排序实验报告 1950509 马家昱

目录

实验内容

1. 随机生成不同规模的数据（10，100，1K，10K，100K，1M，10K正序，10K逆序），并保存到文件中，以input1.txt, input2.txt,...,input8.txt命名。
2. 用上面产生的数据，对不同的排序方法（插入排序、选择排序、冒泡排序、希尔排序、堆排序、快速排序、归并排序）进行排序测试，给出实验时间。
3. 分析各种排序算法的特点，给出时间复杂度，以及是否是稳定排序。

实验要求

1. 程序要添加适当的注释，程序的书写要采用 缩进格式 。
2. 程序要具在一定的 健壮性，即当输入数据非法时， 程序也能适当地做出反应，如 插入删除时指定的位置不对 等等。
3. 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。
4. 根据实验报告模板详细书写实验报告，在实验报告中给出主要算法的复杂度分析。
5. 实验结果以表格形式列出，给出测试环境（硬件环境和软件环境），并对结果进行分析，说明各种排序算法的优缺点。
6. 将实验报告和输入文件打包成.zip文件，上传。

生成测试数据

本人使用python分别生成了数量分别为 **10，100，1K，10K，100K，1M, 逆10K** 且大小均为1~2147483648的随机正整数。实现代码如下：

```
    import random
    import os

    dict=
{'10':"10",'100':"100",'1000':"1K",'10000':"10K",'100000':"100K",'1000000':"1M"}

    def data_create(size):

        filepath="./"+dict[str(size)]+".txt"

        file = open(filepath, 'w')

        for i in range(size):
            file.write(str(random.randint(0, 2147483646) + 1) + " ")

        file.close()

        for size in dict:
        data_create(int(size))
```

不同排序方式的C++实现

插入排序

```
    void InsertSort() {//插入排序
        int len;
        cin >> len;
        int* temp = (int*)malloc((len + 1) * sizeof(int));
        for (int i = 1; i <= len; i++)
            cin >> temp[i];


        for(int i=2;i<=len;i++)
            if (temp[i] < temp[i - 1]) {
                temp[0] = temp[i];
                temp[i] = temp[i - 1];
                int j;
                for (j = i - 2; temp[0] < temp[j]; j--)
                    temp[j + 1] = temp[j];
                temp[j + 1] = temp[0];
            }


        cout << "插入排序:" << endl;
```

```
    for (int i = 1; i <= len; i++)
        cout << temp[i] << ' ';
    cout << endl;
}
```

选择排序

```
int Min(int* temp, int i, int len) {
    int min = 99999;
    int result;
    for (int j = i; j <= len; j++)
        if (temp[j] < min) {
            result = j;
            min = temp[j];
        }
    return result;
}

void SelectSort() {//选择排序
    int len;
    cin >> len;
    int* temp = (int*)malloc((len + 1) * sizeof(int));
    for (int i = 1; i <= len; i++)
        cin >> temp[i];
    for (int i = 1; i < len; i++) {
        int j = Min(temp, i, len);
        if (i != j) {
            int t = temp[i];
            temp[i] = temp[j];
            temp[j] = t;
        }
    }
    cout << "选择排序:" << endl;
    for (int i = 1; i <= len; i++)
        cout << temp[i] << ' ';
    cout << endl;
}
```

冒泡排序

```
void BubbleSort() {//冒泡排序
    int len;
    cin >> len;
    int* temp = (int*)malloc((len) * sizeof(int));
    for (int i = 0; i < len; i++)
        cin >> temp[i];


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

```
            for (int j = 0; j < len - i - 1; j++) {
                if (temp[j] > temp[j + 1]) {
                    int t = temp[j];
                    temp[j] = temp[j + 1];
                    temp[j + 1] = t;
                }
            }
        }


        cout << "冒泡排序:" << endl;
        for (int i = 0; i < len; i++)
            cout << temp[i] << ' ';
        cout << endl;
    }
```

希尔排序

```
    void ShellInsert(int* arr,int len,int gap){//一趟希尔排序
        for (int i = gap + 1; i <= len; ++i){
            if (arr[i]<arr[i-gap]){
                arr[0] = arr[i];
                int j;
                for (j = i - gap; j > 0 && arr[0]<arr[j]; j -= gap)
                    arr[j + gap] = arr[j];
                arr[j + gap] = arr[0];
            }
        }
    }

    void ShellSort() {
        int len;
        cin >> len;
        int* temp = (int*)malloc((len + 1) * sizeof(int));
        for (int i = 1; i <= len; i++)
            cin >> temp[i];
        int dlta[3] = { 5,3,1 };
        for(int i=0;i<3;i++)
            ShellInsert(temp, len, dlta[i]);
        cout << "希尔排序:" << endl;
        for (int i = 1; i <= len; i++)
            cout << temp[i] << ' ';
        cout << endl;
    }
```

堆排序

```
    void HeapAdjust(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]);
            HeapAdjust(arr, len, maxIdx);
        }
    }

    void heap(int* arr, int len) {
        for (int i = len / 2 - 1; i >= 0; i--)
            HeapAdjust(arr, len, i);

        for (int i = len - 1; i >= 1; i--){
            swap(arr[0], arr[i]);
            HeapAdjust(arr, i, 0);
        }
    }

    void HeapSort() {
        int len;
        cin >> len;
        int* temp = (int*)malloc((len) * sizeof(int));
        for (int i = 0; i < len; i++)
            cin >> temp[i];
        heap(temp, len);
    //  cout << "堆排序:" << endl;
        for (int i = 0; i < len; i++)
            cout << temp[i] << ' ';
        cout << endl;
    }
```

快速排序

```
    void quick(int left, int right, int* arr) {
        if (left >= right)
            return;
        int i, j, base, temp;
        i = left, j = right;
        base = arr[left];  //取最左边的数为基准数
        while (i < j)
        {
            while (arr[j] >= base && i < j)
                j--;
            while (arr[i] <= base && i < j)
                i++;
```

```
            if (i < j)
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        //基准数归位
        arr[left] = arr[i];
        arr[i] = base;
        quick(left, i - 1, arr);//递归左边
        quick(i + 1, right, arr);//递归右边
    }

    void QuickSort() {//快速排序
        int len;
        cin >> len;
        int* temp = (int*)malloc((len) * sizeof(int));
        for (int i = 0; i < len; i++)
            cin >> temp[i];
        quick(0, len - 1, temp);
        cout << "快速排序:" << endl;
        for (int i = 0; i < len; i++)
            cout << temp[i] << ' ';
        cout << endl;
    }
```

归并排序

```
    void merge(int* data, int start, int end, int* result)
    {
        int left_length = (end - start + 1) / 2 + 1;
        int left_index = start;
        int right_index = start + left_length;
        int result_index = start;
        while (left_index < start + left_length && right_index < end + 1)  //store
data into new array
        {
            if (data[left_index] <= data[right_index])
                result[result_index++] = data[left_index++];
            else
                result[result_index++] = data[right_index++];
        }
        while (left_index < start + left_length)
            result[result_index++] = data[left_index++];
        while (right_index < end + 1)
            result[result_index++] = data[right_index++];
    }

    void Msort(int* data, int start, int end, int* result)
    {
```

```
        if (1 == end - start)
        {
            if (data[start] > data[end])
            {
                int temp = data[start];
                data[start] = data[end];
                data[end] = temp;
            }
            return;
        }
        else if (end == start)
            return;
        else {
            Msort(data, start, (end - start + 1) / 2 + start, result);
            Msort(data, (end - start + 1) / 2 + start + 1, end, result);
            merge(data, start, end, result);
            for (int i = start; i <= end; ++i)
            {
                data[i] = result[i];
            }
        }
    }


    void MergeSort() {//归并排序
        int len;
        cin >> len;
        int* temp = (int*)malloc((len) * sizeof(int));
        int* result = (int*)malloc((len) * sizeof(int));
        for (int i = 0; i < len; i++)
            cin >> temp[i];
        Msort(temp,0,len-1,result);
        cout << "归并排序:" << endl;
        for (int i = 0; i < len; i++)
            cout << result[i]<<' ';
        cout << endl;
    }
```

实验结果分析

示例代码

使用C++打开txt文件并读取数据，利用clock()函数测得不同排序方式在不同测试数据下所耗费的时间。具体实现如下

```
    //插入排序100K数据量下所用时间
    clock_t start, finish;

    void InsertSort() {//插入排序
        int len=100000;
```

```
    int* temp = (int*)malloc((len + 1) * sizeof(int));
    ifstream in("E:\\数据结构\\Sort\\100K.txt");
    for (int i = 1; i <= len; i++)
        in >> temp[i];


    start = clock();
    for (int i = 2; i <= len; i++)
        if (temp[i] < temp[i - 1]) {
            temp[0] = temp[i];
            temp[i] = temp[i - 1];
            int j;
            for (j = i - 2; temp[0] < temp[j]; j--)
                temp[j + 1] = temp[j];
            temp[j + 1] = temp[0];
        }
    finish = clock();

    cout << "插入排序:" << endl;
    for (int i = 1; i <= len; i++)
        cout << temp[i] << ' ';
    cout << endl;
}

int main() {
    InsertSort();
    cout << double(finish - start) / CLOCKS_PER_SEC << "s";
}
```

配置环境

- CPU: Inter(R)Core(TM)i7-9750H CPU@2.60GHz
- Microsoft Visual Studio 2019

具体数据

单位：s

排序方法	10	100	1K	10K	100k	1M	逆10K
插入排序	0	0	0.001	0.047	4.705	466.315	0.14
选择排序	0	0	0	0.001	0.016	0.183	0.184
冒泡排序	0	0	0.001	0.198	23.003	2316.89	0.235
希尔排序	0	0	0	0.012	1.193	112.262	0.045
堆排序	0	0	0.001	0.01	0.14	1.664	0.016
快速排序	0	0	0	0.001	0.012	0.138	0.155
归并排序	0	0	5	0.001	0.015	0.183	0.001

- 由于`clock()`精度有限，故时间较短时测得数据为0
- 希尔排序的增量设置为`dlta={5,3,1}`

结果分析

- 就平均时间性能而言，快速排序最佳，其所需时间最省，但在最坏情况下不如归并排序与堆排序
- 在n较大时，归并排序所需时间较堆排序省，但所需的辅助存储量多
- 当n较小时，插入排序、选择排序与冒泡排序是最佳的排序方法

排序方法	平均时间	最坏情况	辅助存储	是否稳定
插入排序	$O(n^2)$	$O(n^2)$	O(1)	是
选择排序	$O(n^2)$	$O(n^2)$	O(1)	是
冒泡排序	$O(n^2)$	$O(n^2)$	O(1)	是
堆排序	O(nlogn)	O(nlogn)	O(1)	否
快速排序	O(nlogn)	$O(n^2)$	O(logn)	否
归并排序	O(nlogn)	O(nlogn)	O(n)	否

总结

- 通过本次实验，我使用C++将在课堂上了解到的七种基本排序方法实现，并分别对不同的数据量测试其时间性能，验证其时间复杂度。
- 深入了解了每种算法的思想与原理，对其步骤有了清楚的认识，并且学习了选择在不同应用场景下使用哪种排序算法是最优的思路。