data[], int low, int high);
int findPos(int data[], int low, int high);
//插入排序
void bInsertSort(int data[], int n);
//希尔排序
void shellSort(int data[], int n);
//选择排序
void selectSort(int data[], int n);
//堆排序
void heapSort(int data[], int n);
void swap(int data[], int i, int j);
void heapAdjust(int data[], int i, int n);
//归并排序
void mergeSort(int data[], int first, int last);
void merge(int data[], int low, int mid, int high);
//基数排序
void radixSort(int data[], int n);
int getNumPos(int num, int pos);

int main() {
    int data[10] = {43, 65, 4, 23, 6, 98, 2, 65, 7, 79};
    int i;
    printf("原先数组:");
    for(i=0;i<10;i++) {
        printf("%d  ", data[i]);
```

```
    }
    printf("\n");
    /*printf("冒泡排序:");
  bubleSort(data, 10);
  for(i=0;i<10;i++) {
     printf("%d    ", data[i]);
  }
  printf("\n");
  printf("快速排序:");
  quickSort(data, 0, 9);
  for(i=0;i<10;i++) {
     printf("%d    ", data[i]);
  }
  printf("\n");
  printf("插入排序:");
  bInsertSort(data,10);
  for(i=0;i<10;i++) {
     printf("%d    ", data[i]);
  }
  printf("\n");
  printf("希尔排序:");
  shellSort(data, 10);
  for(i=0;i<10;i++) {
     printf("%d    ", data[i]);
  }
  printf("\n");
  printf("选择排序:");
  selectSort(data, 10);
  for(i=0;i<10;i++) {
     printf("%d    ", data[i]);
  }
  printf("\n");
  int data[11] = {-1, 43, 65, 4, 23, 6, 98, 2, 65, 7, 79};
  int i;
  printf("原先数组:");
  int data[11] = {-1, 43, 65, 4, 23, 6, 98, 2, 65, 7, 79};
  for(i=1;i<11;i++) {
     printf("%d    ", data[i]);
  }
  printf("\n");
  printf(" 堆排序:");
  heapSort(data, 10);
  for(i=1;i<11;i++) {
     printf("%d    ", data[i]);
```

```
    }
    printf("\n");
    printf("归并排序:");
    mergeSort(data, 0, 9);
    for(i=0;i<10;i++) {
        printf("%d   ", data[i]);
    }
    printf("\n");*/
      printf("基数排序:");
      radixSort(data, 10);
      for(i=0;i<10;i++) {
          printf("%d   ", data[i]);
      }
      printf("\n");
      return 0;
}

/*--------------------冒泡排序---------------------*/
void bubleSort(int data[], int n) {
    int i,j,temp;
    //两个for循环，每次取出一个元素跟数组的其他元素比较
    //将最大的元素排到最后。
    for(j=0;j<n-1;j++) {
        //外循环一次，就排好一个数，并放在后面，
        //所以比较前面n-j-1个元素即可
        for(i=0;i<n-j-1;i++) {
            if(data[i]>data[i+1]) {
                temp = data[i];
                data[i] = data[i+1];
                data[i+1] = temp;
            }
        }
    }
}

/*--------------------快速排序---------------------*/
int findPos(int data[], int low, int high) {
    //将大于t的元素赶到t的左边，大于t的元素赶到t的右边
    int t = data[low];
    while(low < high) {
        while(low < high && data[high] >= t) {
            high--;
        }
        data[low] = data[high];
        while(low < high && data[low] <=t) {
```

```
            low++;
        }
        data[high] = data[low];
    }
    data[low] = t;
    //返回此时t在数组中的位置
    return low;
}
//在数组中找一个元素，对大于该元素和小于该元素的两个数组进行再排序
//再对两个数组分为4个数组，再排序，直到最后每组只剩下一个元素为止
void quickSort(int data[], int low, int high) {
    if(low > high) {
        return;
    }
    int pos = findPos(data, low, high);
    quickSort(data, low, pos-1);
    quickSort(data, pos+1, high);
}

/*--------------------插入排序---------------------*/
void bInsertSort(int data[], int n) {
    int low,high,mid;
    int temp,i,j;
    for(i=1;i<n;i++) {
        low = 0;
        //把data[i]元素插入到它的前面data[0-(i-1)]中
        temp =data[i];
        high = i-1;
        //该while是折半，缩小data[i]的范围(优化手段)
        while(low <= high) {
            mid = (low+high)/2;
            if(data[mid] > temp) {
                high = mid-1;
            }
            else {
                low = mid+1;
            }
        }
        int j = i;
        //让data与已经排序好的数组的各个元素比较，小的放前面
        while((j > low) && data[j-1] > temp) {
            data[j] = data[j-1];
            --j;
        }
        data[low] = temp;
```

```
    }
}

/*--------------------希尔排序----------------------*/
void shellSort(int * data, int n) {
    int step,i,j,key;
    //将数组按照step分组，不断二分到每组只剩下一个元素
    for(step=n/2;step>0;step/=2) {
        //将每组中的元素排序，小的在前
        for(i=step;i<n;i++) {
            key = data[i];
            for(j=i-step;j>=0 && key<data[j];j-=step) {
                data[j+step] = data[j];
            }
            //和上面的for循环一起，将组中小的元素换到数组的前面
            data[j+step] = key;
        }
    }
}

/*--------------------选择排序----------------------*/
void selectSort(int data[], int n) {
    int i,j,mix,temp;
    //每次循环数组，找出最小的元素，放在前面，前面的即为排序好的
    for(i=0;i<n-1;i++) {
        //假设最小元素的下标
        int mix = i;
        //将上面假设的最小元素与数组比较，交换出最小的元素的下标
        for(j=i+1;j<n;j++) {
            if(data[j] < data[mix]) {
                mix = j;
            }
        }
        //若数组中真的有比假设的元素还小，就交换
        if(i != mix) {
            temp = data[i];
            data[i] = data[mix];
            data[mix] = temp;
        }
    }
}

/*--------------------堆排序----------------------*/
//堆排序将数组先组成二叉树，默认从数组的data[1]开始排，data[0]是
//无效数据
```

```
void heapSort(int data[], int n) {
    int i;
    //先将数组组成一棵完全二叉树
    //从2/n开始，就是从倒数第二排结点往前开始
    for(i=n/2;i>0;i--) {
        heapAdjust(data, i, n);
    }
    //循环每个结点，将大的结点交换到堆顶
    for(i=n;i>1;i--) {
        swap(data, 1, i);
        //每次交换完都要调整二叉树，将剩下的最大的结点交换到堆顶
        heapAdjust(data, 1, i-1);
    }
}
//交换函数
void swap(int data[], int i, int j) {
    int temp;
    temp = data[i];
    data[i] = data[j];
    data[j] = temp;
}
void heapAdjust(int data[], int i, int n) {
    int j, temp;
    //假设第一个结点的元素是最大的
    temp = data[i];
    //i结点：2*i是i结点的左结点，2*i+1是i结点的右结点
    //把结点元素大的交换到前面
    for(j=2*i;j<=n;j*=2) {
        if(j < n && data[j] < data[j+1]) {
            j++;
        }
        if(temp >= data[j]) {
            break;
        }
        data[i] = data[j];
        i = j;
    }
    data[i] = temp;
}

/*--------------------归并排序----------------------*/
void mergeSort(int data[], int first, int last) {
    int mid = 0;
    //将数组不停的二分分组再组合，直到每组只剩一个元素
    if(first < last) {
```

```
        mid = (first+last)/2;
        mergeSort(data, first, mid);
        mergeSort(data, mid+1, last);
        merge(data, first, mid, last);
    }
    return;
}
void merge(int data[], int low, int mid, int high) {
    int i, k;
    //定义一个临时数组存放传进来的无序数组排好序之后的数组
    int *temp = (int *)malloc((high-low+1)*sizeof(int));
    //将无序数组分成两个序列
    int left_low = low;
    int left_high = mid;
    int right_low = mid+1;
    int right_high = high;
    //将两个序列比较排序，小的排前
    for(k=0;left_low<=left_high && right_low<=right_high;k++) {
        if(data[left_low]<=data[right_low]) {
            temp[k] = data[left_low++];
        }
        else{
            temp[k] = data[right_low++];
        }
    }
    //左序列如果有剩下元素未排序，加到临时数组的末尾
    if(left_low <= left_high) {
        for(i=left_low;i<=left_high;i++) {
            temp[k++] = data[i];
        }
    }
    //右序列如果有剩下元素未排序，加到临时数组的末尾
    if(right_low <= right_high) {
        for(i=right_low;i<=right_high;i++) {
            temp[k++] = data[i];
        }
    }
    //将排好序的小分组转移到原数组中
    for(i=0;i<high-low+1;i++) {
        data[low+i] = temp[i];
    }
    free(temp);
    return;
}
/*--------------------基数排序---------------------*/
```

```
//该函数的作用是找出num的pos位数的数字(比如：23的个位数数字是3)
int getNumPos(int num, int pos) {
    int i;
    int temp = 1;
    for(i=0;i<pos-1;i++) {
        temp *= 10;
    }
    return (num / temp) % 10;
}
void radixSort(int data[], int n) {
    int i,j,k,pos,num,index;
    //这几句话是创建一个从0-9(行)× (n+1)(列)的网格，第一列从上往下是0-9,
    //第二列是该行包含的元素个数，默认为0个
    int *radixArrays[10];
    for(i=0;i<10;i++) {
        radixArrays[i] = (int *)malloc(sizeof(int) * (n+1));
        radixArrays[i][0] = 0;
    }
    //pos最大为31为数，计算机能承受的最大范围了
    for(pos=1;pos<=31;pos++) {
        //该for循环是将数组的元素按照位数(pos)的值放进网格内
        for(i=0;i<n;i++) {
            num = getNumPos(data[i], pos);
            index = ++radixArrays[num][0];
            radixArrays[num][index] = data[i];
        }
        //该for循环是将上面的for循环已经按照某个位数(pos)排列好的元素存入数组
        for(i=0,j=0;i<10;i++) {
            for(k=1;k<=radixArrays[i][0];k++) {
                data[j++] = radixArrays[i][k];
            }
            //清空网格，以便给下个位数排列
            radixArrays[i][0] = 0;
        }
    }
}
```

以上排序算法的优劣（时间复杂度和空间复杂度对比）：

数组排序算法

算法	时间复杂度			空间复杂度
	最佳	平均	最差	最差
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

测试：

```
原先数组:43    65    4    23    6    98    2    65    7    79
 堆排序:2    4    6    7    23    43    65    65    79    98
```



```
原先数组:43    65    4    23    6    98    2    65    7    79
归并排序:2    4    6    7    23    43    65    65    79    98
```

```
原先数组:43    65    4    23    6    98    2    65    7    79
基数排序:2    4    6    7    23    43    65    65    79    98
```



```
原先数组:43    65    4    23    6    98    2    65    7    79
希尔排序:2    4    6    7    23    43    65    65    79    98
```



```
原先数组: 43    65    4    23    6    98    2    65    7    79
冒泡排序:2    4    6    7    23    43    65    65    79    98
```



```
原先数组: 43    65    4    23    6    98    2    65    7    79
快速排序:2    4    6    7    23    43    65    65    79    98
插入排序:希尔排序:[wayne@wayne demo]$ vim sort.c
[wayne@wayne demo]$ g++ sort.c -o sort
[wayne@wayne demo]$ ./sort
原先数组43    65    4    23    6    98    2    65    7    79
插入排序:2    65    4    23    6    6    2    6    6    6
```



```
原先数组:43    65    4    23    6    98    2    65    7    79
选择排序:2    4    6    7    23    43    65    65    79    98
```


c语言排序算法（一） 阅读数 8189

排序算法，是算法之中相对基础的，也是各门语言的必学的算法。本篇文章用C语言为大家介绍排序... 博文 来自： liu659_的博客

十大经典排序算法（C语言实现） 阅读数 1万+

原文链接：https://www.cnblogs.com/onepixel/articles/7674659.html1、冒泡排序（BubbleSort）... 博文 来自： 小码的嵌入式...