项目说明文档

数据结构课程设计

——8种排序算法比较

作 者 姓 名： 曹晓慈

学 号： 2052844

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

# 分析

## 功能分析

排序也称排序算法（sort Alogrithm），排序时将以组数据，依指定顺序进行排序的过程。共分为插入排序（直接插入，希尔排序）、选择排序（简单选择排序、堆排序）、交换排序（冒泡排序、快速排序）、归并排序和基数排序。

# 设计

## 成员与操作设计

用vector结构类型来储存将要用来被排序的数组。还有以下函数用于在对数组排序时进行调用：

void swap(int &a, int &b)； //交换函数

void BubbleSort(vector<int> storage,int count)； //冒泡排序

void SelectionSort(vector<int> storage, int count) //选择排序

void StraightInsertion(vector<int> storage, int count) //直接插入排序

void ShellSort(vector<int> storage, int count) //希尔排序

void QuickSort(vector<int> &storage,int left,int right,int &sum,int count) //快速排序

void adjust(vector<int>& arr, int len, int index)；

void HeapSort(vector<int> arr, int size) //堆排序

void Merge(vector<int> &arr, int start, int mid, int end,int &sum)

void MergeSort(vector<int> &arr,int start,int end,int &sum) //归并排序

int MaxBit(vector<int> arr, int count)

void RadixSort(vector<int> arr,int n) //基数排序

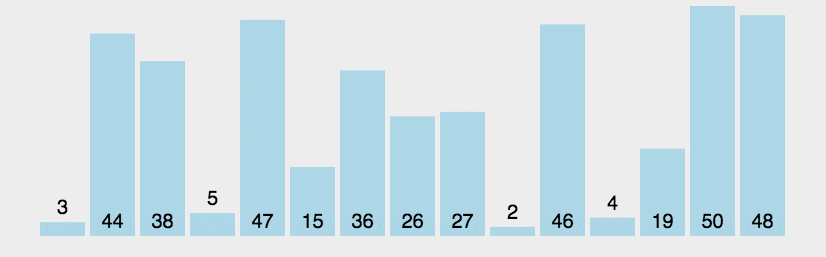
# 实现

## 冒泡排序实现

比较相邻的元素。如果第一个比第二个大，就交换他们两个。对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。

针对所有的元素重复以上的步骤，除了最后一个。

持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。时间复杂度：O ( n\*2)



核心代码：

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

for (int j = 0; j < count - 1 - i; j++)

{

if (arr[j] > arr[j + 1])

{

swap(arr[j], arr[j + 1]);

sum++;

}

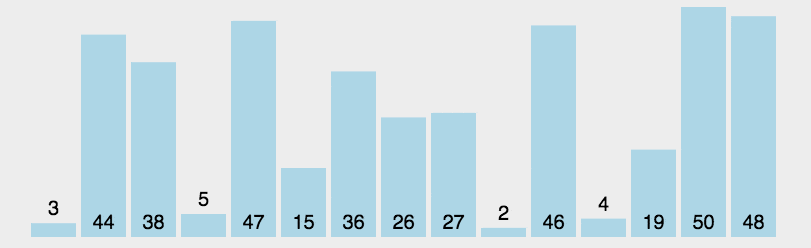
}

## 选择排序实现

首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置。再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。

重复第二步，直到所有元素均排序完毕。

时间复杂度：O ( n\*2)



核心代码：

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

{

min = arr[i];

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

{

if (min > arr[j])

{

swap(min, arr[j]);

sum++;

}

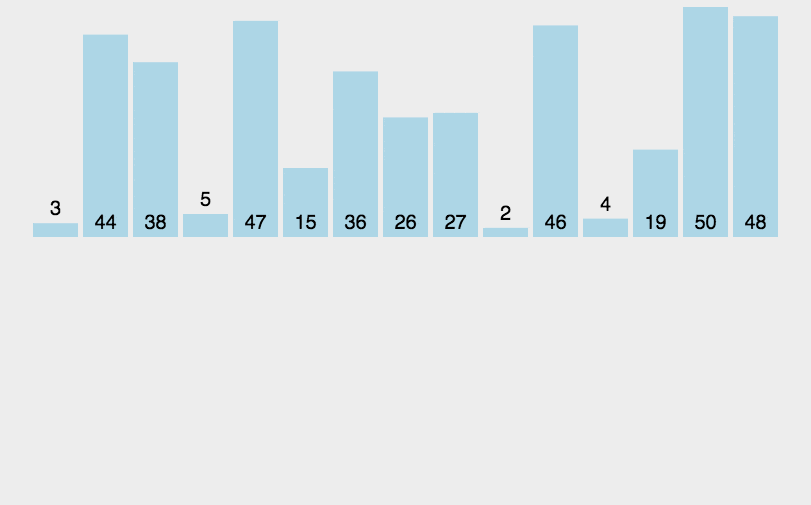
}

arr[i] = min;

}

## 直接插入排序实现

将第一待排序序列第一个元素看做一个有序序列，把第二个元素到最后一个元素当成是未排序序列。从头到尾依次扫描未排序序列，将扫描到的每个元素插入有序序列的适当位置。（如果待插入的元素与有序序列中的某个元素相等，则将待插入元素插入到相等元素的后面。）



核心代码：

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

{

int tmp = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > tmp)

{

arr[j + 1] = arr[j];

sum++;

j--;

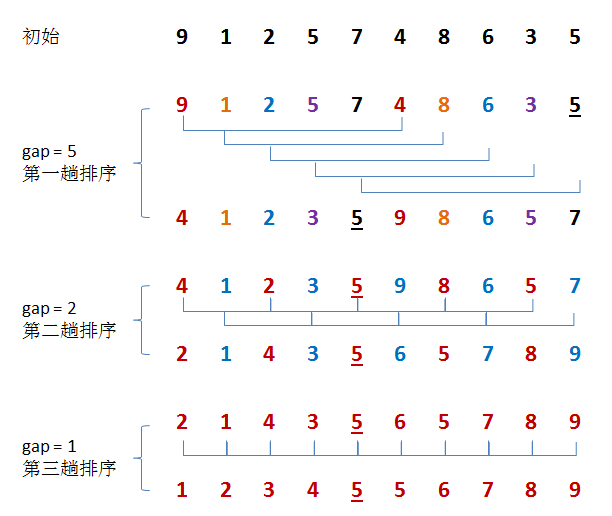
}

arr[j + 1] = tmp;

}

## 希尔排序实现

先将整个待排序的记录序列分割成为若干子序列分别进行直接插入排序，待整个序列中的记录"基本有序"时，再对全体记录进行依次直接插入排序。



核心代码：

gap = count / 2;//设置增量

while (gap > 0)//出口

{

for (i = gap; i < count; i++)//对所有组采用直接插入排序

{

tmp = arr[i];//对相隔d个为止一组采用直接插入排序

j = i - gap;

while (j >= 0 && arr[j] > tmp)

{

arr[j + gap] = arr[j];

sum++;

j = j - gap;

}

arr[j + gap] = tmp;

}

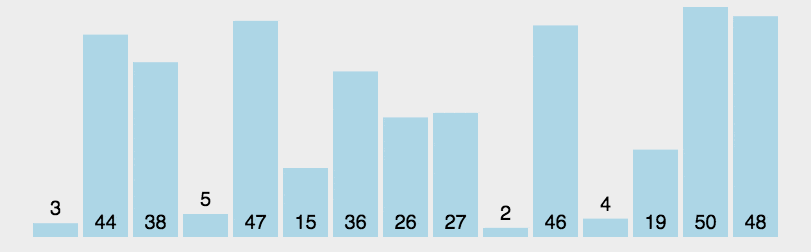
gap = gap / 2;//减小增量

}

## 快速排序实现

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）。

实现为从数列中挑出一个元素，称为 "基准"（pivot）;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；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。



核心代码：

if (left < right)

{

int i = left, j = right, x = storage[left];

while (i < j)

{

while (i < j && storage[j] >= x) // 从右向左找第一个小于x的数

j--;

if (i < j)

{

storage[i++] = storage[j];

sum++;

}

while (i < j && storage[i] < x) // 从左向右找第一个大于等于x的数

i++;

if (i < j)

{

storage[j--] = storage[i];

sum++;

}

}

storage[i] = x;

sum++;

QuickSort(storage, left, i - 1,sum,count); // 递归调用

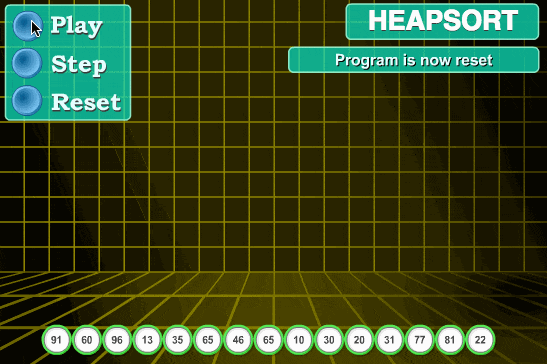
QuickSort(storage, i + 1, right,sum,count);

}

## 堆排序实现

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。

实现为创建一个堆 H[0……n-1]，把堆首（最大值）和堆尾互换，把堆的尺寸缩小 ，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，重复步骤 2，直到堆的尺寸为 1。



核心代码：

void adjust(vector<int>& arr, int len, int index)

{

int left = 2 \* index + 1; // index的左子节点

int right = 2 \* index + 2;// index的右子节点

int maxIdx = index;

if (left<len && arr[left] > arr[maxIdx]) maxIdx = left;

if (right<len && arr[right] > arr[maxIdx]) maxIdx = right;

if (maxIdx != index)

{

swap(arr[maxIdx], arr[index]);

adjust(arr, len, maxIdx);

}

}

// 堆排序

void HeapSort(vector<int> arr, int size)

{

// 构建大根堆（从最后一个非叶子节点向上）

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

{

adjust(arr, size, i);

}

// 调整大根堆

for (int i = size - 1; i >= 1; i--)

{

swap(arr[0], arr[i]); // 将当前最大的放置到数组末尾

adjust(arr, i, 0); // 将未完成排序的部分继续进行堆排序

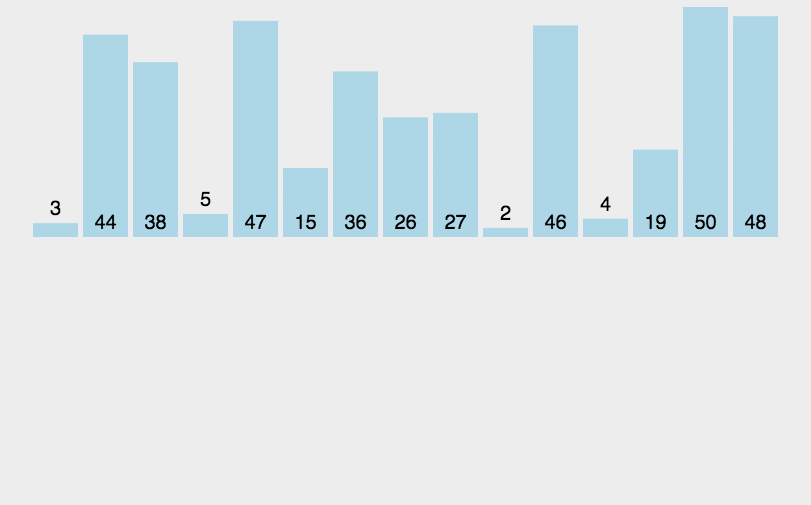
}

}

## 归并排序实现

归并排序（Merge sort）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。

实现为申请空间，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；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；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；重复步骤 3 直到某一指针达到序列尾；将另一序列剩下的所有元素直接复制到合并序列尾。



核心代码：

void Merge(vector<int> &arr, int start, int mid, int end,int &sum)

{

int length = end - start + 1;

int\* temp\_array = new int[length];

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

{

temp\_array[index] = arr[start + index];

}

int i = 0;

int j = mid - start + 1;

for (int k = start; k <= end; k++)

{

//临界位置

if (i == mid - start + 1)

{

arr[k] = temp\_array[j++];

sum++;

continue;

}

if (j == end - start + 1)

{

arr[k] = temp\_array[i++];

sum++;

continue;

}

//非临界位置

if (temp\_array[i] > temp\_array[j])

{

arr[k] = temp\_array[j++];

sum++;

}

else

{

arr[k] = temp\_array[i++];

sum++;

}

}

delete[] temp\_array;

}

void MergeSort(vector<int> &arr,int start,int end,int &sum) //归并排序

{

if (start >= end)

return;

int mid = (start + end) / 2;

MergeSort(arr, start, mid,sum);

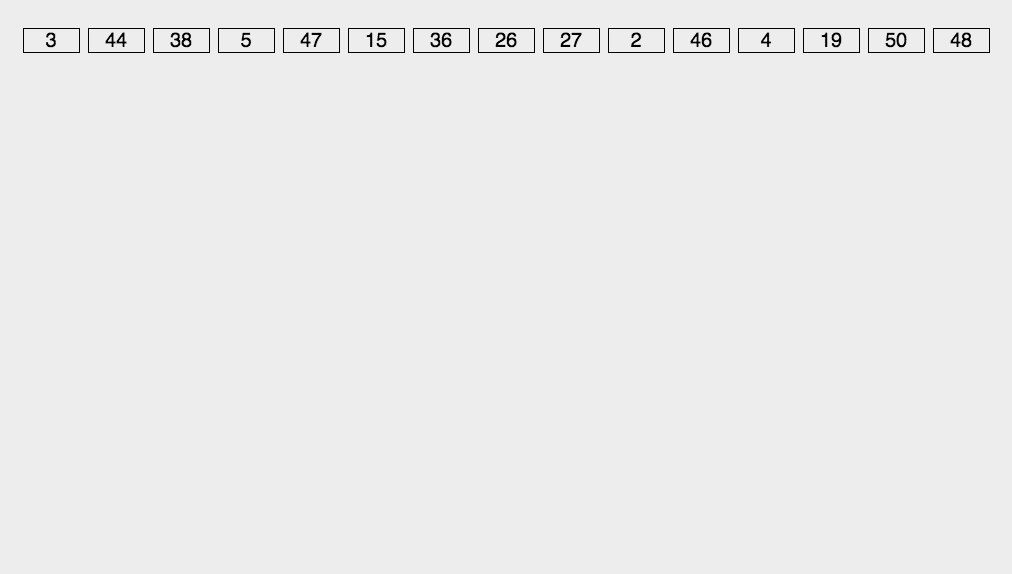
MergeSort(arr, mid + 1, end,sum);

Merge(arr, start, mid, end,sum);

}

## 基数排序实现

基数排序是一种非比较型整数排序算法，其原理是将整数按位数切割成不同的数字，然后按每个位数分别比较。



核心代码：

int MaxBit(vector<int> arr, int count)

{

int Bit = 1, tmp = 10;

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

{

while (arr[i] >= tmp)

{

tmp = tmp \* 10;

Bit++;

}

}

return Bit;

}

void RadixSort(vector<int> arr,int n) //基数排序

{

int maxbit = MaxBit(arr, n);

vector<int> tmp(n, 0);

vector<int> count(10, 0); //计数器

int i, j, k;

int radix = 1;

for (i = 1; i <= maxbit; i++) //进行d次排序

{

for (j = 0; j < 10; j++)

count[j] = 0; //每次分配前清空计数器

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

{

k = (arr[j] / radix) % 10; //统计每个桶中的记录数

count[k]++;

}

for (j = 1; j < 10; j++)

count[j] = count[j - 1] + count[j]; //将tmp中的位置依次分配给每个桶

for (j = n - 1; j >= 0; j--) //将所有桶中记录依次收集到tmp中

{

k = (arr[j] / radix) % 10;

tmp[count[k] - 1] = arr[j];

count[k]--;

}

for (j = 0; j < n; j++) //将临时数组的内容复制到data中

arr[j] = tmp[j];

radix = radix \* 10;

}

}

# 测试

程序测试运行结果

