中 国 矿 业 大 学

2018 级《数据结构与算法分析》课程作业

学生姓名 王茂凯

学　　号 04181425

中国矿业大学信控学院

1. 简答：排序算法的稳定性？本章介绍的哪些方法是稳定的排序算法，哪些是不稳定的排序算法？

稳定性:当待排序序列中有两个或两个以上相同的关键字时,排序前后这些关键字的相对顺序若发生了变化,则排序算法是不稳定的,否则就是稳定排序算法

稳定排序算法:冒泡排序,插入排序,归并排序,桶排序,基数排序
不稳定排序算法:选择排序,快速排序,堆排序

2. 设待排序的对象为{12，2，16，30，28，10，16*，20，6，18}，是分别写出用一下排序方法每趟排序后的结果。
（1）直接插入排序　（2）快速排序　（3）希尔排序　（4）冒泡排序　（5）基数排序　（6）直接选择排序　（7）归并排序

关于稳定不稳定上题已经说明,因为算法的不同,结果可能有差异

```
input num:
10
input val:
12 2 16 30 28 10 16 20 6 18
insertsort:
2 12 16 30 28 10 16 20 6 18
2 12 16 30 28 10 16 20 6 18
2 12 16 30 28 10 16 20 6 18
2 12 16 28 30 10 16 20 6 18
2 10 12 16 28 30 16 20 6 18
2 10 12 16 16 28 30 20 6 18
2 10 12 16 16 20 28 30 6 18
2 6 10 12 16 16 20 28 30 18
2 6 10 12 16 16 18 20 28 30
```

```
quicksort:
10 2 6 12 28 30 16 20 16 18
6 2 10 12 28 30 16 20 16 18
2 6 10 12 28 30 16 20 16 18
2 6 10 12 16 18 16 20 28 30
2 6 10 12 16 16 18 20 28 30
2 6 10 12 16 16 18 20 28 30
```

```
shellsort:
10 2 16 6 18 12 16 20 30 28
10 2 16 6 16 12 18 20 30 28
2 6 10 12 16 16 18 20 28 30
```

```
bubblesort:
2 12 16 28 10 16 20 6 18 30
2 12 16 10 16 20 6 18 28 30
2 12 10 16 16 6 18 20 28 30
2 10 12 16 6 16 18 20 28 30
2 10 12 6 16 16 18 20 28 30
2 10 6 12 16 16 18 20 28 30
2 6 10 12 16 16 18 20 28 30
2 6 10 12 16 16 18 20 28 30
```

```
radixsort:
30 10 20 12 2 16 16 6 28 18
2 6 10 12 16 16 18 20 28 30
```

```
selectsort:
2 12 16 30 28 10 16 20 6 18
2 6 16 30 28 10 16 20 12 18
2 6 10 30 28 16 16 20 12 18
2 6 10 12 28 16 16 20 30 18
2 6 10 12 16 28 16 20 30 18
2 6 10 12 16 16 28 20 30 18
2 6 10 12 16 16 18 20 30 28
2 6 10 12 16 16 18 20 30 28
2 6 10 12 16 16 18 20 28 30
```

```
mergesort:
2 12 16 30 28 10 16 20 6 18
2 12 16 30 28 10 16 20 6 18
2 12 16 28 30 10 16 20 6 18
2 12 16 28 30 10 16 20 6 18
2 12 16 28 30 10 16 20 6 18
2 12 16 28 30 10 16 20 6 18
2 12 16 28 30 10 16 20 6 18
2 12 16 28 30 6 10 16 18 20
2 6 10 12 16 16 18 20 28 30
```

```cpp
class mysort
{
public:
    mysort(int n = 5) : _n(n),
                        arry(new int[_n]())
    {
        cout << "input val:" << endl;
        for (int i = 0; i < _n; ++i)
            cin >> arry[i];
    }
    ~mysort()
    {
        if (arry != nullptr)
            delete[] arry;
        arry = nullptr;
    }
    void bubblesort(); //冒泡排序
```

```
    void insertsort(); //插入排序
    void selectsort(); //选择排序
    void shellsort();  //希尔排序
    void radixsort();  //基数排序
    void print()
    {
        for (int i = 0; i < _n; ++i)
            cout << arry[i] << " ";
        cout << endl;
    }

private:
    int _n;
    int *arry;
    int maxbit();                                   //基数排序求位数函数
    int bitnumber(const int &x, const int &bit); //基数排序求某位数函数

    friend void quicksort(mysort &test, int low, int high);      //快速
排序
    friend int partition(mysort &test, int low, int high);       //快速
排序分离函数
    friend void mergesort(mysort &test, int low, int high);      //归并
排序
    friend void merge(mysort &test, int low, int mid, int high); //归并
排序合并函数
};
//冒泡排序
void mysort::bubblesort()
{
    bool flag = true; //结束标志
    int i = 0;
    int j = 0;
    int temp = 0;
    while (flag)
    {
        flag = false;
        for (j = 0; j < _n - i - 1; ++j)
        {
            if (arry[j] > arry[j + 1]) //交换
            {
                temp = arry[j];
                arry[j] = arry[j + 1];
                arry[j + 1] = temp;
                flag = true; //发生交换则置为 true
```

```
            }
        }
        ++i;
        print();
    }
}
//插入排序
void mysort::insertsort() //在已经有序的序列中找的合适的位置插入
{
    int i = 0;
    int j = 0;
    int temp = 0;
    for (i = 0; i < _n - 1; ++i)
    {
        if (arry[i] > arry[i + 1])
        {
            temp = arry[i + 1];    //临时保存第 i+1 个数
            arry[i + 1] = arry[i]; //将第 i 个数后移
            //在已经排好的序列中找位置
            for (j = i - 1; arry[j] > temp && j >= 0; --j) //从后往前找
                arry[j + 1] = arry[j];
            arry[j + 1] = temp;
        }
        print();
    }
}
//选择排序
void mysort::selectsort() //在待排序的序列中选择一个最小的元素与最前面的元素交换
{
    for (int i = 0; i < _n - 1; ++i)
    {
        int k = i;
        for (int j = i + 1; j < _n; ++j) //寻找待排序列中最小元素
            if (arry[j] < arry[k])
                k = j;
        if (k != i) //将最小元素与第 i 个交换
        {
            int temp = arry[i];
            arry[i] = arry[k];
            arry[k] = temp;
        }
        //print();
    }
```

```
}
//希尔排序
void mysort::shellsort()
{
    int gap = _n / 2; //以 gap 间隔两两对应比较交换
    int temp = 0;
    while (gap > 0)
    {
        for (int i = gap; i < _n; ++i)
        {
            temp = arry[i];          //从 gap 开始
            int preindex = i - gap; //与第 i 个数对应的数
            while (preindex >= 0 && arry[preindex] > temp)
            {
                arry[preindex + gap] = arry[preindex];
                preindex -= gap; //以 gap 为周期向前移动
            }
            arry[preindex + gap] = temp;
        }
        gap /= 2; //缩小间隔
        //print();
    }
}
//基数排序
int mysort::maxbit() //求序列中最大数的位数
{
    int maxvalue = arry[0];      //最大数
    int digits = 0;              //最大位数
    for (int i = 1; i < _n; ++i) //找最大数
        if (arry[i] > maxvalue)
            maxvalue = arry[i];
    while (maxvalue != 0)
    {
        ++digits;
        maxvalue /= 10;
    }
    return digits;
}
int mysort::bitnumber(const int &x, const int &bit) //求 x 第 bit 位上的数字
{
    int temp = 1;
    for (int i = 1; i < bit; ++i)
        temp *= 10;
    return (x / temp) % 10;
```

```
}
void mysort::radixsort() //应用到桶排序**画图理解**难
{
    int i, j, k, bit, max;
    max = maxbit();
    int **temparry = new int *[10]();
    for (i = 0; i < 10; ++i)
        temparry[i] = new int[_n + 1]();

    for (bit = 1; bit <= max; ++bit)
    {
        for (j = 0; j < _n; ++j)
        {
            int num = bitnumber(arry[j], bit);
            int index = ++temparry[num][0];
            temparry[num][index] = arry[j];
        }
        for (i = 0, j = 0; i < 10; ++i)
        {
            for (k = 1; k <= temparry[i][0]; ++k)
                arry[j++] = temparry[i][k];
            temparry[i][0] = 0;
        }
        //print();
    }

    for (i = 0; i < 10; ++i)
        delete[] temparry[i];
    delete[] temparry;
}
//快速排序
void quicksort(mysort &test, int low, int high)
{
    int mid = 0;
    if (low < high)
    {
        mid = partition(test, low, high);
        quicksort(test, low, mid - 1);  //递归基准位置前的序列
        quicksort(test, mid + 1, high); //递归基准位置后的序列
        //test.print();
    }
}
int partition(mysort &test, int low, int high)
{
```

```
    int i = low;
    int j = high;
    int pivot = test.arry[i]; //以第一个数为基准
    while (i < j)
    {
        while (i < j && test.arry[j] > pivot) //从右向左移动指针直到遇到比基准小的数
            j--;
        while (i < j && test.arry[i] <= pivot) //从左向右移动指针直到遇到比基准大的数
            i++;
        if (i < j)
            swap(test.arry[i++], test.arry[j--]); //交换这两个数并移动指针
    }
    //移动准基位置,此后基准数确定了在序列中的位置,返回基准数的位置
    if (test.arry[i] > pivot)
    {
        swap(test.arry[i - 1], test.arry[low]);
        return i - 1;
    }
    swap(test.arry[i], test.arry[low]);
    return i;
}
//归并排序
void mergesort(mysort &test, int low, int high)
{
    if (low < high)
    {
        int mid = (low + high) / 2;     //以中心分成两个序列
        mergesort(test, low, mid);      //递归前序列
        mergesort(test, mid + 1, high); //递归后序列
        merge(test, low, mid, high);    //合并
        //test.print();
    }
}
void merge(mysort &test, int low, int mid, int high)
{
    int *temparry = new int[high - low + 1]();
    int i = low, j = mid + 1, k = 0;
    while (i <= mid && j <= high) //序列的两部分合并
    {
        if (test.arry[i] <= test.arry[j])
            temparry[k++] = test.arry[i++];
```

```
        else
            temparry[k++] = test.arry[j++];
    }
    //多余序列移动
    while (i <= mid)
        temparry[k++] = test.arry[i++];
    while (j <= high)
        temparry[k++] = test.arry[j++];
    //将序列移到原空间
    for (i = low, k = 0; i <= high; ++i)
        test.arry[i] = temparry[k++];
    delete[] temparry;
}
```