山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学　　号　　：202200400053	姓名：　王宇涵	班级：　22 级 2 班
实验题目：排序算法		
实验学时：2	实验日期：　　2023-9-20	

实验目的：

掌握各种简单排序算法。

软件开发环境：

Vscode

1. 实验内容

1、题目描述：

用任意一种排序方式给出 n 个整数按升序排序后的结果，满足以下要求：

1.不得使用与实验相关的 STL；

2.需使用类模版(template<class T>)；

3.需定义排序类，封装各排序方法；

4.排序数据需使用动态数组存储；

5.排序类需提供以下操作：名次排序、及时终止的选择排序、及时终止的冒泡排序、插入排序。

输入输出格式：

输入：输入的第一行是一个整数 n(1<=n<=1000)，表示需排序的数的个数。接下来一行是 n 个整数，数的范围是 0 到 1000，每两个相邻数据间用一个空格分隔。

输出：一行排好序的序列。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

1. 名次排序（Rank Sort）：

- 名次排序是一种基于元素的相对大小关系的排序方法。它不直接对元素进行比较和交换，而是通过计算每个元素在原数组中的名次来确定其在排序后的位置。

- 首先，创建一个与原数组相同大小的辅助数组，用于存储每个元素的名次。

- 然后，对原数组中的每个元素，遍历整个数组并比较它与其他元素的大小关系，统计小于等于它的元素个数，作为该元素的名次。

- 最后，根据名次将元素放置到新数组中的相应位置，这就完成了排序。

- 名次排序适用于一些特殊的情况，例如需要同时排序多个数组。

2. 及时终止的选择排序（Timely Terminating Selection Sort）：

- 这是改进版的选择排序，旨在在已经有序或接近有序的情况下提前终止排序，减少不必要的比较和交换。

- 与传统选择排序相似，它在每一轮中选择未排序部分的最小元素，并与未排序部分的第一个元素交换。

- 在每一轮中，它还检查是否发生了交换。如果没有交换发生，说明数组已经有序，可以提前终止排序。

- 这种算法在某些情况下可以显著减少比较次数，但在最坏情况下仍然是 O(n^2)。

3. 及时终止的冒泡排序（Timely Terminating Bubble Sort）：

- 这是改进版的冒泡排序，也旨在在已经有序或接近有序的情况下提前终止排序，减少不必要的比较和交换。

- 与传统冒泡排序相似，它在每一轮中比较相邻元素并交换它们，将较大的元素逐步“冒泡”到数组的末尾。

- 在每一轮中，它还检查是否发生了交换。如果没有交换发生，说明数组已经有序，可以提前终止排序。

- 这种算法在某些情况下可以显著减少比较次数，但在最坏情况下仍然是 O(n^2)。

4. 插入排序（Insertion Sort）：

- 插入排序是一种简单而有效的排序算法，适用于小规模数据或者已经接近有序的数据。

- 它将数组分成已排序和未排序两部分，初始时已排序部分只包含一个元素（即第一个元素）。

- 然后，遍历未排序部分的元素，将每个元素插入已排序部分的合适位置，以保持已排序部分的有序性。

- 插入排序是稳定的排序算法，最坏情况下的时间复杂度是 O(n^2)，但在最好情况下（已经有序）的时间复杂度是 O(n)。

3. 测试结果（测试输入，测试输出）

输入:5 5 4 3 2 1

输出(测试每种排序方法):1 2 3 4 5

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

结果无明显问题,但存在细节问题,如需要析构函数来释放数组和名次数组的内存

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

```
#include<iostream>
using namespace std;
int n;
template <class T>
class MySort
{
private:
    T* a=new T[n];
    T* r=new T[n];
public:
    ~MySort(){delete[]a,delete[] r;}
    void set_element(T x,int index)
        {
            a[index]=x;
        }
    T get_element(int index)
        {
            return a[index];
        }
    T get_r_element(int index)
    {
        return r[index];
    }
public:
    void rank_compute()
```

```
{
    for(int i=0;i<n;i++)
    r[i]=0;
    for(int i=1;i<n;i++)
        for(int j=0;j<i;j++)
        if(a[j]<=a[i])
        r[i]++;
        else r[j]++;
}
void rank_sort()
{
    T *u= new T[n];
    for(int i=0;i<n;i++)
    u[r[i]]=a[i];
    for(int i=0;i<n;i++)
    a[i]=u[i];
    delete[] u;
}
void improve_seletion_sort()
{
    bool sort=false;
    for(int i=n-1;i>=1&&!sort;i--)
{
    sort=true;
    int indexOfMax=0;
    for(int j=1;j<i+1;j++)
    if(a[j]>a[indexOfMax])
    {
        indexOfMax=j;
    }
    else sort=false;
    swap(a[i],a[indexOfMax]);
```

```
    }
    }
    void improve_bubble_sort()
    {
        bool sort=false;
        for(int i=n-1;i>=1&&!sort;i--)
        {
            sort=true;
            for(int j=0;j<i;j++)
            if(a[j]>a[j+1])
            {
                swap(a[j],a[j+1]);
                sort=false;
            }
        }
    }
    void insert_sort()
    {
        for(int i=1;i<n;i++)
        {
            int x=a[i];
            int j;
            for(j=i-1;j>=0&&a[j]>x;j--)
                a[j+1]=a[j];
            a[j+1]=x;
        }
    }
};


int main()
{
```

```
    cin>>n;
    MySort<int> m_sort;
    for(int i=0;i<n;i++)
    {
        int x;
        cin>>x;
        m_sort.set_element(x,i);
    }
    m_sort.insert_sort();
    for(int i=0;i<n;i++)
    cout<<m_sort.get_element(i)<<" ";
    return 0;
}
```