排序算法的比较

一、实验要求

1. 算法和代码的设计与实现
   分别设计并实现插入排序、合并排序、快速排序的算法
2. 测试：设计测试数据集，编写测试程序，用于测试

   a)正确性：所实现的三种算法的正确性；

   b)算法复杂性：三种排序算法中，设计测试数据集，评价各个算法在算法复杂性上的表现；（最好情况、最差情况、平均情况）

   c)效率：在三种排序算法中，设计测试数据集，评价各个算法中比较的频率，腾挪的频率。
3. 撰写评价报告
   a)结合第二步的测试和实验结果，在理论上给予总结和评价三种排序算法在算法复杂性和效率上的表现。形成电子版实验报告。

二、实验环境

操作系统：Windows10

IDE：Visual Studio 2019

三、实验原理

3.1 插入排序

插入排序的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。比较是从有序序列的末尾开始，也就是想要插入的元素和已经有序的最大者开始比起，如果比它大则直接插入在其后面，否则一直往前找直到找到它该插入的位置。如果碰见一个和插入元素相等的，那么插入元素把想插入的元素放在相等元素的后面。所以，相等元素的前后顺序没有改变，插入排序是稳定的。

在最坏情况下，数组完全逆序，插入第2个元素时要考察前1个元素，插入第3个元素时，要考虑前2个元素，.....，插入第N个元素，要考虑前 `N - 1` 个元素。因此，最坏情况下的比较次数是 `1 + 2 + 3 + ... + (N - 1)`，等差数列求和，结果为 $N(N-1)/2$，所以最坏情况下的复杂度为 $O(N^2)$。

最好情况下，数组已经是有序的，每插入一个元素，只需要考查前一个元素，因此最好情况下，插入排序的时间复杂度为$O(N)$。

平均情况下的时间复杂度是 $O(N^2)$

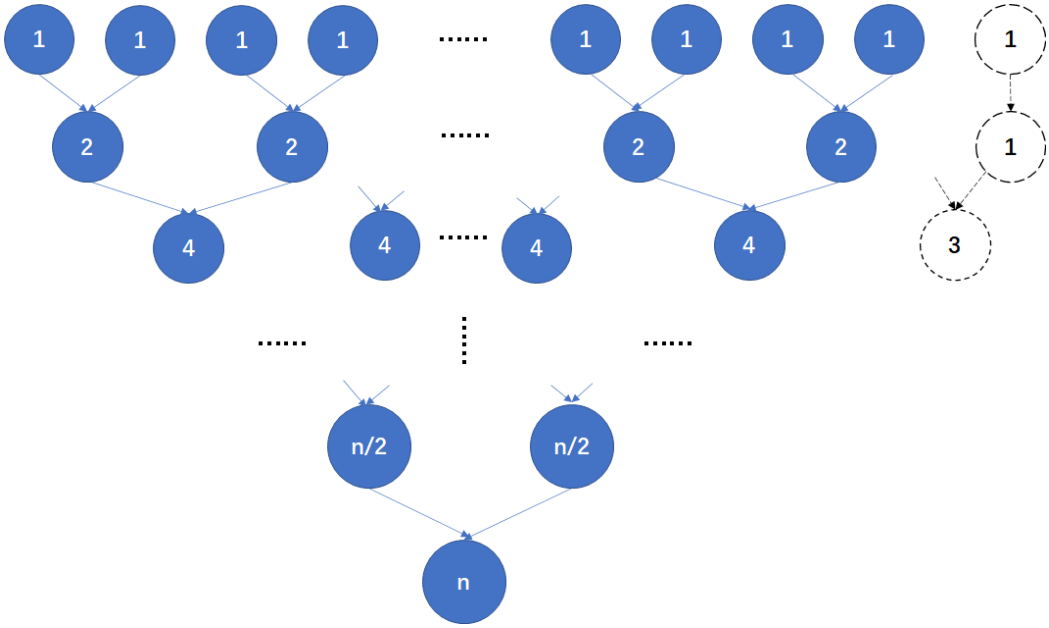
3.2 合并排序

合并排序的工作原理是利用分治的思想，将待排序元素分成大小大致相同的2个子集合，分别对2个子集合进行排序，最终将排好序的子集合合并成为所要求的排好序的集合。

首先将长度为1的n个数组相邻元素两两配对，构成了长度为2的n/2个数组，合并时用比较算法对这每个子数组中元素进行排序；

再将这些长度为2的n/2个数组两两合并，构成了长度为4，个数为n/4的子数组，合并时用比较算法对每个子数组元素排序。重复上述操作，直到形成长度为n，子数组个数=1的整个数组为止。

合并排序非递归的合并过程如下图所示：



对于特定长度n的序列，不论原序列的取值情况如何，合并过程中的子数组一定是从1、2、4.....一直到n（每一步的剩余子数组也相同），因此合并排序的最好与最坏复杂度相同。又因为树高为$logN$，每一行的合并过程开销均为$O(N)$，因此最好、最坏时间复杂度均为$O(NlogN)$。

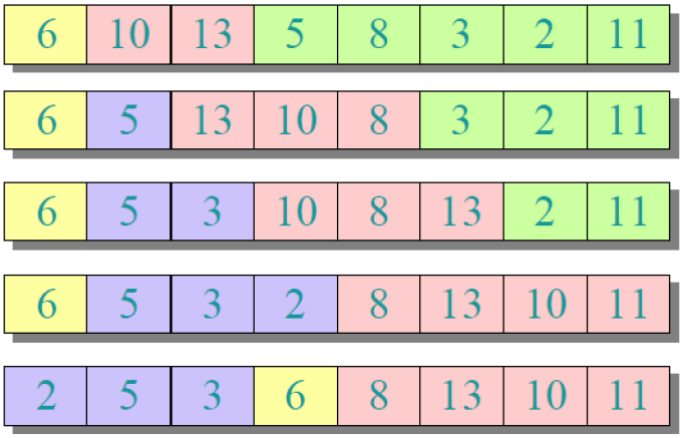
此外，合并过程中我们可以保证如果两个当前元素相等时，我们把处在前面的序列的元素保存在结果序列的前面，这样就保证了稳定性。所以，合并排序也是稳定的排序算法。

3.3 快速排序

快速排序的基本思想：

1. 先从数列中取出一个数作为基准数。
2. 分区过程，将比这个数大的数全放到它的右边，小于或等于它的数全放到它的左边。
3. 再对左右区间重复第二步，直到各区间只有一个数。

快速排序举例（基准数为A[0]=6）：



在最好情况下，每次选择的基准数都将原序列分成两个大小相同的子序列，再递归进入子序列排序。根据合并排序的分析，可知此时的时间复杂度为$O(NlogN)$。

在最坏情况下，每次选择的基准数都是原序列的最大或最小值，即每次将原序列分为一个大小为0的序列与大小为N-1的序列。因此将重复N次$O(N)$操作，此时的操作将退化为一个冒泡排序，时间复杂度为$O(N^2)$。

在随机选择基准数的情况下，期望的平均时间复杂度为$O(NlogN)$

四、程序设计

4.1 数据类定义

由于需要统计排序算法的腾挪次数与比较次数，于是新定义一个数据类，记录每个数据的值Value, 腾挪次数MoveTimes, 比较次数CompareTimes，并重定义该类的赋值运算、比较运算，在进行运算时将对应运算类型的计数值加一。

```
class Data
{
public:
    Data();
    ~Data();
    int Value, MoveTimes, CompareTimes;

    /*数据类初始化函数，在给Value赋值的同时清空MoveTimes和CompareTimes*/
    void Init(const int);

    /*赋值运算重载*/
    void operator=(const int);
    void operator=(const Data&);

    /*比较运算重载*/
    bool operator<(const int);
    bool operator<(const Data&);
    bool operator>(const int);
    bool operator>(const Data&);
    bool operator<=(const int);
    bool operator<=(const Data&);
```

```
    bool operator>=(const int);
    bool operator>=(const Data&);
};
```

```
void Data::Init(const int _Value)
{
    Value = _Value;
    MoveTimes = 0;
    CompareTimes = 0;
}

void Data::operator=(const int _Value)
{
    Value = _Value;
    MoveTimes++;
}

void Data::operator=(const Data& rhs)
{
    Value = rhs.Value;
    MoveTimes++;
}

bool Data::operator<(const int _Value)
{
    CompareTimes++;
    return Value < _Value;
}

bool Data::operator<(const Data& rhs)
{
    CompareTimes++;
    return Value < rhs.Value;

}

/*......*/
```

4.2 数据产生程序

定义数据产生类，传入需要生成的数列长度与需要的类型（有从小到大、从大到小、随机、对于快速排序最优的Middle In First队列四种），生成原始数据并返回数据开头指针。

```
#include "Data.h"
class DataCreate
```

```
{
public:
    /*定义产生数据类型，以适应多种复杂度情况测试需求*/
    enum SequenceType{
        SmallToLarge = 0,    //从小到大的有序数列
        Random,              //随机数列
        LargeToSmall,        //从大到小的有序数列
        MiddleInFirst,       //每个子序列中，最中间的数字在头尾的序列（用于
测试快速排序）
        Sepa                 //均分两边（用于测试合并排序）
    };

    /*序列构造函数，传入需要生成数列的长度与类型*/
    DataCreate(int, SequenceType _type = Random);

    /*序列析构函数，自动释放申请的空间*/
    ~DataCreate();

    /*返回shne'q*/
    Data* GetData();
private:
    /*用于生成MiddleInFirst类型序列*/
    void Adjust(int, int, int);

    /*用于生成Sepa类型序列*/
    void Separate();
    void anti_MergePass(int );
    void anti_Merge(int, int, int);

    void Swap(Data&, Data&);
    int N;  //需要随机生成的数据量
    SequenceType type; //需要随机生成的序列类型
    Data* data; //随机生成的数据数组
};
```

```
DataCreate::DataCreate(int n, SequenceType _type)
{
    N = n;
    data = new Data[N];
    type = _type;
}

Data* DataCreate::GetData()
{
    Time++;
    srand((int)(time(0)+ Time));  // 产生随机种子
    switch (type)
```

```
    {
    case DataCreate::SmallToLarge:  //从小到大的有序数列
        for (int i = 0; i < N; i++)
            data[i] = N - i;
        break;
    case DataCreate::Random:        //随机数列
        for (int i = 0; i < N; i++)
        {
            data[i].Init(rand() % 100000000);
        }
        break;
    case DataCreate::LargeToSmall:  //从大到小的有序数列
        for (int i = 0; i < N; i++)
            data[i] = i + 1;
        break;
    case DataCreate::MiddleInFirst: //每个子序列中，最中间的数字在头尾的序列（用于测试快速排序）
        for (int i = 0; i < N; i++)
            data[i] = i + 1;
        Adjust(0, N - 1, 1);
        for (int i = 0; i < N; i++)
        {
            data[i].CompareTimes = 0;
            data[i].MoveTimes = 0;
        }
        break;
    default:
        break;
    }
    return data;
}

void DataCreate::Swap(Data& x, Data& y)
{
    int temp = x.Value;
    x = y;
    y = temp;
}

/*将l到r的序列调整为Middle值在最左或最有的序列，belong代表当前子序列是父序列的左边还是右边*/
void DataCreate::Adjust(int l, int r, int belong)
{
    if (l >= r)
        return;
    int base = (l + r) / 2;
    if (belong == 0)            //如果是父序列的左边
    {
        Swap(data[r], data[base]);      //在进行快速排序算法时，将会选择data[r]为基准值，因此将data[base]换到data[r]的位置，便于子序列均等划分
        if (l < r)
            sort(data + l, data + r - 1);
```

```
        Adjust(l, base - 1, 0);
        Adjust(base, r - 1, 1);
    }
    else                            //如果是父序列的右边
    {
        Swap(data[l], data[base]);      //在进行快速排序算法时，将会选择data[l]为基准值，因此将data[base]换到data[l]的位置，便于子序列均等划分
        if (l < r)
            sort(data + l + 1, data + r);
        Adjust(l + 1, base, 0);
        Adjust(base + 1, r, 1);
    }

}

void DataCreate::anti_Merge(int l, int m, int r)
{
    int i = m, j = r, k = r;
    while ((i >= l) && (j >= m + 1))
    {
        A[i--] = data[k--].Value;
        A[j--] = data[k--].Value;
    }

    if (i < l)
    {
        for (int q = j; q >= m + 1; q--)
            A[q] = data[k--].Value;
    }
    else
    {
        for (int q = i; q >= l; q--)
            A[q] = data[k--].Value;
    }
    for (int i = l; i <= r; ++i)
        data[i] = A[i];
}

void DataCreate::anti_MergePass(int s)
{
    int i = 0;
    while (i + 2 * s - 1 < N)
    {
        anti_Merge(i, i + s - 1, i + 2 * s - 1);
        i = i + 2 * s;
    }
    if (i + s - 1 < N - 1)
    {
        anti_Merge(i, i + s - 1, N - 1);
    }

}
```

```
void DataCreate::Separate()
{
    A = new int[N];
    int maxS = 1;
    while (maxS < N) maxS *= 2;
    maxS /= 2;
    int s = maxS;
    while (s >= 1)
    {
        anti_MergePass(s);
        s /= 2;
    }
    delete[] A;
}

DataCreate::~DataCreate()
{
    delete[] data;
}
```

关于生成Middle In First数列的Adjust设计思路，将在后文描述快速排序的最优情况中说明。

4.3 白盒测试程序

白盒测试主要测试算法是否正常通过每一个分支路径。定义分支遍历校验结构体`BranchStruct`，在算法开始运行时将自己的所有可能出现的分支存入`BranchStruct branch[]`数组中，并存下每个分支的id记录。用`passed`记录每个每个分支是否经过，初始情况下置为0。在运行过程中如果经过i号分支，就将对应`branch[i]`的passed置为1。算法运行完成后，检查每个分支是否经过，并输出检查结果。

```
#pragma once
#include <string>
#include <vector>
using namespace std;
class Branch
{
public:
    struct BranchStruct
    {
        string branch_name;             //分支名称
        int passed;                     //是否已经过
        BranchStruct() : branch_name(""), passed(0){};
        BranchStruct(string _name) :
            branch_name(_name), passed(0){};
    };
```

```
    Branch();
    ~Branch();
    /*添加原始分支*/
    int AddBranch(string);

    /*经过分支记录*/
    void Pass(int);

    /*打印分支遍历结果*/
    void showResult();

    /*分支总数*/
    int tot_branch;

    /*存储所有分支*/
    BranchStruct branch[100];

};
```

```
int Branch::AddBranch(string condition)
{
    branch[++tot_branch] = BranchStruct(condition);
    return tot_branch;      //向添加者返回当前分支的id
}

void Branch::Pass(int id)
{
    branch[id].passed = 1;
}

void Branch::showResult()
{
    int allPassed = 1;
    cout << "分支遍历结果如下:" << endl;
    for (int i = 1; i <= tot_branch; ++i)
    {
        if (branch[i].passed == 0)
        {
            allPassed = 0;
            cout << branch[i].branch_name << " : " << "×" << endl;
        }
        else
            cout << branch[i].branch_name << " : " << "√" << endl;
    }
    if (allPassed == 1)
        cout << "Congratulations! 通过了所有分支:)" << endl;
    else
    {
```

```
        cout << "没有所有分支:(  未通过的分支如下:" << endl;
        for (int i = 1; i <= tot_branch; ++i)
            if (branch[i].passed == 0)
                cout << branch[i].branch_name << endl;
    }
    return;
}
```

4.4 黑盒测试程序

黑盒测试主要测试算法测试算法的排序功能正确性，并打印出比较次数和移动次数，以及运行时间效率。

正确性：在main函数中实现，检查排序结果是否有序

```
cout << "检测排序结果从1到" << n << "是否有序:" << endl;
Sorted = 1;
FOR(i, 0, n - 2)
if (B[i] > B[i + 1])
{
    Sorted = 0;
    break;
}
if (Sorted)
cout << "检测结果: 排序正确，数列有序" << endl;
else cout << "[Error!] 排序错误" << endl;
sort.showBranchTest();
```

比较次数和移动次数：新定义一个数据类，记录每个数据的值Value, 腾挪次数MoveTimes, 比较次数CompareTimes，并重定义该类的赋值运算、比较运算，在进行运算时将对应运算类型的计数值加一

```
void Data::Init(const int _Value)
{
    Value = _Value;
    MoveTimes = 0;
    CompareTimes = 0;
}

void Data::operator=(const int _Value)
{
    Value = _Value;
    MoveTimes++;
}

void Data::operator=(const Data& rhs)
{
    Value = rhs.Value;
    MoveTimes++;
}
```

```
bool Data::operator<(const int _Value)
{
    CompareTimes++;
    return Value < _Value;
}

bool Data::operator<(const Data& rhs)
{
    CompareTimes++;
    return Value < rhs.Value;

}
bool Data::operator>(const int _Value)
{
    CompareTimes++;
    return Value > _Value;
}

bool Data::operator>(const Data& rhs)
{
    CompareTimes++;
    return Value > rhs.Value;
}

bool Data::operator<=(const int _Value)
{
    CompareTimes++;
    return Value <= _Value;
}

bool Data::operator<=(const Data& rhs)
{
    CompareTimes++;
    return Value <= rhs.Value;

}
bool Data::operator>=(const int _Value)
{
    CompareTimes++;
    return Value >= _Value;
}

bool Data::operator>=(const Data& rhs)
{
    CompareTimes++;
    return Value >= rhs.Value;
}
```

运行时间效率：为了使统计结果更加准确，新定义一个能够记录到微秒时间模块ustime

```
#ifdef _WIN32
#include <windows.h>
#else
#include <time.h>
#endif  // _WIND32

// 定义64位整形
#if defined(_WIN32) && !defined(CYGWIN)
typedef __int64 int64_t;
#else
typedef long long int64t;
#endif  // _WIN32

int64_t GetSysTimeMicros();
```

```
// 获取系统的当前时间，单位微秒(us)
int64_t GetSysTimeMicros()
{
#ifdef _WIN32
    // 从1601年1月1日0:0:0:000到1970年1月1日0:0:0:000的时间(单位100ns)
#define EPOCHFILETIME   (116444736000000000UL)
    FILETIME ft;
    LARGE_INTEGER li;
    int64_t tt = 0;
    GetSystemTimeAsFileTime(&ft);
    li.LowPart = ft.dwLowDateTime;
    li.HighPart = ft.dwHighDateTime;
    // 从1970年1月1日0:0:0:000到现在的微秒数(UTC时间)
    tt = (li.QuadPart - EPOCHFILETIME) / 10;
    return tt;
#else
    timeval tv;
    gettimeofday(&tv, 0);
    return (int64_t)tv.tv_sec * 1000000 + (int64_t)tv.tv_usec;
#endif // _WIN32
    return 0;
}
```

通过记录算法运行前后的系统时间，求出差值，即可得到算法运行时间。

4.5 算法程序

4.5.1 算法类成员

```
#include "Data.h"
#include "Branch.h"
#include "ustime.h"
#define LL long long
class XXXXSort
{
public:
    /*由外部传入需要需排序的序列长度和数据指针*/
    XXXXSort(int, Data*);
    ~XXXXSort();

    /*执行排序算法*/
    void Run();

    /*统计所有数据上的腾挪次数总和*/
    LL CountMove();

    /*统计所有数据上的比较次数总和*/
    LL CountCompare();

    /*展示分支测试结果*/
    void showBranchTest();

    /*记录算法运行时间*/
    double RunTime;
private:
    int N;

    /*数据存储单元*/
    Data* A;

    /*定义分支统计类*/
    Branch branch;

    /*存储所有分支id*/
    int b1, b2, b3, //(j >= 0) && (A[j] > key)
    b5, b6,.....
           .....
};

LL XXXXSort::CountMove()
{
    LL Sum = 0;
    FOR(i, 0, N - 1)
        Sum += A[i].MoveTimes;
    return Sum;
}

LL XXXXSort::CountCompare()
{
```

```
    LL Sum = 0;
    FOR(i, 0, N - 1)
        Sum += A[i].CompareTimes;
    return Sum;
}

void XXXXSort::showBranchTest()
{
    branch.showResult();
}
```

4.5.2 插入排序

```
InsertSort::InsertSort(int n, Data* _A)
{
    N = n;
    A = _A;

    /*初始化所有分支*/
    b1 = branch.AddBranch("(j >= 0) = True , (A[j] > key) = True");
    b2 = branch.AddBranch("(j >= 0) = True , (A[j] > key) =
Flase");
    b3 = branch.AddBranch("(j >= 0) = False");  //因j < 0时 A[j]无意
义，此时无需考虑A[j]的情况
}

void InsertSort::Run()
{
    long long BeginTime = GetSysTimeMicros();
    FOR (i, 1, N - 1) {
        int key = A[i].Value;
        int j = i - 1;
        while ((j >= 0) && (A[j] > key)) {
            branch.Pass(b1);
            A[j + 1] = A[j];
            j--;
        }
        if ((j >= 0) && !(A[j] > key))
            branch.Pass(b2);
        if (!(j >= 0))
            branch.Pass(b3);
        A[j + 1] = key;
    }
    RunTime = (GetSysTimeMicros() - BeginTime) / 1000.0;
}
```

4.5.3 合并排序

```
MergeSort::MergeSort(int n, Data* _A)
{
    N = n;
    A = _A;

    /*初始化所有分支*/
    b1 = branch.AddBranch("(i <= m) = True , (j <= r) = True");
    b2 = branch.AddBranch("(i <= m) = True , (j <= r) = Flase");
    b3 = branch.AddBranch("(i <= m) = False , (j <= r) = True");

    b5 = branch.AddBranch("(c[i] <= c[j]) = True");
    b6 = branch.AddBranch("(c[i] <= c[j]) = Flase");

    b7 = branch.AddBranch("(i > m) = True");
    b8 = branch.AddBranch("(i > m) = Flase");

    b9 = branch.AddBranch("(i + 2 * s - 1 < n) = True");
    b10 = branch.AddBranch("(i + 2 * s - 1 < n) = Flase");

    b11 = branch.AddBranch("(i + s - 1 < n - 1) = True");
    b12 = branch.AddBranch("(i + s - 1 < n - 1) = Flase");

    b13 = branch.AddBranch("(s < N) = True");
    b14 = branch.AddBranch("(s < N) = Flase");
}

void MergeSort::Merge(Data* c, Data* d, int l, int m, int r)
{
    int i = l, j = m + 1, k = l;
    while ((i <= m) && (j <= r))
    {
        branch.Pass(b1);
        if (c[i] <= c[j])
        {
            branch.Pass(b5);
            d[k++] = c[i++];
        }
        else
        {
            branch.Pass(b6);
            d[k++] = c[j++];
        }
    }
    if ((i <= m) && !(j <= r)) branch.Pass(b2);
    if (!(i <= m) && (j <= r)) branch.Pass(b3);

    if (i > m)
    {
        branch.Pass(b7);
        FOR(q, j, r)
```

```
            d[k++] = c[q];
    }
    else
    {
        branch.Pass(b8);
        FOR(q, i, m)
            d[k++] = c[q];
    }
}

void MergeSort::MergePass(Data* x, Data* y, int s, int n)
{
    int i = 0;
    while (i + 2 * s - 1 < n)
    {
        branch.Pass(b9);
        Merge(x, y, i, i + s - 1, i + 2 * s - 1);
        i = i + 2 * s;
    }
    branch.Pass(b10);
    if (i + s - 1 < n - 1 )
    {
        branch.Pass(b11);
        Merge(x, y, i, i + s - 1, n - 1);
    }
    else
    {
        branch.Pass(b12);
        FOR(j, i, n - 1)
            y[j] = x[j];
    }
}

void MergeSort::Run()
{
    B = new Data[N];
    double BeginTime = GetSysTimeMicros();
    int s = 1;
    while(s < N)
    {
        branch.Pass(b13);
        MergePass(A, B, s, N);
        s += s;
        MergePass(B, A, s, N);
        s += s;
    }
    branch.Pass(b14);
    RunTime = (GetSysTimeMicros() - BeginTime) / 1000.0;
}
```

4.5.4 快速排序

```
QuickSort::QuickSort(int n, Data* _A)
{
    N = n;
    A = _A;

    /*初始化所有分支*/
    b1 = branch.AddBranch("(l >= r) = True");
    b2 = branch.AddBranch("(l >= r) = Flase");

    b3 = branch.AddBranch("(A[j] <= x) = True");
    b4 = branch.AddBranch("(A[j] <= x) = Flase");
}

void QuickSort::Run()
{
    double BeginTime = GetSysTimeMicros();
    srand((int)time(0));
    Sort(0, N - 1);
    RunTime = (GetSysTimeMicros() - BeginTime) / 1000.0;
}

void QuickSort::Swap(Data& x, Data& y)
{
    int temp = x.Value;
    x = y;
    y = temp;
}

void QuickSort::Sort(int l, int r)
{

    if (l >= r)
    {
        A[0] > -1;
        branch.Pass(b1);
        return;
    }
    branch.Pass(b2);
    int i, x, base = l;

    /*避免最差情况的优化开关*/
    base = l + rand() % (r - l + 1);
    if (l != base) Swap(A[l], A[base]);

    x = A[l].Value;
    i = l;
    FOR(j, l + 1, r)
    {

        if (A[j] <= x)
```

```
        {
            branch.Pass(b3);
            i++;
            if (i != j) Swap(A[i], A[j]);
        }
        else
        {
            branch.Pass(b4);
        }
    }

    Swap(A[l], A[i]);
    Sort(l, i - 1);
    Sort(i + 1, r);
}
```

五、程序测试

5.1 正确性测试

正确性测试分为两部分：一是验证排序是否成功，而是验证是否便利了所有分支。

5.1.1 插入排序

N=10

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:0
N=10
原始序列：25736 12618 15553 30300 25080 4231 16545 3376 8283 3584

----------------------插入排序-----------------------
排序结果：3376 3584 4231 8283 12618 15553 16545 25080 25736 30300
检测排序结果从1到10是否有序:
检测结果：排序正确，数列有序
分支遍历结果如下:
(j >= 0) = True , (A[j] > key) = True : √
(j >= 0) = True , (A[j] > key) = Flase : √
(j >= 0) = False : √
Congratulations! 通过了所有分支:)
```

N=100

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:0
N=100
原始序列：25936 12910 23923 23602 31677 14386 9797 21421 32451 21768 14163 24082 18978 18808 5013 25803 2742 5059 29152
28334 27142 15650 25874 30570 6171 3151 21648 22053 10126 16710 22770 30563 28645 15949 20732 28202 27168 16946 17104 82
92 15246 6100 23907 4740 25317 24292 11822 5740 15250 20973 24358 24974 19278 11666 4333 20050 3570 7448 9546 19640 1319
6 20516 3543 17983 1202 7621 29433 30981 24854 22708 18197 3631 3163 14555 6657 28555 15655 32277 16569 8308 8826 12341
9556 7671 21310 8265 21302 2612 23583 3872 4998 9990 24476 7433 11747 25574 31838 26334 8530 780

----------------------插入排序------------------------
排序结果：780 1202 2612 2742 3151 3163 3543 3570 3631 3872 4333 4740 4998 5013 5059 5740 6100 6171 6657 7433 7448 7621 7
671 8265 8292 8308 8530 8826 9546 9556 9797 9990 10126 11666 11747 11822 12341 12910 13196 14163 14386 14555 15246 15250
 15650 15655 15949 16569 16710 16946 17104 17983 18197 18808 18978 19278 19640 20050 20516 20732 20973 21302 21310 21421
 21648 21768 22053 22708 22770 23583 23602 23907 23923 24082 24292 24358 24476 24854 24974 25317 25574 25803 25874 25936
 26334 27142 27168 28202 28334 28555 28645 29152 29433 30563 30570 30981 31677 31838 32277 32451
检测排序结果从1到100是否有序:
检测结果：排序正确，数列有序
分支遍历结果如下:
(j >= 0) = True , (A[j] > key) = True : √
(j >= 0) = True , (A[j] > key) = Flase : √
(j >= 0) = False : √
Congratulations! 通过了所有分支:)
```

N=1000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:0
N=1000
```

（省略构造序列）

```
检测排序结果从1到1000是否有序:
检测结果：排序正确，数列有序
分支遍历结果如下:
(j >= 0) = True , (A[j] > key) = True : √
(j >= 0) = True , (A[j] > key) = Flase : √
(j >= 0) = False : √
Congratulations! 通过了所有分支:)
```

N=10000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:0
N=10000
```

（省略构造序列）

```
检测排序结果从1到10000是否有序:
检测结果：排序正确，数列有序
分支遍历结果如下:
(j >= 0) = True , (A[j] > key) = True : √
(j >= 0) = True , (A[j] > key) = Flase : √
(j >= 0) = False : √
Congratulations! 通过了所有分支:)
```

N=100000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:0
N=100000
```

（省略构造序列）

```
检测排序结果从1到100000是否有序:
检测结果：排序正确，数列有序
分支遍历结果如下:
(j >= 0) = True ， (A[j] > key) = True ： √
(j >= 0) = True ， (A[j] > key) = Flase ： √
(j >= 0) = False ： √
Congratulations! 通过了所有分支:)
```

5.1.2 合并排序

N=10

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:1
N=10
原始序列：27353 24663 10957 14089 18987 14650 17112 28935 22289 7714

----------------------合并排序------------------------
排序结果：7714 10957 14089 14650 17112 18987 22289 24663 27353 28935
检测排序结果从1到10是否有序:
检测结果：排序正确，数列有序
分支遍历结果如下:
(i <= m) = True ， (j <= r) = True ： √
(i <= m) = True ， (j <= r) = Flase ： √
(i <= m) = False ， (j <= r) = True ： √
(c[i] <= c[j]) = True ： √
(c[i] <= c[j]) = Flase ： √
(i > m) = True ： √
(i > m) = Flase ： √
(i + 2 * s - 1 < n) = True ： √
(i + 2 * s - 1 < n) = Flase ： √
(i + s - 1 < n - 1) = True ： √
(i + s - 1 < n - 1) = Flase ： √
(s < N) = True ： √
(s < N) = Flase ： √
Congratulations! 通过了所有分支:)
```

N=100

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:1
N=100
原始序列：27464 29661 28515 13041 9234 17087 24632 27712 26627 28056 10479 11928 25284 29241 3489 13565 3419 10080 1192
27832 28415 19001 7693 3833 24618 1316 32576 32193 1009 21433 18188 26623 27537 26633 28105 30970 6218 29991 30164 5870
20336 744 60 29946 29887 12006 6590 31801 24303 32459 21492 20528 23176 10188 32457 29288 20216 28531 5680 18779 14818 2
466 14358 3733 29882 6101 13972 26643 30863 11687 6762 12527 5159 32408 10665 13631 18513 6413 18095 25914 15991 21144 1
5192 17308 3443 18810 17523 4940 16547 8038 29043 2108 21824 20872 11367 32657 8683 20934 32301 773

----------------------合并排序------------------------
排序结果：60 744 773 1009 1192 1316 2108 2466 3419 3443 3489 3733 3833 4940 5159 5680 5870 6101 6218 6413 6590 6762 7693
 8038 8683 9234 10080 10188 10479 10665 11367 11687 11928 12006 12527 13041 13565 13631 13972 14358 14818 15192 15991 16
547 17087 17308 17523 18095 18188 18513 18779 18810 19001 20216 20336 20528 20872 20934 21144 21433 21492 21824 23176 24
303 24618 24632 25284 25914 26623 26627 26633 26643 27464 27537 27712 27832 28056 28105 28415 28515 28531 29043 29241 29
288 29661 29882 29887 29946 29991 30164 30863 30970 31801 32193 32301 32408 32457 32459 32576 32657
检测排序结果从1到100是否有序:
检测结果：排序正确，数列有序
分支遍历结果如下:
(i <= m) = True ， (j <= r) = True ： √
(i <= m) = True ， (j <= r) = Flase ： √
(i <= m) = False ， (j <= r) = True ： √
(c[i] <= c[j]) = True ： √
(c[i] <= c[j]) = Flase ： √
(i > m) = True ： √
(i > m) = Flase ： √
(i + 2 * s - 1 < n) = True ： √
(i + 2 * s - 1 < n) = Flase ： √
(i + s - 1 < n - 1) = True ： √
(i + s - 1 < n - 1) = Flase ： √
(s < N) = True ： √
(s < N) = Flase ： √
Congratulations! 通过了所有分支:)
```

N=1000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:1
N=1000
原始序列：27575 1890 13305 11994 32250 19524 32151 26489 30965 15630 5450 13426 24341 1992 4359 13236 30494 13945 30669
8471 23326 21065 26677 12954 29600 16867 3403 2542 13089 26397 21916 24377 9672 16347 8336 18708 10577 3632 22010 17877
24907 19399 7150 27156 15235 27778 7891 22632 6336 13408 8120 26226 28080 9380 15596 4893 28707 17460 10440 12835 9335 1
8239 17104 28884 24403 23074 18171 6163 3993 5005 1870 12193 25749 20961 25519 26270 9339 2013 29689 25932 18192 15482 6
```

（省略构造序列）

```
98 30669 30674 30732 30805 30808 30862 30919 30965 30967 31052 31108 31154 31163 31199 31207 31207 31318 31368 31369 314
02 31426 31444 31481 31493 31508 31533 31552 31563 31624 31679 31686 31702 31763 31763 31794 31794 31802 31835 31941 320
00 32048 32061 32079 32099 32102 32110 32151 32182 32250 32292 32334 32358 32365 32383 32407 32479 32492 32509 32529 325
38 32608 32642 32708 32714 32746 32756
检测排序结果从1到1000是否有序:
检测结果: 排序正确，数列有序
分支遍历结果如下:
(i <= m) = True , (j <= r) = True : √
(i <= m) = True , (j <= r) = Flase : √
(i <= m) = False , (j <= r) = True : √
(c[i] <= c[j]) = True : √
(c[i] <= c[j]) = Flase : √
(i > m) = True : √
(i > m) = Flase : √
(i + 2 * s - 1 < n) = True : √
(i + 2 * s - 1 < n) = Flase : √
(i + s - 1 < n - 1) = True : √
(i + s - 1 < n - 1) = Flase : √
(s < N) = True : √
(s < N) = Flase : √
Congratulations! 通过了所有分支:)
```

N=10000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:1
N=10000
原始序列：27791 23157 12692 27309 24884 16544 8197 29896 6618 3073 28456 16333 10947 12706 27250 10670 31010 17594 20424
 21003 23085 26999 1848 17165 10356 25850 743 8594 17265 28324 23370 10378 11616 9872 31641 29600 15185 21856 29313 2383
8 2939 22845 15130 29450 13779 25625 4632 18325 27357 28471 11077 8375 29890 2030 23343 19219 20132 22954 2333 5152 1796
6 6451 32072 12170 23407 13618 24392 3030 21224 28657 30925 23110 11747 18018 4239 19964 32007 18531 29064 22113 16682 1
7984 29280 22728 20070 30585 4393 11599 25967 32454 30540 25350 30499 28505 31171 3924 18019 14179 27017 11414 23586 508
```

（省略构造序列）

```
32401 32407 32408 32409 32411 32412 32414 32425 32425 32426 32434 32438 32440 32442 32443 32443 32444 32446 32447 32449
32454 32458 32458 32459 32461 32464 32468 32469 32476 32476 32480 32484 32486 32489 32491 32496 32501 32501 32507 32508
32509 32515 32517 32530 32531 32532 32532 32533 32534 32540 32545 32546 32549 32556 32557 32559 32565 32566 32566 32574
32579 32588 32589 32593 32595 32597 32598 32599 32612 32620 32622 32628 32640 32647 32653 32653 32656 32657 32662 32664
32667 32667 32669 32673 32675 32678 32681 32681 32682 32684 32689 32693 32699 32703 32707 32710 32713 32715 32717 32723
32725 32735 32737 32737 32741 32747 32750 32752 32754 32754 32759 32760 32761 32762
检测排序结果从1到10000是否有序:
检测结果: 排序正确，数列有序
分支遍历结果如下:
(i <= m) = True , (j <= r) = True : √
(i <= m) = True , (j <= r) = Flase : √
(i <= m) = False , (j <= r) = True : √
(c[i] <= c[j]) = True : √
(c[i] <= c[j]) = Flase : √
(i > m) = True : √
(i > m) = Flase : √
(i + 2 * s - 1 < n) = True : √
(i + 2 * s - 1 < n) = Flase : √
(i + s - 1 < n - 1) = True : √
(i + s - 1 < n - 1) = Flase : √
(s < N) = True : √
(s < N) = Flase : √
Congratulations! 通过了所有分支:)
```

N=100000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:1
N=100000
```

（省略构造序列）

```
2 32743 32743 32743 32743 32743 32743 32744 32744 32746 32747 32747 32747 32747 32748 32748 32748 32748 32749 32750 3275
0 32751 32751 32751 32751 32752 32752 32752 32752 32752 32753 32753 32754 32754 32754 32754 32754 32754 32754 32755 3275
5 32755 32755 32756 32756 32756 32756 32756 32757 32757 32757 32757 32757 32757 32757 32757 32758 32759 32759 32759 32759 3275
9 32759 32759 32760 32760 32760 32760 32760 32760 32761 32761 32761 32762 32763 32763 32764 32764 32764 32765 32765 32765 3276
6 32767 32767 32767 32767 32767
检测排序结果从1到100000是否有序:
检测结果: 排序正确，数列有序
分支遍历结果如下:
(i <= m) = True , (j <= r) = True : √
(i <= m) = True , (j <= r) = Flase : √
(i <= m) = False , (j <= r) = True : √
(c[i] <= c[j]) = True : √
(c[i] <= c[j]) = Flase : √
(i > m) = True : √
(i > m) = Flase : √
(i + 2 * s - 1 < n) = True : √
(i + 2 * s - 1 < n) = Flase : √
(i + s - 1 < n - 1) = True : √
(i + s - 1 < n - 1) = Flase : √
(s < N) = True : √
(s < N) = Flase : √
Congratulations! 通过了所有分支:)
```

N=1000000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:1
N=1000000
```

（省略构造序列）

```
检测排序结果从1到1000000是否有序:
检测结果: 排序正确，数列有序
分支遍历结果如下:
(i <= m) = True , (j <= r) = True : √
(i <= m) = True , (j <= r) = Flase : √
(i <= m) = False , (j <= r) = True : √
(c[i] <= c[j]) = True : √
(c[i] <= c[j]) = Flase : √
(i > m) = True : √
(i > m) = Flase : √
(i + 2 * s - 1 < n) = True : √
(i + 2 * s - 1 < n) = Flase : √
(i + s - 1 < n - 1) = True : √
(i + s - 1 < n - 1) = Flase : √
(s < N) = True : √
(s < N) = Flase : √
Congratulations! 通过了所有分支:)
```

N=10000000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:1
N=10000000
```

（省略构造序列）

```
排序结果:
检测排序结果从1到10000000是否有序:
检测结果: 排序正确, 数列有序
分支遍历结果如下:
(i <= m) = True , (j <= r) = True : √
(i <= m) = True , (j <= r) = Flase : √
(i <= m) = False , (j <= r) = True : √
(c[i] <= c[j]) = True : √
(c[i] <= c[j]) = Flase : √
(i > m) = True : √
(i > m) = Flase : √
(i + 2 * s - 1 < n) = True : √
(i + 2 * s - 1 < n) = Flase : √
(i + s - 1 < n - 1) = True : √
(i + s - 1 < n - 1) = Flase : √
(s < N) = True : √
(s < N) = Flase : √
Congratulations! 通过了所有分支:)
```

5.1.3 快速排序

N=10

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:2
N=10
原始序列：31092 10820 18202 8695 32694 21540 9157 19541 20924 13297

----------------------快速排序------------------------
排序结果：8695 9157 10820 13297 18202 19541 20924 21540 31092 32694
检测排序结果从1到10是否有序:
检测结果: 排序正确, 数列有序
分支遍历结果如下:
(l >= r) = True : √
(l >= r) = Flase : √
(A[j] <= x) = True : √
(A[j] <= x) = Flase : √
Congratulations! 通过了所有分支:)
```

N=100

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:2
N=100
原始序列：31458 2226 20141 16810 8279 2582 30071 19367 2446 12841 26197 261 1992 14774 12600 17154 1056 32512 11938 2819
9 30409 12284 20746 30238 25496 25789 9602 29635 32706 15920 18327 22909 25830 23817 19577 16847 30030 17661 26972 3114
21874 50 22813 15366 7869 5876 17705 20017 12180 12201 13723 30822 15509 27399 19620 1825 1820 20612 20784 19890 23820 2
4334 19654 23696 17868 5629 16560 29169 143 2630 13917 22679 14297 23537 12035 10477 3647 8141 13960 8274 26734 4450 758
7 693 6254 8487 7999 14946 20008 13045 29984 10779 28966 8380 8136 25963 18331 21877 26012 17207

----------------------快速排序------------------------
排序结果：50 143 261 693 1056 1820 1825 1992 2226 2582 2630 3114 3647 4450 5629 5876 6254 7587 7869 7999 8136 8141
8274 8279 8380 8487 9602 10477 10779 11938 12035 12180 12201 12284 12600 12841 13045 13723 13917 13960 14297 14774 14946
 15366 15509 15920 16560 16810 16847 17154 17207 17661 17705 17868 18327 18331 19367 19577 19620 19654 19890 20008 20017
 20141 20612 20746 20784 21874 21877 22679 22813 22909 23537 23696 23817 23820 24334 25496 25789 25830 25963 26012 26197
 26734 26972 27399 28199 28966 29169 29635 29984 30030 30071 30238 30409 30822 31458 32512 32706
检测排序结果从1到100是否有序:
检测结果: 排序正确, 数列有序
分支遍历结果如下:
(l >= r) = True : √
(l >= r) = Flase : √
(A[j] <= x) = True : √
(A[j] <= x) = Flase : √
Congratulations! 通过了所有分支:)
```

N=1000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:2
N=1000
原始序列：31605 27152 4829 18315 13683 15445 6292 18711 24571 31090 19542 2243 15202 5694 28207 10936 28218 9680 28784 5
466 19819 20798 8286 18215 27270 31914 18215 17377 23637 15744 22298 23791 30135 1529 30998 22785 24234 10724 3651 16115
 17322 8357 31233 18420 3898 10366 18462 15591 25024 12045 844 17161 15253 8982 20434 23508 3421 4995 7809 7204 8852 162
97 3050 24217 6762 31949 16334 30012 25297 5351 24790 7781 27092 677 28804 16604 31018 11955 9066 13118 28683 4666 2458
4770 20290 25255 7005 27773 19332 10925 32296 16323 23040 29837 5579 6479 27447 27029 13804 20357 18446 2790 1972 8502 1
7697 26491 30664 15299 16475 29543 16507 27782 7364 25694 25666 365 12441 14199 11368 7400 22946 17456 2 262 11200 179 1
0022 5595 28102 23384 20056 12068 22665 19818 24133 21806 19515 2862 31668 5363 28660 29985 17337 7046 28562 11906 6187
8370 19395 32016 4185 25822 28768 22037 25081 17972 1277 15523 22113 12547 11198 21790 8186 28513 8812 20986 9201 7188 3
1175 7622 3119 15539 22302 26218 13708 18184 29726 28077 20031 5561 20712 17146 14489 4135 19558 19043 1524 2455 5854 13
542 7654 7056 12999 19951 26042 32643 30684 16927 31776 11186 19568 17228 23843 14154 2305 11721 604 18898 14863 4839 21
349 6601 13358 19643 31377 16011 11867 32546 29146 22901 22156 10755 23579 23113 5259 12500 32384 25300 28419 31584 2821
8 3534 18139 5063 18039 7425 23748 26054 31377 21166 21774 824 6277 31276 11712 4573 4008 25554 23271 18011 20373 22551
32046 4226 21354 17912 26287 23764 27956 12442 25097 3909 26023 8241 28595 6975 17427 6688 28048 23785 21139 25698 23747
 6057 12936 6184 14788 15848 5445 18371 6835 26133 27916 15301 2029 4332 10810 2915 17118 9464 22718 5150 28438 17238 12
751 5283 20894 22973 30175 28399 6796 5563 8906 21942 27861 7489 23124 26886 6243 12386 3282 1576 14951 6232 15399 20167
 14321 1992 27378 11091 18110 20551 3078 19624 30800 4380 1563 28907 31556 17223 8131 28597 1761 18115 26527 20852 9453
21769 27858 30222 19908 31957 23478 3783 2731 2860 21198 28516 4705 1438 1020 31376 5111 6601 30196 11648 9186 9758 3121
```

（省略构造序列）

```
7 18371 18420 18446 18462 18479 18589 18608 18618 18618 18652 18687 18693 18711 18717 18735 18775 18802 18898 18948 1899
4 19043 19062 19237 19286 19289 19332 19340 19395 19425 19454 19506 19515 19541 19542 19551 19558 19568 19592 19624 1964
3 19772 19781 19792 19818 19819 19827 19899 19908 19951 20003 20031 20056 20071 20078 20090 20123 20127 20137 20140 2016
7 20171 20215 20250 20264 20271 20290 20293 20331 20332 20332 20357 20373 20428 20434 20467 20487 20514 20530 20546 2054
8 20551 20620 20656 20662 20666 20683 20712 20798 20813 20852 20894 20986 20992 21007 21015 21065 21102 21131 21139 2116
6 21198 21245 21286 21304 21312 21349 21354 21373 21380 21420 21462 21470 21584 21722 21748 21769 21774 21790 21806 2187
2 21919 21928 21942 21955 22037 22090 22113 22114 22152 22152 22156 22250 22298 22302 22344 22345 22385 22399 22454 2251
9 22521 22551 22641 22665 22718 22721 22726 22736 22757 22785 22821 22852 22901 22903 22913 22913 22946 22973 23007 2304
0 23113 23119 23124 23177 23262 23263 23266 23271 23304 23384 23450 23478 23491 23508 23538 23579 23631 23637 23641 2368
7 23747 23748 23764 23785 23791 23843 23844 23903 23907 23936 23969 23993 24047 24080 24133 24197 24217 24234 24317 2432
7 24410 24470 24526 24567 24571 24594 24613 24656 24677 24689 24730 24733 24739 24772 24790 24796 24848 24969 25024 2504
5 25081 25097 25102 25123 25184 25194 25225 25255 25297 25300 25320 25460 25554 25566 25572 25666 25694 25698 25735 2578
6 25816 25822 25845 25944 26009 26023 26042 26045 26054 26068 26088 26133 26166 26171 26218 26226 26244 26287 26387 2639
6 26491 26521 26527 26534 26601 26759 26866 26866 26886 27029 27065 27080 27089 27092 27152 27270 27336 27369 27370 2737
8 27430 27447 27571 27617 27773 27782 27811 27828 27858 27860 27861 27916 27937 27941 27954 27956 27999 28012 28042 2804
8 28065 28077 28085 28102 28110 28125 28178 28183 28206 28207 28218 28218 28291 28298 28354 28360 28399 28419 28438 2844
1 28464 28498 28513 28516 28562 28566 28591 28595 28597 28645 28646 28660 28683 28721 28768 28784 28804 28820 28828 2884
0 28842 28907 28958 29074 29118 29124 29146 29147 29396 29397 29543 29618 29644 29673 29703 29726 29756 29796 29837 2987
9 29883 29977 29985 29997 30012 30050 30135 30148 30159 30175 30189 30196 30222 30230 30257 30350 30361 30414 30518 3060
0 30658 30664 30684 30800 30827 30902 30907 30923 30930 30998 31015 31018 31044 31087 31090 31109 31175 31217 31220 3123
3 31239 31276 31337 31342 31360 31376 31377 31377 31530 31556 31559 31584 31605 31629 31667 31668 31682 31710 31774 3177
6 31778 31896 31914 31949 31953 31957 31995 32016 32046 32179 32296 32308 32371 32384 32487 32546 32559 32591 32610 3264
3 32714 32715
检测排序结果从1到1000是否有序:
检测结果: 排序正确，数列有序
分支遍历结果如下:
(l >= r) = True : √
(l >= r) = Flase : √
(A[j] <= x) = True : √
(A[j] <= x) = Flase : √
Congratulations! 通过了所有分支:)
```

N=10000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:2
N=10000
原始序列：31739 9085 16364 21871 10596 12601 2830 29765 13418 8395 13477 4049 5392 28734 20582 902 13644 11450 5540 1970
5 29103 157 1301 10901 3399 26572 7129 4752 7365 32332 14264 5661 11483 7436 4268 1252 10216 14597 27551 25775 16087 144
69 27253 4454 22852 13000 7501 6461 20705 19913 14596 10540 31769 11863 5883 7584 19443 11883 25115 2928 312 19897 5398
21778 15576 4229 11759 8207 28554 16568 15036 16052 27827 17715 9131 25098 9353 867 22083 10249 18808 32534 13804 23049
25797 11405 30130 28537 7793 10450 16198 10451 15456 19532 19269 22223 7354 19344 20157 8664 12178 23162 25070 26044 132
53 1405 17485 30119 15141 13623 8655 23634 14807 4168 15715 27768 9466 12133 9408 13706 25737 22440 20834 12451 7299 120
74 23247 14782 4425 8148 7711 10535 11079 23666 32443 8495 24022 15369 9317 25636 30049 24427 23161 18048 4295 21768 298
10 4449 18945 12039 23471 26937 20997 16278 10872 3271 1002 28542 30780 70 13132 25338 15132 30948 14834 24777 25262 728
5 2222 12162 4090 26985 16261 23848 3785 23291 4368 9470 29728 7596 29986 26524 1543 26608 28332 26947 20622 19239 26845
 19054 3090 4302 28557 28329 14525 30711 18139 17830 4940 26358 9749 10894 18197 25950 6617 19861 328 29228 20858 23505
18805 2625 23461 22860 17230 29462 5701 24920 15791 7595 21923 16674 13019 13760 1746 8330 7689 12616 20810 11570 18975
29723 31699 31701 32189 23531 6309 5083 10365 18553 15565 22826 30073 29704 18130 18494 26382 12306 30873 14460 24544 23
099 26758 6339 15646 7555 3556 23109 26120 21439 27783 32149 29870 6986 9135 30208 23546 29944 25648 22305 12016 32331 4
560 176 27957 7146 5105 15515 30926 10318 6512 856 6443 31423 21452 23161 12132 23167 28043 385 32706 12869 16345 24822
13441 6124 2931 28599 32212 31853 14531 656 29482 6155 25420 14860 6576 26317 13385 650 13335 18737 20625 23187 16279 74
55 23998 21355 16948 25372 25230 32742 20032 17344 1674 17789 24353 2410 29925 17313 29524 25138 26278 17744 1970 32287
```

（省略构造序列）

```
检测排序结果从1到10000是否有序:
检测结果: 排序正确，数列有序
分支遍历结果如下:
(l >= r) = True : √
(l >= r) = Flase : √
(A[j] <= x) = True : √
(A[j] <= x) = Flase : √
Congratulations! 通过了所有分支:)
```

N=100000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:2
N=100000
----------------------快速排序-----------------------
检测排序结果从1到100000是否有序:
检测结果：排序正确，数列有序
分支遍历结果如下:
(l >= r) = True : √
(l >= r) = Flase : √
(A[j] <= x) = True : √
(A[j] <= x) = Flase : √
Congratulations! 通过了所有分支:)
```

N=1000000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:2
N=1000000
----------------------快速排序-----------------------
检测排序结果从1到1000000是否有序:
检测结果：排序正确，数列有序
分支遍历结果如下:
(l >= r) = True : √
(l >= r) = Flase : √
(A[j] <= x) = True : √
(A[j] <= x) = Flase : √
Congratulations! 通过了所有分支:)
```

N=10000000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:0
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:2
N=10000000
----------------------快速排序-----------------------
检测排序结果从1到10000000是否有序:
检测结果：排序正确，数列有序
分支遍历结果如下:
(l >= r) = True : √
(l >= r) = Flase : √
(A[j] <= x) = True : √
(A[j] <= x) = Flase : √
Congratulations! 通过了所有分支:)
```

5.2 算法复杂性测试

5.2.1 插入排序

理论分析

首先，插入排序的空间复杂度为$O(1)$，只需一个元素的辅助空间，用于元素的位置交换。

最好情况

最好情况下，数组已经是有序的，每次考虑一个新的数据应该插入到什么位置时，只需要比较前一个元素即可得出结果，每次需要一次比较次数，每次将新数据插入到原有位置，需要一次腾挪次数。因此最好情况下，插入排序的$C_{min} = N - 1, M_{min} = N - 1$ 。所以最好情况下的时间复杂度为$O(N)$。

最坏情况

数组完全逆序，插入第2个元素时要比较前1个元素，插入第3个元素时，要比较前2个元素，.....，插入第N个元素，要比较前 N - 1 个元素。

因此，最坏情况下的比较次数是 $C_{max} = 1 + 2 + 3 + \ldots + (N - 1) = N(N - 1)/2$，而在每次向前比较的同时，还要将用于比较的有序数列元素向后腾挪一位，再加上最后找到正确位置后插入当前新元素，因此，最坏情况下的腾挪次数
$M_{max} = (1 + 1) + (2 + 1) + (3 + 1) + \ldots + (N - 1 + 1) = N(N + 1)/2 - 1$。所以最坏情况下的复杂度为 $O(N^2)$。

平均情况

在考虑第i个元素的插入位置时，最少比较1次，移动1次，最多比较$i - 1$次，移动i次，因此平均比较次数$C_{avg} = (N - 1 + N(N - 1)/2)/2 = (N^2 + N - 2)/4$，平均移动次数
$M_{avg} = (N - 1 + N(N + 1)/2)/2 = (N^2 + 7N - 8)/4$，平均复杂度为$O(N^2)$

运用以上公式，得到比较次数和腾挪次数的理论值：

N	最好情况（理论）	最坏情况（理论）	平均情况（理论）
10	$M = 9, C = 9$	$M = 54, C = 45$	$M = 40, C = 27$
100	$M = 99, C = 99$	$M = 5049, C = 4950$	$M = 2673, C = 2524$
1000	$M = 999, C = 999$	$M = 500499, C = 499500$	$M = 251748, C = 250249$
10000	$M = 9999, C = 9999$	$M = 50004999, C = 49995000$	$M = 2500174998 C = 2500024999$

实际测试

对于插入排序，要达到最好情况要构造升序数列，要达到最坏情况要构造降序数列，测试平均情况将构造10次随机数列，取平均值。

测试结果汇总表

N	最好情况	最坏情况	平均情况
10	$M = 9, C = 9$	$M = 54, C = 45$	$M = 31, C = 29$
100	$M = 99, C = 99$	$M = 5049, C = 4950$	$M = 2495, C = 2490$
1000	$M = 999, C = 999$	$M = 500499, C = 499500$	$M = 252037, C = 252030$
10000	$M = 9999, C = 9999$	$M = 50004999, C = 49995000$	$M = 24968711, C = 24968701$

可以看到，最好情况和最坏情况和理论分析完全一致，平均情况在误差允许的范围内和理论分析也一致。

详细测试截图如下所示：

N=10时测试结果

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:1
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:0
N=10
----------------------最好情况----------------------
构造最好情况下的原始序列:1 2 3 4 5 6 7 8 9 10

排序完毕，统计结果：腾挪次数：9 比较次数：9

----------------------最坏情况----------------------
构造最坏情况下的原始序列:10 9 8 7 6 5 4 3 2 1

排序完毕，统计结果：腾挪次数：54        比较次数：45

----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 ： 腾挪次数：37        比较次数：35
第2次测试 ： 腾挪次数：20        比较次数：18
第3次测试 ： 腾挪次数：35        比较次数：32
第4次测试 ： 腾挪次数：34        比较次数：33
第5次测试 ： 腾挪次数：32        比较次数：30
第6次测试 ： 腾挪次数：24        比较次数：24
第7次测试 ： 腾挪次数：32        比较次数：29
第8次测试 ： 腾挪次数：29        比较次数：27
第9次测试 ： 腾挪次数：34        比较次数：32
第10次测试 ： 腾挪次数：37       比较次数：35
平均腾挪次数：31        平均比较次数：29

----------------------总情况----------------------
对于插入排序，N = 10
最好情况:腾挪次数：9    比较次数：9
最坏情况:腾挪次数：54   比较次数：45
平均情况:腾挪次数：31   比较次数：29
```

N=100测试结果

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:1
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:0
N=100
----------------------最好情况----------------------
构造最好情况下的原始序列:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

排序完毕，统计结果：腾挪次数：99        比较次数：99

----------------------最坏情况----------------------
构造最坏情况下的原始序列:100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 6
9 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 2
9 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

排序完毕，统计结果：腾挪次数：5049      比较次数：4950

----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 ： 腾挪次数：2661      比较次数：2654
第2次测试 ： 腾挪次数：2562      比较次数：2558
第3次测试 ： 腾挪次数：2581      比较次数：2574
第4次测试 ： 腾挪次数：2769      比较次数：2765
第5次测试 ： 腾挪次数：2225      比较次数：2223
第6次测试 ： 腾挪次数：2266      比较次数：2263
第7次测试 ： 腾挪次数：2480      比较次数：2471
第8次测试 ： 腾挪次数：2476      比较次数：2470
第9次测试 ： 腾挪次数：2500      比较次数：2495
第10次测试 ： 腾挪次数：2437     比较次数：2431
平均腾挪次数：2495      平均比较次数：2490

----------------------总情况----------------------
对于插入排序，N = 100
最好情况:腾挪次数：99   比较次数：99
最坏情况:腾挪次数：5049 比较次数：4950
平均情况:腾挪次数：2495 比较次数：2490
```

N=1000测试结果

```
----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 : 腾挪次数：249532     比较次数：249527
第2次测试 : 腾挪次数：254179     比较次数：254173
第3次测试 : 腾挪次数：257215     比较次数：257205
第4次测试 : 腾挪次数：248150     比较次数：248141
第5次测试 : 腾挪次数：250830     比较次数：250826
第6次测试 : 腾挪次数：252282     比较次数：252274
第7次测试 : 腾挪次数：249692     比较次数：249683
第8次测试 : 腾挪次数：255819     比较次数：255814
第9次测试 : 腾挪次数：254186     比较次数：254181
第10次测试 : 腾挪次数：248489    比较次数：248482
平均腾挪次数：252037     平均比较次数：252030

----------------------总情况----------------------
对于插入排序, N = 1000
最好情况:腾挪次数：999   比较次数：999
最坏情况:腾挪次数：500499        比较次数：499500
平均情况:腾挪次数：252037        比较次数：252030
```

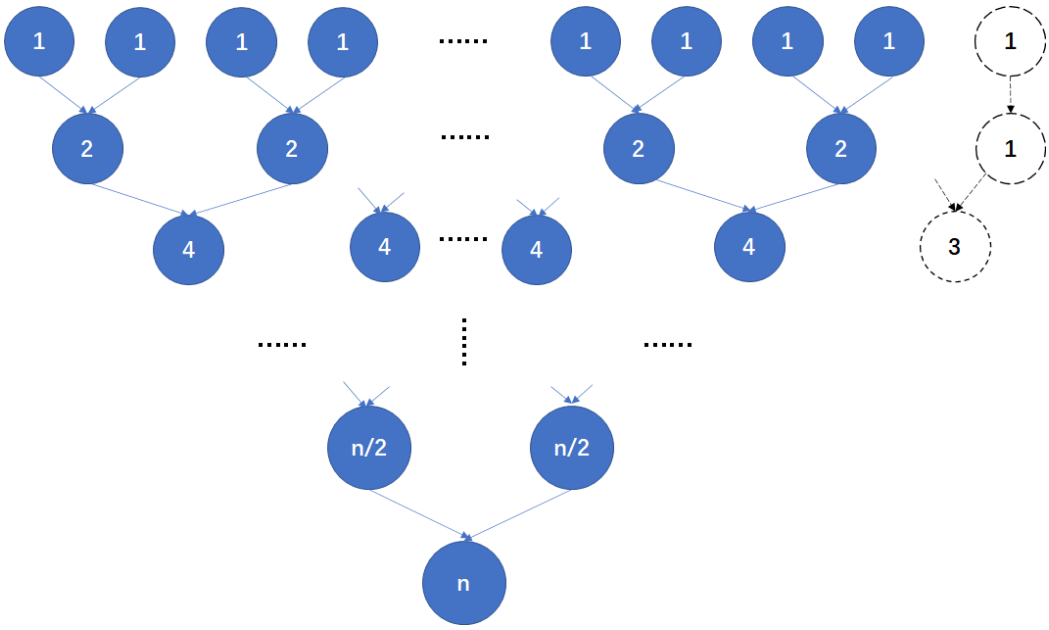
N=10000测试结果

```
----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 : 腾挪次数：25117097   比较次数：25117087
第2次测试 : 腾挪次数：24953138   比较次数：24953129
第3次测试 : 腾挪次数：25005753   比较次数：25005742
第4次测试 : 腾挪次数：25004409   比较次数：25004400
第5次测试 : 腾挪次数：24800454   比较次数：24800444
第6次测试 : 腾挪次数：24787264   比较次数：24787253
第7次测试 : 腾挪次数：25122047   比较次数：25122038
第8次测试 : 腾挪次数：25079736   比较次数：25079724
第9次测试 : 腾挪次数：24913182   比较次数：24913171
第10次测试 : 腾挪次数：24904038 比较次数：24904028
平均腾挪次数：24968711   平均比较次数：24968701

----------------------总情况----------------------
对于插入排序, N = 10000
最好情况:腾挪次数：9999 比较次数：9999
最坏情况:腾挪次数：50004999      比较次数：49995000
平均情况:腾挪次数：24968711      比较次数：24968701
```

5.2.2 合并排序

理论分析

合并排序的过程如下：树高为$NlogN$



腾挪次数

不论何种情况，在树的每一层上发生的腾挪次数一定为N，因为都需要将长度为N的序列搬运到另一个长度为N的数组上（且每个数据进搬运一次），因此$M_{min} = M_{max} = NlogN$

比较次数

在最好情况下，每次合并时左侧数列均小于右侧数列，只需要比较一半数列长度次数，剩下的直接拷贝，比较次数约为$N/2$，$C_{min} \approx N/2 * logN = 1/2NlogN$。最坏情况下，每次合并时左右两侧大小分布均等，在合并两个长度为c的序列时，需要比较的次数为$2c-1$，需要比较完整的数列长度次数，
$C_{max} \approx \sum_{i=0}^{logN-1}(N-2^i) = NlogN - (1+2^{logN-1}) * (logN)/2$

运用上述公式，得到理论值为

N	最好情况（理论）	最坏情况（理论）
10	$M=40, C=20$	$M=40, C=23$
100	$M=700, C=350$	$M=700, C=537$
1000	$M=10000, C=5000$	$M=10000, C=8489$
10000	$M=140000, C=70000$	$M=140000, C=121809$
100000	$M=1700000, C=850000$	$M=1700000, C=1534465$
1000000	$M=20000000, C=10000000$	$M=20000000, C=18475713$

实际测试

对于合并排序，要达到最好情况，只需要构造降序数列即可。

要达到最坏情况，需要构造满足下列条件的数列：对于合并排序中可能划分出的所有子序列，**左右子序列中的最大值与次大值必须出现在左子序列与右子序列的最右端**，这样才能使所有子序列的排序过程能够比较子序列中的所有数字。

构造方法如下：

首先生成`1,2,3,...,N`的有序数列，然后调用Separate函数，依据合并排序的顺序，从大到小遍历所有子序列，对于每个子序列，将原序列数据从大到小逆序分别存储在两个子序列中，其结果将最大值放在了子序列最右端。

```
void DataCreate::Separate()
{
    A = new int[N];
    int maxS = 1;
    while (maxS < N) maxS *= 2;      //先找到最大的子序列
    maxS /= 2;
    int s = maxS;
    while (s >= 1)                   //从最大的子序列开始依次遍历
    {
        anti_MergePass(s);
        s /= 2;
    }
    delete[] A;
}
```

```
void DataCreate::anti_MergePass(int s)
{
    int i = 0;
    while (i + 2 * s - 1 < N)        //和合并排序子序列遍历过程相同
    {
        anti_Merge(i, i + s - 1, i + 2 * s - 1);
        i = i + 2 * s;
    }
    if (i + s - 1 < N - 1)
    {
        anti_Merge(i, i + s - 1, N - 1);
    }

}

void DataCreate::anti_Merge(int l, int m, int r)
{
    int i = m, j = r, k = r;
    while ((i >= l) && (j >= m + 1))         //将原序列数据从大到小逆序
分别存储在两个子序列中
    {
        A[i--] = data[k--].Value;
        A[j--] = data[k--].Value;
    }

    if (i < l)                               //多余的数据同样需要存储
    {
        for (int q = j; q >= m + 1; q--)
            A[q] = data[k--].Value;
    }
    else
    {
        for (int q = i; q >= l; q--)
            A[q] = data[k--].Value;
    }
    for (int i = l; i <= r; ++i)
        data[i] = A[i];
}


/*
例：N=10
生成 10 4 6 2 8 3 5 1 9 7
*/
```

测试结果一览：

N	最好情况	最坏情况	平均情况
10	$M=40, C=15$	$M=40, C=27$	$M=40, C=23$
100	$M=700, C=316$	$M=700, C=589$	$M=700, C=556$

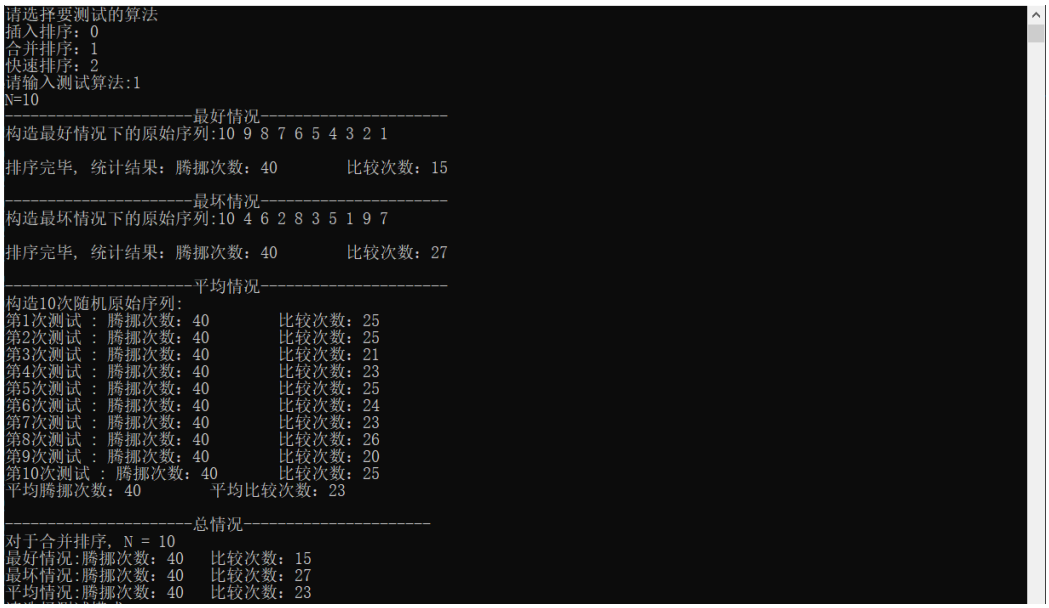
N	最好情况	最坏情况	平均情况
1000	$M = 10000, C = 4932$	$M = 10000, C = 8985$	$M = 10000 C = 8721$
10000	$M = 140000, C = 64608$	$M = 140000, C = 126321$	$M = 140000 C = 123661$
100000	$M = 1700000, C = 815024$	$M = 1700000, C = 1592993$	$M = 1700000 C = 1566558$
1000000	$M = 20000000, C = 9884992$	$M = 20000000, C = 18980545$	$M = 20000000$ $C = 18715951$

从结果可以看出，最好情况与最差情况的实际表现与理论分析十分接近，推测误差在于合并排序中在每次指定子序列长度合并时会剩下一个小序列进行特殊处理，由此导致一些细微的偏差。在误差允许的范围内，可以认为测试结果符合预期。

与此同时，在针对随机数列的测试中，平均情况的比较次数与最坏情况十分接近，可以认为合并排序是一个在平均条件下十分接近最坏情况的算法。

详细测试截图如下所示：

N=10



N=100

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:1
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:1
N=100
----------------------最好情况----------------------
构造最好情况下的原始序列:100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 6
9 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 2
9 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

排序完毕，统计结果：腾挪次数：700       比较次数：316

----------------------最坏情况----------------------
构造最坏情况下的原始序列:100 36 68 16 84 24 52 8 92 28 60 12 76 20 44 4 96 32 64 14 80 22 48 6 88 26 56 10 72 18 40 2 98
 34 66 15 82 23 50 7 90 27 58 11 74 19 42 3 94 30 62 13 78 21 46 5 86 25 54 9 70 17 38 1 99 59 75 43 83 51 67 35 91 55 7
1 39 79 47 63 31 95 57 73 41 81 49 65 33 87 53 69 37 77 45 61 29 97 89 93 85

排序完毕，统计结果：腾挪次数：700       比较次数：589

----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 ： 腾挪次数：700       比较次数：554
第2次测试 ： 腾挪次数：700       比较次数：556
第3次测试 ： 腾挪次数：700       比较次数：561
第4次测试 ： 腾挪次数：700       比较次数：547
第5次测试 ： 腾挪次数：700       比较次数：553
第6次测试 ： 腾挪次数：700       比较次数：565
第7次测试 ： 腾挪次数：700       比较次数：565
第8次测试 ： 腾挪次数：700       比较次数：550
第9次测试 ： 腾挪次数：700       比较次数：548
第10次测试 ： 腾挪次数：700      比较次数：563
平均腾挪次数：700       平均比较次数：556

----------------------总情况----------------------
对于合并排序，N = 100
最好情况:腾挪次数：700  比较次数：316
最坏情况:腾挪次数：700  比较次数：589
平均情况:腾挪次数：700  比较次数：556
```

N=1000

```
42 30 958 446 702 190 830 318 574 62 894 382 638 126 766 254 510 11 974 462 718 206 846 334 590 78 910 398 654 142 782 2
70 526 19 942 430 686 174 814 302 558 46 878 366 622 110 750 238 494 3 994 482 738 226 866 354 610 98 930 418 674 162 80
2 290 546 34 962 450 706 194 834 322 578 66 898 386 642 130 770 258 514 13 978 466 722 210 850 338 594 82 914 402 658 14
6 786 274 530 21 946 434 690 178 818 306 562 50 882 370 626 114 754 242 498 5 986 474 730 218 858 346 602 90 922 410 666
 154 794 282 538 26 954 442 698 186 826 314 570 58 890 378 634 122 762 250 506 9 970 458 714 202 842 330 586 74 906 394
650 138 778 266 522 17 938 426 682 170 810 298 554 42 874 362 618 106 746 234 490 1 999 487 743 231 871 359 615 103 935
423 679 167 807 295 551 55 967 455 711 199 839 327 583 71 903 391 647 135 775 263 519 39 983 471 727 215 855 343 599 87
919 407 663 151 791 279 535 47 951 439 695 183 823 311 567 63 887 375 631 119 759 247 503 31 991 479 735 223 863 351 607
 95 927 415 671 159 799 287 543 51 959 447 703 191 831 319 575 67 895 383 639 127 767 255 511 35 975 463 719 207 847 335
 591 79 911 399 655 143 783 271 527 43 943 431 687 175 815 303 559 59 879 367 623 111 751 239 495 27 995 483 739 227 867
 355 611 99 931 419 675 163 803 291 547 53 963 451 707 195 835 323 579 69 899 387 643 131 771 259 515 37 979 467 723 211
 851 339 595 83 915 403 659 147 787 275 531 45 947 435 691 179 819 307 563 61 883 371 627 115 755 243 499 29 987 475 731
 219 859 347 603 91 923 411 667 155 795 283 539 49 955 443 699 187 827 315 571 65 891 379 635 123 763 251 507 33 971 459
 715 203 843 331 587 75 907 395 651 139 779 267 523 41 939 427 683 171 811 299 555 57 875 363 619 107 747 235 491 25 997
 485 741 229 869 357 613 133 933 421 677 165 805 293 549 101 965 453 709 197 837 325 581 117 901 389 645 149 773 261 517
 85 981 469 725 213 853 341 597 125 917 405 661 157 789 277 533 93 949 437 693 181 821 309 565 109 885 373 629 141 757 2
45 501 77 989 477 733 221 861 349 605 129 925 413 669 161 797 285 541 97 957 445 701 189 829 317 573 113 893 381 637 145
 765 253 509 81 973 461 717 205 845 333 589 121 909 397 653 153 781 269 525 89 941 429 685 173 813 301 557 105 877 365 6
21 137 749 237 493 73 993 481 737 289 865 353 609 225 929 417 673 257 801 321 545 193 961 449 705 273 833 337 577 209 89
7 385 641 241 769 305 513 177 977 465 721 281 849 345 593 217 913 401 657 249 785 313 529 185 945 433 689 265 817 329 56
1 201 881 369 625 233 753 297 497 169 985 601 729 473 857 537 665 409 921 569 697 441 793 505 633 377 953 585 713 457 82
5 521 649 393 889 553 681 425 761 489 617 361 969 841 905 777 937 809 873 745

排序完毕，统计结果：腾挪次数：10000      比较次数：8985

----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 ： 腾挪次数：10000      比较次数：8712
第2次测试 ： 腾挪次数：10000      比较次数：8731
第3次测试 ： 腾挪次数：10000      比较次数：8722
第4次测试 ： 腾挪次数：10000      比较次数：8744
第5次测试 ： 腾挪次数：10000      比较次数：8741
第6次测试 ： 腾挪次数：10000      比较次数：8706
第7次测试 ： 腾挪次数：10000      比较次数：8707
第8次测试 ： 腾挪次数：10000      比较次数：8718
第9次测试 ： 腾挪次数：10000      比较次数：8734
第10次测试 ： 腾挪次数：10000     比较次数：8704
平均腾挪次数：10000      平均比较次数：8721

----------------------总情况----------------------
对于合并排序，N = 1000
最好情况:腾挪次数：10000         比较次数：4932
最坏情况:腾挪次数：10000         比较次数：8985
平均情况:腾挪次数：10000         比较次数：8721
```

N=10000

```
949 7901 8925 7349 9437 7605 8413 7093 9693 7733 8669 7221 9181 7477 8157 6965 9821 7797 8797 7285 9309 7541 8285 7029 9
565 7669 8541 7157 9053 7413 8029 6901 9885 7837 8861 7317 9373 7573 8349 7061 9629 7701 8605 7189 9117 7445 8093 6933 9
757 7765 8733 7253 9245 7509 8221 6997 9501 7637 8477 7125 8989 7381 7965 6869 9989 7941 8965 7369 9477 7625 8453 7113 9
733 7753 8709 7241 9221 7497 8197 6985 9861 7817 8837 7305 9349 7561 8325 7049 9605 7689 8581 7177 9093 7433 8069 6921 9
925 7877 8901 7337 9413 7593 8389 7081 9669 7721 8645 7209 9157 7465 8133 6953 9797 7785 8773 7273 9285 7529 8261 7017 9
541 7657 8517 7145 9029 7401 8005 6889 9957 7909 8933 7353 9445 7609 8421 7097 9701 7737 8677 7225 9189 7481 8165 6969 9
829 7801 8805 7289 9317 7545 8293 7033 9573 7673 8549 7161 9061 7417 8037 6905 9893 7845 8869 7321 9381 7577 8357 7065 9
637 7705 8613 7193 9125 7449 8101 6937 9765 7769 8741 7257 9253 7513 8229 7001 9509 7641 8485 7129 8997 7385 7973 6873 9
973 7925 8949 7361 9461 7617 8437 7105 9717 7745 8693 7233 9205 7489 8181 6977 9845 7809 8821 7297 9333 7553 8309 7041 9
589 7681 8565 7169 9077 7425 8053 6913 9909 7861 8885 7329 9397 7585 8373 7073 9653 7713 8629 7201 9141 7457 8117 6945 9
781 7777 8757 7265 9269 7521 8245 7009 9525 7649 8501 7137 9013 7393 7989 6881 9941 7893 8917 7345 9429 7601 8405 7089 9
685 7729 8661 7217 9173 7473 8149 6961 9813 7793 8789 7281 9301 7537 8277 7025 9557 7665 8533 7153 9045 7409 8021 6897 9
877 7829 8853 7313 9365 7569 8341 7057 9621 7697 8597 7185 9109 7441 8085 6929 9749 7761 8725 7249 9237 7505 8213 6993 9
493 7633 8469 7121 8981 7377 7957 6865 9993 8841 9353 8329 9609 8585 9097 8073 9737 8713 9225 8201 9481 8457 8969 7945 9
865 8777 9289 8265 9545 8521 9033 8009 9673 8649 9161 8137 9417 8393 8905 7881 9929 8809 9321 8297 9577 8553 9065 8041 9
705 8681 9193 8169 9449 8425 8937 7913 9801 8745 9257 8233 9513 8489 9001 7977 9641 8617 9129 8105 9385 8361 8873 7849 9
961 8825 9337 8313 9593 8569 9081 8057 9721 8697 9209 8185 9465 8441 8953 7929 9833 8761 9273 8249 9529 8505 9017 7993 9
657 8633 9145 8121 9401 8377 8889 7865 9897 8793 9305 8281 9561 8537 9049 8025 9689 8665 9177 8153 9433 8409 8921 7897 9
769 8729 9241 8217 9497 8473 8985 7961 9625 8601 9113 8089 9369 8345 8857 7833 9977 8833 9345 8321 9601 8577 9089 8065 9
729 8705 9217 8193 9473 8449 8961 7937 9849 8769 9281 8257 9537 8513 9025 8001 9665 8641 9153 8129 9409 8385 8897 7873 9
913 8801 9313 8289 9569 8545 9057 8033 9697 8673 9185 8161 9441 8417 8929 7905 9785 8737 9249 8225 9505 8481 8993 7969 9
633 8609 9121 8097 9377 8353 8865 7841 9945 8817 9329 8305 9585 8561 9073 8049 9713 8689 9201 8177 9457 8433 8945 7921 9
817 8753 9265 8241 9521 8497 9009 7985 9649 8625 9137 8113 9393 8369 8881 7857 9881 8785 9297 8273 9553 8529 9041 8017 9
681 8657 9169 8145 9425 8401 8913 7889 9753 8721 9233 8209 9489 8465 8977 7953 9617 8593 9105 8081 9361 8337 8849 7825 9
985 9857 9921 9793 9953 9825 9889 9761 9969 9841 9905 9777 9937 9809 9873 9745

排序完毕, 统计结果：腾挪次数：140000    比较次数：126321

----------平均情况----------
构造10次随机原始序列:
第1次测试 : 腾挪次数：140000    比较次数：123710
第2次测试 : 腾挪次数：140000    比较次数：123634
第3次测试 : 腾挪次数：140000    比较次数：123660
第4次测试 : 腾挪次数：140000    比较次数：123667
第5次测试 : 腾挪次数：140000    比较次数：123568
第6次测试 : 腾挪次数：140000    比较次数：123602
第7次测试 : 腾挪次数：140000    比较次数：123743
第8次测试 : 腾挪次数：140000    比较次数：123752
第9次测试 : 腾挪次数：140000    比较次数：123629
第10次测试 : 腾挪次数：140000    比较次数：123654
平均腾挪次数：140000    平均比较次数：123661

----------总情况----------
对于合并排序, N = 10000
最好情况:腾挪次数：140000        比较次数：64608
最坏情况:腾挪次数：140000        比较次数：126321
平均情况:腾挪次数：140000        比较次数：123661
```

N=100000

```
9 95433 98553 95945 96969 94921 99065 96201 97225 95177 98041 95689 96713 94665 99897 96617 97849 95593 98873 96105 9712
9 95081 99385 96361 97385 95337 98361 95849 96873 94825 99641 96489 97593 95465 98617 95977 97001 94953 99129 96233 9725
7 95209 98105 95721 96745 94697 99769 96553 97721 95529 98745 96041 97065 95017 99257 96297 97321 95273 98233 95785 9680
9 94761 99513 96425 97465 95401 98489 95913 96937 94889 99001 96169 97193 95145 97977 95657 96681 94633 99977 96657 9792
9 95633 98953 96145 97169 95121 99465 96401 97425 95377 98441 95889 96913 94865 99721 96529 97673 95505 98697 96017 9704
1 94993 99209 96273 97297 95249 98185 95761 96785 94737 99849 96593 97801 95569 98825 96081 97105 95057 99337 96337 9736
1 95313 98313 95825 96849 94801 99593 96465 97545 95441 98569 95953 96977 94929 99081 96209 97233 95185 98057 95697 9672
1 94673 99913 96625 97865 95601 98889 96113 97137 95089 99401 96369 97393 95345 98377 95857 96881 94833 99657 96497 9760
9 95473 98633 95985 97009 94961 99145 96241 97265 95217 98121 95729 96753 94705 99785 96561 97737 95537 98761 96049 9707
3 95025 99273 96305 97329 95281 98249 95793 96817 94769 99529 96433 97481 95409 98505 95921 96945 94897 99017 96177 9720
1 95153 97993 95665 96689 94641 99945 96641 97897 95617 98921 96129 97153 95105 99433 96385 97409 95361 98409 95873 9689
7 94849 99689 96513 97641 95489 98665 96001 97025 94977 99177 96257 97281 95233 98153 95745 96769 94721 99817 96577 9776
9 95553 98793 96065 97089 95041 99305 96321 97345 95297 98281 95809 96833 94785 99561 96449 97513 95425 98537 95937 9696
1 94913 99049 96193 97217 95169 98025 95681 96705 94657 99881 96609 97833 95585 98857 96097 97121 95073 99369 96353 9737
7 95329 98345 95841 96865 94817 99625 96481 97577 95457 98601 95969 96993 94945 99113 96225 97249 95201 98089 95713 9673
7 94689 99753 96545 97705 95521 98729 96033 97057 95009 99241 96289 97313 95265 98217 95777 96801 94753 99497 96417 9744
9 95393 98473 95905 96929 94881 98985 96161 97185 95137 97961 95649 96673 94625 99985 98449 98961 97937 99473 98193 9870
5 97681 99729 98321 98833 97809 99217 98065 98577 97553 99857 98385 98897 97873 99345 98129 98641 97617 99601 98257 9876
9 97745 99089 98001 98513 97489 99921 98417 98929 97905 99409 98161 98673 97649 99665 98289 98801 97777 99153 98033 9854
5 97521 99793 98353 98865 97841 99281 98097 98609 97585 99537 98225 98737 97713 99025 97969 98481 97457 99953 98433 9894
5 97921 99441 98177 98689 97665 99697 98305 98817 97793 99185 98049 98561 97537 99825 98369 98881 97857 99313 98113 9862
5 97601 99569 98241 98753 97729 99057 97985 98497 97473 99889 98401 98913 97889 99377 98145 98657 97633 99633 98273 9878
5 97761 99121 98017 98529 97505 99761 98337 98849 97825 99249 98081 98593 97569 99505 98209 98721 97697 98993 97953 9846
5 97441 99969 99457 99713 99201 99841 99329 99585 99073 99905 99393 99649 99137 99777 99265 99521 99009 99937 99425 9968
1 99169 99809 99297 99553 99041 99873 99361 99617 99105 99745 99233 99489 98977

排序完毕, 统计结果：腾挪次数：1700000    比较次数：1592993

----------平均情况----------
构造10次随机原始序列:
第1次测试 : 腾挪次数：1700000    比较次数：1566746
第2次测试 : 腾挪次数：1700000    比较次数：1566310
第3次测试 : 腾挪次数：1700000    比较次数：1566564
第4次测试 : 腾挪次数：1700000    比较次数：1566674
第5次测试 : 腾挪次数：1700000    比较次数：1566359
第6次测试 : 腾挪次数：1700000    比较次数：1566649
第7次测试 : 腾挪次数：1700000    比较次数：1566420
第8次测试 : 腾挪次数：1700000    比较次数：1566519
第9次测试 : 腾挪次数：1700000    比较次数：1566756
第10次测试 : 腾挪次数：1700000    比较次数：1566584
平均腾挪次数：1700000    平均比较次数：1566558

----------总情况----------
对于合并排序, N = 100000
最好情况:腾挪次数：1700000        比较次数：815024
最坏情况:腾挪次数：1700000        比较次数：1592993
平均情况:腾挪次数：1700000        比较次数：1566558
```

N=1000000

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:1
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:1
N=1000000
----------------------最好情况----------------------
构造最好情况下的原始序列:

排序完毕，统计结果：腾挪次数：20000000  比较次数：9884992

----------------------最坏情况----------------------
构造最坏情况下的原始序列:

排序完毕，统计结果：腾挪次数：20000000  比较次数：18980545

----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 ：腾挪次数：20000000  比较次数：18715822
第2次测试 ：腾挪次数：20000000  比较次数：18716384
第3次测试 ：腾挪次数：20000000  比较次数：18716289
第4次测试 ：腾挪次数：20000000  比较次数：18716661
第5次测试 ：腾挪次数：20000000  比较次数：18715013
第6次测试 ：腾挪次数：20000000  比较次数：18715123
第7次测试 ：腾挪次数：20000000  比较次数：18715475
第8次测试 ：腾挪次数：20000000  比较次数：18717034
第9次测试 ：腾挪次数：20000000  比较次数：18715833
第10次测试 ：腾挪次数：20000000 比较次数：18715876
平均腾挪次数：20000000  平均比较次数：18715951

----------------------总情况----------------------
对于合并排序，N = 1000000
最好情况:腾挪次数：20000000      比较次数：9884992
最坏情况:腾挪次数：20000000      比较次数：18980545
平均情况:腾挪次数：20000000      比较次数：18715951
```

5.2.3 快速排序

理论分析

在最好情况下，每次都划分得很均匀，如果排序n个关键字，其递归树的深度就为log_2N+1，即仅需递归log_2N次，需要时间为T（n）的话，第一次Partiation应该是需要对整个数组扫描一遍，做n次比较。然后，获得的枢轴将数组一分为二，那么各自还需要T（n/2）的时间（注意是最好情况，所以平分两半）。于是不断地划分下去，我们就有了下面的不等式推断。

$$
\begin{aligned}
T（n） &\leq 2T（n/2） + nT（1） = 0 \\
T（n） &\leq 2（2T（n/4） + n/2） + n = 4T（n/4） + 2n \\
T（n） &\leq 4（2T（n/8） + n/4） + 2n = 8T（n/8） + 3n \\
&\cdots \\
T（n） &\leq nT（1） + （log2n） \times n = O(nlogn)
\end{aligned}
$$

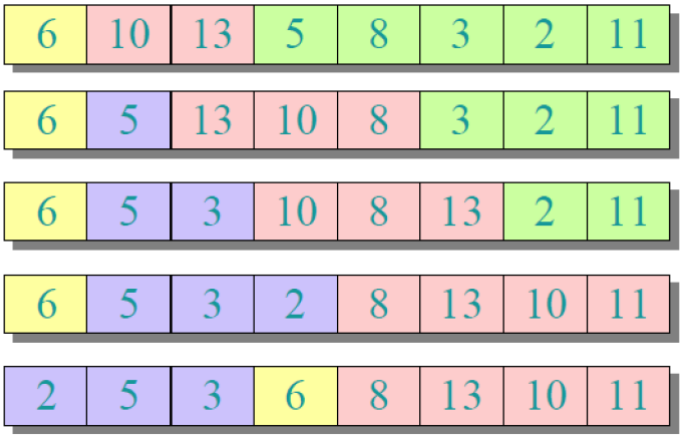
也就是说，在最优的情况下，快速排序算法的时间复杂度为O(nlogn)。

在最坏的情况下，待排序的序列为正序或者逆序，每次划分只得到一个比上一次划分少一个记录的子序列，注意另一个为空。如果递归树画出来，它就是一棵斜树。此时需要执行n-1次递归调用，且第i次划分需要经过n-i次关键字的比较才能找到第i个记录，也就是枢轴的位置，因此比较次数为$1+2+3+\ldots+N=N(N+1)/2$，复杂度为$O(N^2)$

实际测试

对于快速排序，要达到最好情况，需要构造满足下列条件的数列：**每个子序列选择的基准数是这段序列的中值**，构造方法如下：

首先生成1,2,3,...,N的有序数列，然后进行Adjust调整操作：先将序列排序，然后取中间的那个值，将它和最左端的值交换（我们的快速排序算法选择序列最左端的值作为基准值）。 然后将原序列分为两个大小相同的子序列。不过要注意：



左边的子序列，在快速排序的过程中，原来在最右边的数字被换到了最左边。因此，我们调整数列的过程中，递归进入子序列时，还要传入belong参数告诉子序列它是左子序列还是右子序列，如果是左子序列应当把中值放到最右端，如果是右子序列应当把中值放到最左端。

```
void DataCreate::Adjust(int l, int r, int belong)
{
    if (l >= r)
        return;
    int base = (l + r) / 2;
    if (belong == 0)                //判断当前是左子序列还是右子序列
    {
        Swap(data[r], data[base]);  //如果是左子序列应当把中值放到最右端
        if (l < r)
            sort(data + l, data + r - 1);
        Adjust(l, base - 1, 0);
        Adjust(base, r - 1, 1);
    }
    else
    {
        Swap(data[l], data[base]);  //如果是右子序列应当把中值放到最左端
        if (l < r)
            sort(data + l + 1, data + r);
        Adjust(l + 1, base, 0);
        Adjust(base + 1, r, 1);
    }
}
/*
如：N=10
5 1 4 3 2 8 7 6 9 10
*/
```

要达到最坏情况，构造升序数列即可。

测试结果一览：

N	最好情况	最坏情况	平均情况
10	$M=12,C=26$	$M=18,C=55$	$M=22,C=34$
100	$M=134,C=552$	$M=198,C=5050$	$M=647,C=802$
1000	$M=1276,C=8753$	$M=1998,C=500500$	$M=12004C=12790$
10000	$M=11808,C=119536$	$M=INF,C=INF$	$M=152739C=174981$
100000	$M=134462,C=1537874$	$M=INF,C=INF$	$M=1933120C=2198957$
1000000	$M=1310716,C=18737875$	$M=INF,C=INF$	$M=20721713$ $C=36802590$

最好情况与平均情况均符合$O(NlogN)$的复杂度，最坏情况符合$O(N^2)$的复杂度。在测试过程中我发现，在N=10000即之后的测试过程中，在最坏情况下出现了爆栈的情况，原因在于在最坏情况下，序列长度为N会进入N层递归调用，过多的递归调用容易导致栈溢出。

N=10测试结果

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:1
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:2
N=10
----------------------最好情况----------------------
构造最好情况下的原始序列:5 1 4 3 2 8 7 6 9 10

排序完毕，统计结果：腾挪次数：12        比较次数：26

----------------------最坏情况----------------------
构造最坏情况下的原始序列:1 2 3 4 5 6 7 8 9 10

排序完毕，统计结果：腾挪次数：18        比较次数：55

----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 : 腾挪次数：20        比较次数：31
第2次测试 : 腾挪次数：14        比较次数：36
第3次测试 : 腾挪次数：26        比较次数：30
第4次测试 : 腾挪次数：20        比较次数：34
第5次测试 : 腾挪次数：22        比较次数：40
第6次测试 : 腾挪次数：20        比较次数：37
第7次测试 : 腾挪次数：42        比较次数：31
第8次测试 : 腾挪次数：24        比较次数：32
第9次测试 : 腾挪次数：20        比较次数：37
第10次测试 : 腾挪次数：16       比较次数：35
平均腾挪次数：22        平均比较次数：34

----------------------总情况----------------------
对于快速排序, N = 10
最好情况:腾挪次数：12   比较次数：26
最坏情况:腾挪次数：18   比较次数：55
平均情况:腾挪次数：22   比较次数：34
```

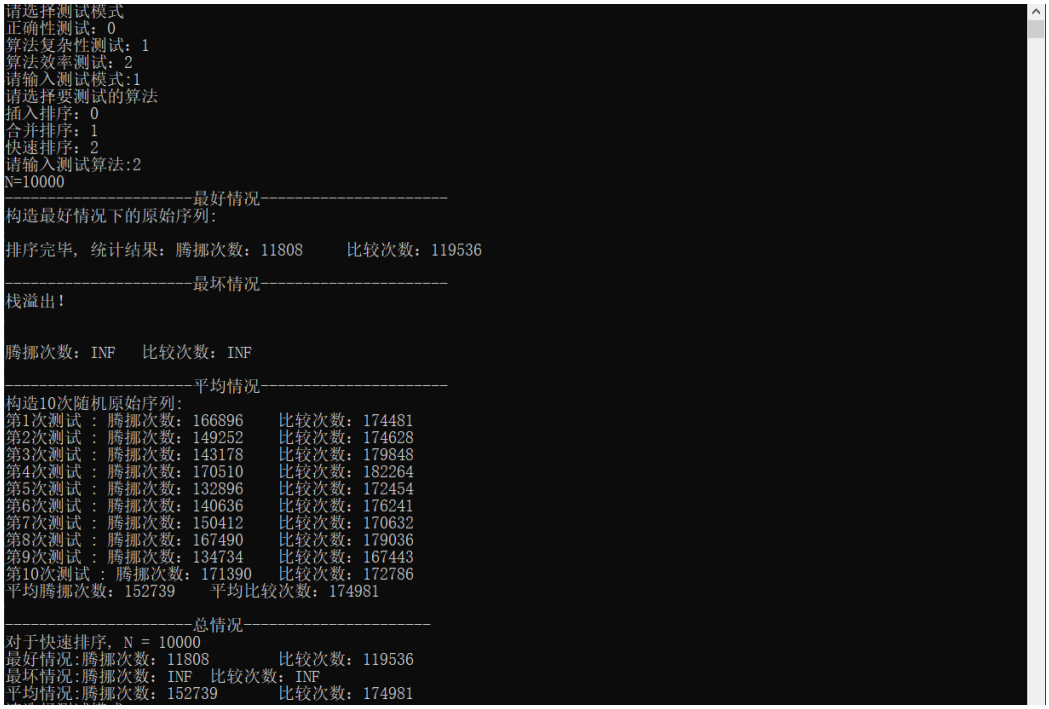
N=100测试结果

```
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:1
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:2
N=100
----------------------最好情况----------------------
构造最好情况下的原始序列:50 2 1 5 4 3 9 8 7 11 10 6 18 14 13 17 16 15 21 20 19 24 22 23 12 37 27 26 30 29 28 34 33 32 36
 35 31 43 39 38 42 41 40 46 45 44 49 47 48 25 75 52 51 55 54 53 59 58 57 61 60 56 68 64 63 67 66 65 71 70 69 74 72 73 62
 88 77 76 80 79 78 84 83 82 87 85 86 81 94 90 89 93 92 91 97 96 95 99 98 100

排序完毕, 统计结果：腾挪次数：134       比较次数：552

----------------------最坏情况----------------------
构造最坏情况下的原始序列:1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75
 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

排序完毕, 统计结果：腾挪次数：198       比较次数：5050

----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 : 腾挪次数：706       比较次数：744
第2次测试 : 腾挪次数：778       比较次数：904
第3次测试 : 腾挪次数：656       比较次数：848
第4次测试 : 腾挪次数：532       比较次数：889
第5次测试 : 腾挪次数：716       比较次数：887
第6次测试 : 腾挪次数：576       比较次数：813
第7次测试 : 腾挪次数：664       比较次数：754
第8次测试 : 腾挪次数：640       比较次数：717
第9次测试 : 腾挪次数：582       比较次数：695
第10次测试 : 腾挪次数：620      比较次数：771
平均腾挪次数：647       平均比较次数：802

----------------------总情况----------------------
对于快速排序, N = 100
最好情况:腾挪次数：134  比较次数：552
最坏情况:腾挪次数：198  比较次数：5050
平均情况:腾挪次数：647  比较次数：802
```

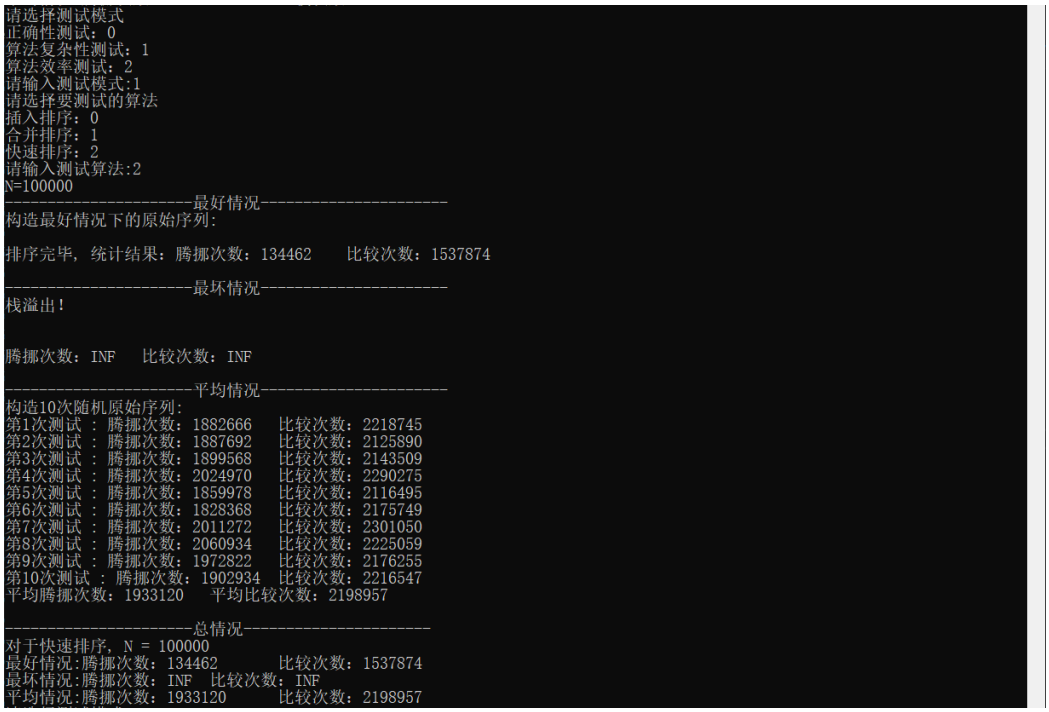
N=1000测试结果

```
 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291
 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321
 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351
 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381
 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411
 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441
 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471
 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501
 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531
 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561
 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591
 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621
 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651
 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681
 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711
 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741
 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771
 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801
 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831
 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861
 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891
 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921
 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951
 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981
 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

排序完毕, 统计结果：腾挪次数：1998       比较次数：500500

----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 : 腾挪次数：11514      比较次数：12350
第2次测试 : 腾挪次数：12064      比较次数：12441
第3次测试 : 腾挪次数：11996      比较次数：13271
第4次测试 : 腾挪次数：11810      比较次数：12694
第5次测试 : 腾挪次数：11002      比较次数：12807
第6次测试 : 腾挪次数：10884      比较次数：12762
第7次测试 : 腾挪次数：12188      比较次数：12987
第8次测试 : 腾挪次数：12218      比较次数：11733
第9次测试 : 腾挪次数：13592      比较次数：13190
第10次测试 : 腾挪次数：12780     比较次数：13668
平均腾挪次数：12004      平均比较次数：12790

----------------------总情况----------------------
对于快速排序, N = 1000
最好情况:腾挪次数：1276 比较次数：8753
最坏情况:腾挪次数：1998 比较次数：500500
平均情况:腾挪次数：12004         比较次数：12790
```

N=10000测试结果



N=100000 测试结果



N=1000000 测试结果

```
请选择测试模式
正确性测试：0
算法复杂性测试：1
算法效率测试：2
请输入测试模式:1
请选择要测试的算法
插入排序：0
合并排序：1
快速排序：2
请输入测试算法:2
N=1000000
----------------------最好情况----------------------
构造最好情况下的原始序列:

排序完毕，统计结果：腾挪次数：1310716    比较次数：18737875

----------------------最坏情况----------------------
栈溢出!


腾挪次数：INF    比较次数：INF

----------------------平均情况----------------------
构造10次随机原始序列:
第1次测试 ： 腾挪次数：21548848   比较次数：38185184
第2次测试 ： 腾挪次数：20053220   比较次数：36602300
第3次测试 ： 腾挪次数：20408548   比较次数：36571306
第4次测试 ： 腾挪次数：20101556   比较次数：36479141
第5次测试 ： 腾挪次数：21182142   比较次数：36330086
第6次测试 ： 腾挪次数：19917246   比较次数：35927364
第7次测试 ： 腾挪次数：20667152   比较次数：36386881
第8次测试 ： 腾挪次数：21263706   比较次数：36663408
第9次测试 ： 腾挪次数：21944994   比较次数：38784421
第10次测试 ： 腾挪次数：20129720 比较次数：36095815
平均腾挪次数：20721713   平均比较次数：36802590

----------------------总情况----------------------
对于快速排序，N = 1000000
最好情况:腾挪次数：1310716       比较次数：18737875
最坏情况:腾挪次数：INF   比较次数：INF
平均情况:腾挪次数：20721713      比较次数：36802590
```

5.3 算法效率测试

对于每个N，生成10组随机数据，比较腾挪次数、比较次数和程序运行时间

N=10

```
N=10
构造10次随机原始序列:
-------------------------第1次测试-------------------------
原始序列:
4853 13215 542 8604 3296 28104 9002 31272 20848 12737
插入排序：腾挪次数：23   比较次数：22   运行时间0
合并排序：腾挪次数：40   比较次数：25运行时间0
快速排序：腾挪次数：22   比较次数：33运行时间0
-------------------------第2次测试-------------------------
原始序列:
4856 23963 18406 32668 13611 32030 12114 28344 4592 6589
插入排序：腾挪次数：36   比较次数：35   运行时间0
合并排序：腾挪次数：40   比较次数：21运行时间0
快速排序：腾挪次数：22   比较次数：33运行时间0
-------------------------第3次测试-------------------------
原始序列:
4860 1944 3502 23963 23925 3189 15227 25417 21103 441
插入排序：腾挪次数：30   比较次数：28   运行时间0
合并排序：腾挪次数：40   比较次数：25运行时间0
快速排序：腾挪次数：22   比较次数：30运行时间0
-------------------------第4次测试-------------------------
原始序列:
4863 12692 21366 15258 1472 7116 18339 22490 4847 27060
插入排序：腾挪次数：25   比较次数：24   运行时间0
合并排序：腾挪次数：40   比较次数：25运行时间0
```

```
快速排序：腾挪次数：16  比较次数：29运行时间0
-------------------------第5次测试-------------------------
原始序列:
4866 23441 6463 6554 11786 11043 21452 19563 21359 20912
插入排序：腾挪次数：22  比较次数：22  运行时间0
合并排序：腾挪次数：40  比较次数：25运行时间0
快速排序：腾挪次数：16  比较次数：41运行时间0
-------------------------第6次测试-------------------------
原始序列:
4870 1421 24327 30617 22101 14969 24564 16635 5102 14764
插入排序：腾挪次数：32  比较次数：31  运行时间0
合并排序：腾挪次数：40  比较次数：21运行时间0
快速排序：腾挪次数：24  比较次数：31运行时间0
-------------------------第7次测试-------------------------
原始序列:
4873 12170 9423 21912 32415 18896 27677 13708 21614 8616
插入排序：腾挪次数：28  比较次数：28  运行时间0
合并排序：腾挪次数：40  比较次数：24运行时间0
快速排序：腾挪次数：20  比较次数：38运行时间0
-------------------------第8次测试-------------------------
原始序列:
4876 22918 27287 13208 9962 22823 30789 10781 5357 2468
插入排序：腾挪次数：37  比较次数：36  运行时间0
合并排序：腾挪次数：40  比较次数：21运行时间0
快速排序：腾挪次数：22  比较次数：36运行时间0
-------------------------第9次测试-------------------------
原始序列:
4879 898 12383 4503 20277 26749 1133 7853 21869 29088
插入排序：腾挪次数：21  比较次数：20  运行时间0
合并排序：腾挪次数：40  比较次数：25运行时间0
快速排序：腾挪次数：18  比较次数：25运行时间0
-------------------------第10次测试-------------------------
原始序列:
4883 11647 30247 28566 30591 30676 4246 4926 5613 22940
插入排序：腾挪次数：30  比较次数：29  运行时间0
合并排序：腾挪次数：40  比较次数：21运行时间0
快速排序：腾挪次数：28  比较次数：29运行时间0
----------------------总情况----------------------
插入排序：平均腾挪次数：28  平均比较次数：27平均运行时间0
合并排序：平均腾挪次数：40  平均比较次数：23平均运行时间0
快速排序：平均腾挪次数：21  平均比较次数：32平均运行时间0
```

N=100

```
N=100
构造10次随机原始序列:
-------------------------第1次测试-------------------------
原始序列:
```

```
5010 4851 6054 16763 6876 19979 27328 21834 26973 12541 4550 16092
9723 20329 30957 1123 7598 25526 19727 19237 7478 4602 17147 20863
4873 21017 26896 18986 3472 17712 14090 31933 27026 31284 20507
20339 17649 26070 11070 7727 2452 4522 5576 701 9380 18057 17471
29275 15867 13876 25968 21624 985 26681 27580 30395 24281 28187 179
583 8740 8877 22244 22078 3045 30911 19954 27523 10749 7420 7403
9669 30455 28435 30422 13753 17214 5653 23947 28838 16013 13342
5832 16679 13736 32056 30592 5421 22602 13692 31094 27971 7006
23163 20235 28950 2929 31708 26964 11572
插入排序：腾挪次数：2285  比较次数：2280  运行时间0
合并排序：腾挪次数：700  比较次数：563运行时间0
快速排序：腾挪次数：498  比较次数：667运行时间0
-------------------------第2次测试-------------------------
原始序列：
5013 15599 23918 8059 17190 23905 30440 18907 10716 6393 4402 16136
31862 4108 20381 18461 1648 11184 21557 9994 18893 20082 32162
27878 19476 32076 15436 2694 13465 5329 17091 20302 8189 24235 5469
19015 19704 6983 15649 16754 1622 21455 8675 13148 28224 2137 20401
5875 3773 3678 11118 19864 21369 13165 23229 31605 20676 30752
25377 14865 31709 30544 17506 22817 14449 19845 4658 8610 24415
5296 20752 20260 8894 18461 6765 13161 29473 9379 19469 14382 18969
22814 18097 13129 32252 6943 32755 16628 741 29665 27505 6249 15612
22911 32557 3759 18424 26725 22324 15283
插入排序：腾挪次数：2381  比较次数：2376  运行时间0
合并排序：腾挪次数：700  比较次数：556运行时间0
快速排序：腾挪次数：578  比较次数：721运行时间0
-------------------------第3次测试-------------------------
原始序列：
5016 26347 9014 32122 27505 27832 784 15980 27228 245 4254 16180
21233 20654 9806 3031 28466 29609 23388 750 30309 2795 14409 2124
1311 10366 3977 19169 23458 25714 20092 8670 22120 17186 23199
17690 21760 20665 20228 25781 793 5620 11775 25595 14301 18985
23331 15243 24448 26248 29036 18104 8984 32417 18878 47 17071 550
17806 29146 21910 19442 12767 23557 25853 8779 22129 22464 5313
3172 1333 30852 20101 8486 15875 12569 8964 13104 14991 32694 21925
32285 30362 9579 18001 14597 2149 27836 11649 12870 23915 17295
24219 22660 12111 11336 1150 21742 17684 18994
插入排序：腾挪次数：2557  比较次数：2554  运行时间0
合并排序：腾挪次数：700  比较次数：557运行时间0
快速排序：腾挪次数：624  比较次数：708运行时间0
-------------------------第4次测试-------------------------
原始序列：
5020 4328 26879 23417 5051 31759 3897 13053 10971 26865 4106 16224
10603 4432 31998 20369 22516 15266 25219 24275 8956 18276 29424
9139 15914 21425 25285 2877 683 13331 23092 29807 3283 10137 8161
16366 23816 1578 24807 2040 32732 22552 14875 5274 377 3064 26260
24611 12354 16050 14187 16344 29367 18900 14527 1257 13465 3116
10236 10660 12111 8341 8029 24297 4489 30481 6832 3550 18979 1048
14682 8675 31308 31280 24986 11977 21223 16830 10514 18238 24881
8988 9859 6028 3749 22251 4312 6276 22557 28843 20326 28340 57
22409 24434 18913 16644 16759 13043 22705
插入排序：腾挪次数：2552  比较次数：2546  运行时间0
```

```
合并排序：腾挪次数：700  比较次数：561运行时间0
快速排序：腾挪次数：476  比较次数：781运行时间0
-------------------------第5次测试-------------------------
原始序列:
5023 15076 11975 14713 15366 2918 7009 10125 27483 20717 3958 16268
32742 20979 21422 4939 16566 923 27050 15032 20372 989 11670 16153
30517 32484 13826 19353 10676 948 26093 18176 17215 3088 25891
15042 25872 15259 29386 11067 31902 6717 17975 17721 19221 19912
29190 1211 260 5852 32105 14584 16982 5384 10176 2467 9860 5681
2666 24942 2313 30008 3291 25037 15893 19415 24303 17404 32645
31692 28030 19267 9747 21306 1329 11385 714 20556 6036 3783 27838
18459 22124 2478 22265 29906 6474 17483 696 12048 16736 6618 8664
22157 3988 26490 32139 11777 8403 26415
插入排序：腾挪次数：2530  比较次数：2527  运行时间0
合并排序：腾挪次数：700  比较次数：566运行时间0.997
快速排序：腾挪次数：528  比较次数：692运行时间0
-------------------------第6次测试-------------------------
原始序列:
5026 25825 29839 6008 25680 6844 10122 7198 11227 14569 3810 16312
22113 4757 10846 22277 10616 19349 28880 5788 31787 16470 26685
23168 12352 10775 2366 3060 20669 21333 29094 6545 31146 28807
10853 13717 27927 28940 1197 20094 31073 23650 21075 30168 5298
3992 32119 10579 20935 28422 17255 12824 4598 24635 5825 3677 6255
8247 27864 6455 25282 18907 31321 25776 27297 8349 9007 31258 13543
29568 8611 29859 20954 11332 10440 10793 12973 24281 1558 22095
30794 27930 1621 31696 8014 4792 8637 28691 11604 28020 13147 17664
17270 21906 16310 1299 14865 6794 3763 30126
插入排序：腾挪次数：2516  比较次数：2513  运行时间0
合并排序：腾挪次数：700  比较次数：564运行时间0
快速排序：腾挪次数：574  比较次数：695运行时间0
-------------------------第7次测试-------------------------
原始序列:
5030 3805 14935 30071 3227 10771 13234 4271 27738 8421 3662 16356
11484 21303 270 6847 4666 5006 30711 29313 10435 31951 8932 30183
26955 21834 23675 19536 30661 8950 32095 27681 12309 21758 28583
12393 29983 9853 5776 29121 30244 7814 24174 9847 24142 20840 2281
19947 8841 18224 2405 11064 24981 11119 1474 4887 2649 10813 20294
20737 15483 7805 26583 26516 5933 30051 26478 12344 27210 27444
21960 7682 32161 1357 19551 10201 25233 28007 29848 7639 982 4633
13886 28145 26530 12446 10799 7131 22511 11225 9557 28710 25876
21654 28632 8876 30359 1811 31890 1069
插入排序：腾挪次数：2453  比较次数：2450  运行时间0
合并排序：腾挪次数：700  比较次数：559运行时间0
快速排序：腾挪次数：686  比较次数：759运行时间0
-------------------------第8次测试-------------------------
原始序列:
```

```
5033 14553 31 21367 13542 14698 16347 1344 11482 2273 3514 16400
855 5082 22462 24186 31484 23431 32542 20070 21850 14664 23947 4429
8790 124 12215 3244 7886 29336 2328 16050 26240 14710 13546 11068
32039 23535 10355 5380 29414 24747 27274 22293 10218 4920 5210
29315 29516 8026 20324 9304 12596 30370 29891 6097 31812 13378
12724 2251 5684 29472 21845 27256 17337 18985 11181 26198 8108
25320 2541 18274 10600 24151 28661 9609 4724 31733 25370 25951 3938
14104 26151 24595 12278 20101 12961 18339 651 27198 5968 6988 1715
21403 8186 16453 13086 29596 27250 4780
插入排序: 腾挪次数：2399  比较次数：2398  运行时间0
合并排序: 腾挪次数：700  比较次数：571运行时间0
快速排序: 腾挪次数：530  比较次数：770运行时间0
-------------------------第9次测试-------------------------
原始序列:
5036 25302 17895 12662 23856 18624 19459 31184 27993 28893 3367
16444 22994 21628 11887 8756 25534 9088 1604 10826 498 30145 6194
11444 23393 11183 756 19719 17879 16953 5329 4419 7403 7661 31276
9744 1327 4448 14934 14407 28585 8912 30374 1972 29063 21768 8140
5915 17422 30596 5474 7544 212 16854 25540 7307 28207 15944 5154
16532 28653 18371 17107 27996 28741 7919 28652 7284 21774 23196
15890 28865 21807 14177 5004 9017 16983 2690 20892 11495 6894 23575
5648 21045 30795 27755 15124 29546 11559 10403 2378 18034 10321
21152 20509 24030 28580 24614 22610 8491
插入排序: 腾挪次数：2449  比较次数：2445  运行时间0
合并排序: 腾挪次数：700  比较次数：562运行时间0
快速排序: 腾挪次数：502  比较次数：721运行时间0
-------------------------第10次测试-------------------------
原始序列:
5039 3282 2992 3957 1403 22551 22572 28257 11737 22745 3219 16489
12365 5406 1311 26094 19584 27513 3435 1583 11913 12858 21209 18458
5228 22242 22065 3427 27872 4570 8330 25556 21334 612 16238 8420
3382 18129 19513 23434 27756 25845 706 14419 15139 5848 11069 15283
5328 20398 23392 5784 20595 3337 21189 8517 24602 18510 30352 30814
18854 7269 12369 28735 7377 29621 13356 21138 2672 21072 29239 6689
246 4202 14115 8424 29242 6416 16415 29807 9850 279 17913 17495
16543 2642 17286 7986 22466 26376 31557 29080 18928 20900 63 31607
11306 19631 17969 12202
插入排序: 腾挪次数：2298  比较次数：2291  运行时间0
合并排序: 腾挪次数：700  比较次数：545运行时间0
快速排序: 腾挪次数：482  比较次数：715运行时间0
----------------------总情况----------------------
插入排序：平均腾挪次数：2442  平均比较次数：2438平均运行时间0
合并排序：平均腾挪次数：700  平均比较次数：560平均运行时间0.0997
快速排序：平均腾挪次数：547  平均比较次数：722平均运行时间0
```

N=1000（以下省略原始数列）：

```
N=1000
构造10次随机原始序列:
-------------------------第1次测试-------------------------
插入排序: 腾挪次数：252431  比较次数：252427  运行时间7.492
```

```
合并排序: 腾挪次数：10000  比较次数：8742运行时间0.994
快速排序: 腾挪次数：9258  比较次数：11289运行时间0.998
------------------------第2次测试------------------------
插入排序: 腾挪次数：257816  比较次数：257811  运行时间7.022
合并排序: 腾挪次数：10000  比较次数：8718运行时间0.997
快速排序: 腾挪次数：9122  比较次数：11022运行时间0
------------------------第3次测试------------------------
插入排序: 腾挪次数：248552  比较次数：248546  运行时间6.981
合并排序: 腾挪次数：10000  比较次数：8718运行时间0
快速排序: 腾挪次数：7964  比较次数：11736运行时间0
------------------------第4次测试------------------------
插入排序: 腾挪次数：251243  比较次数：251239  运行时间7.981
合并排序: 腾挪次数：10000  比较次数：8735运行时间0
快速排序: 腾挪次数：8880  比较次数：11184运行时间0
------------------------第5次测试------------------------
插入排序: 腾挪次数：250849  比较次数：250847  运行时间6.981
合并排序: 腾挪次数：10000  比较次数：8722运行时间0.998
快速排序: 腾挪次数：10238  比较次数：11802运行时间0.997
------------------------第6次测试------------------------
插入排序: 腾挪次数：248661  比较次数：248655  运行时间6.981
合并排序: 腾挪次数：10000  比较次数：8713运行时间0.997
快速排序: 腾挪次数：9470  比较次数：11550运行时间0.997
------------------------第7次测试------------------------
插入排序: 腾挪次数：252469  比较次数：252463  运行时间7.979
合并排序: 腾挪次数：10000  比较次数：8698运行时间0.997
快速排序: 腾挪次数：9768  比较次数：11482运行时间0.998
------------------------第8次测试------------------------
插入排序: 腾挪次数：248299  比较次数：248294  运行时间7.978
合并排序: 腾挪次数：10000  比较次数：8705运行时间0.997
快速排序: 腾挪次数：11004  比较次数：12275运行时间0.997
------------------------第9次测试------------------------
插入排序: 腾挪次数：251874  比较次数：251869  运行时间6.981
合并排序: 腾挪次数：10000  比较次数：8723运行时间0
快速排序: 腾挪次数：10026  比较次数：11627运行时间0
------------------------第10次测试------------------------
插入排序: 腾挪次数：242498  比较次数：242495  运行时间6.967
合并排序: 腾挪次数：10000  比较次数：8706运行时间0.997
快速排序: 腾挪次数：8230  比较次数：11899运行时间0.997
----------------------总情况----------------------
插入排序：平均腾挪次数：250469  平均比较次数：250464平均运行时间7.334
合并排序：平均腾挪次数：10000  平均比较次数：8718平均运行时间0.699
快速排序：平均腾挪次数：9396  平均比较次数：11586平均运行时间0.598
```

N=10000

```
N=10000
构造10次随机原始序列:
------------------------第1次测试------------------------
插入排序: 腾挪次数：24946667  比较次数：24946661  运行时间709.657
```

```
合并排序: 腾挪次数：140000  比较次数：123640运行时间7.98
快速排序: 腾挪次数：154586  比较次数：168782运行时间8.976
------------------------第2次测试------------------------
插入排序: 腾挪次数：25097050  比较次数：25097033  运行时间702.314
合并排序: 腾挪次数：140000  比较次数：123700运行时间8.976
快速排序: 腾挪次数：157550  比较次数：159472运行时间8.976
------------------------第3次测试------------------------
插入排序: 腾挪次数：24905129  比较次数：24905125  运行时间691.478
合并排序: 腾挪次数：140000  比较次数：123729运行时间8.976
快速排序: 腾挪次数：126364  比较次数：155858运行时间8.005
------------------------第4次测试------------------------
插入排序: 腾挪次数：25189502  比较次数：25189496  运行时间711.055
合并排序: 腾挪次数：140000  比较次数：123760运行时间7.981
快速排序: 腾挪次数：139846  比较次数：156134运行时间8.498
------------------------第5次测试------------------------
插入排序: 腾挪次数：24870109  比较次数：24870103  运行时间694.171
合并排序: 腾挪次数：140000  比较次数：123669运行时间8.892
快速排序: 腾挪次数：157668  比较次数：164789运行时间8.976
------------------------第6次测试------------------------
插入排序: 腾挪次数：25045213  比较次数：25045204  运行时间698.926
合并排序: 腾挪次数：140000  比较次数：123673运行时间8.982
快速排序: 腾挪次数：150950  比较次数：167140运行时间7.978
------------------------第7次测试------------------------
插入排序: 腾挪次数：24936534  比较次数：24936528  运行时间687.138
合并排序: 腾挪次数：140000  比较次数：123640运行时间8.975
快速排序: 腾挪次数：136698  比较次数：159222运行时间8.976
------------------------第8次测试------------------------
插入排序: 腾挪次数：24934668  比较次数：24934657  运行时间695.193
合并排序: 腾挪次数：140000  比较次数：123666运行时间7.978
快速排序: 腾挪次数：135174  比较次数：163275运行时间7.979
------------------------第9次测试------------------------
插入排序: 腾挪次数：24782750  比较次数：24782744  运行时间687.142
合并排序: 腾挪次数：140000  比较次数：123684运行时间8.976
快速排序: 腾挪次数：162968  比较次数：176435运行时间9.976
------------------------第10次测试------------------------
插入排序: 腾挪次数：25291254  比较次数：25291248  运行时间698.133
合并排序: 腾挪次数：140000  比较次数：123722运行时间8.972
快速排序: 腾挪次数：138730  比较次数：170615运行时间8.988
----------------------总情况----------------------
插入排序：平均腾挪次数：24999887  平均比较次数：24999879平均运行时间697.52
合并排序：平均腾挪次数：140000  平均比较次数：123688平均运行时间8.669
快速排序：平均腾挪次数：146053  平均比较次数：164172平均运行时间8.733
```

N=100000

```
N=100000
构造10次随机原始序列:
------------------------第1次测试------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：1700000  比较次数：1566514运行时间103.374
快速排序: 腾挪次数：1785644  比较次数：2092572运行时间106.751
```

```
-------------------------第2次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：1700000  比较次数：1566482运行时间102.746
快速排序: 腾挪次数：2011874  比较次数：2145910运行时间115.228
-------------------------第3次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：1700000  比较次数：1566501运行时间103.23
快速排序: 腾挪次数：2044898  比较次数：2199411运行时间112.907
-------------------------第4次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：1700000  比较次数：1566341运行时间104.251
快速排序: 腾挪次数：1745834  比较次数：2087402运行时间104.908
-------------------------第5次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：1700000  比较次数：1566283运行时间105.225
快速排序: 腾挪次数：1790484  比较次数：2091532运行时间105.226
-------------------------第6次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：1700000  比较次数：1566444运行时间103.722
快速排序: 腾挪次数：1889598  比较次数：2082892运行时间111.702
-------------------------第7次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：1700000  比较次数：1566537运行时间103.724
快速排序: 腾挪次数：1769492  比较次数：2085727运行时间107.712
-------------------------第8次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：1700000  比较次数：1566342运行时间103.722
快速排序: 腾挪次数：1834410  比较次数：2132055运行时间108.711
-------------------------第9次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：1700000  比较次数：1566727运行时间103.724
快速排序: 腾挪次数：1874600  比较次数：2102462运行时间107.168
-------------------------第10次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：1700000  比较次数：1566415运行时间105.827
快速排序: 腾挪次数：1836156  比较次数：2061210运行时间107.713
----------------------总情况----------------------
插入排序：平均腾挪次数：INF  平均比较次数：INF平均运行时间INF
合并排序：平均腾挪次数：1700000  平均比较次数：1566458平均 运行时间103.96
快速排序：平均腾挪次数：1858299  平均比较次数：2108117平均运行时间108.80
```

N=1000000

```
N=1000000
构造10次随机原始序列:
-------------------------第1次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：20000000  比较次数：18716164运行时间1255.88
快速排序: 腾挪次数：22607830  比较次数：37561564运行时间1730.01
-------------------------第2次测试-------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
```

```
合并排序: 腾挪次数：20000000  比较次数：18716361运行时间1239.07
快速排序: 腾挪次数：21691938  比较次数：36003737运行时间1609.5
------------------------第3次测试------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：20000000  比较次数：18716139运行时间1269.71
快速排序: 腾挪次数：21264938  比较次数：35153019运行时间1595.75
------------------------第4次测试------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：20000000  比较次数：18714672运行时间1256.25
快速排序: 腾挪次数：22581378  比较次数：36274983运行时间1686.48
------------------------第5次测试------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：20000000  比较次数：18716069运行时间1246.78
快速排序: 腾挪次数：20446144  比较次数：36142673运行时间1599.79
------------------------第6次测试------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：20000000  比较次数：18715129运行时间1235.28
快速排序: 腾挪次数：20408936  比较次数：35904266运行时间1572.82
------------------------第7次测试------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：20000000  比较次数：18716018运行时间1213.44
快速排序: 腾挪次数：20066808  比较次数：36690773运行时间1575.81
------------------------第8次测试------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：20000000  比较次数：18715010运行时间1220.45
快速排序: 腾挪次数：20221116  比较次数：36690962运行时间1580.32
------------------------第9次测试------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：20000000  比较次数：18715826运行时间1219.03
快速排序: 腾挪次数：21199768  比较次数：35809143运行时间1576.54
------------------------第10次测试------------------------
插入排序: 腾挪次数：INF  比较次数：INF  运行时间INF
合并排序: 腾挪次数：20000000  比较次数：18715371运行时间1223.86
快速排序: 腾挪次数：20686440  比较次数：35361099运行时间1553.09
----------------------总情况----------------------
插入排序：平均腾挪次数：INF  平均比较次数：INF平均运行时间INF
合并排序：平均腾挪次数：20000000  平均比较次数：18715675平均运行时间1237.98
快速排序：平均腾挪次数：21117529  平均比较次数：36159221平均运行时间1608.01
```

腾挪次数比较

N	插入排序	合并排序	快速排序
10	28	40	21
100	2442	700	547
1000	250469	10000	9396
10000	24999887	140000	146053
100000	INF	1700000	1858299
1000000	INF	20000000	21117529
10000000	INF	240000000	240209383

插入排序的腾挪次数远大于合并排序与快速排序。合并排序在N较小时的腾挪次数大于快速排序，在N较大时的腾挪次数小于快速排序。

比较次数比较

N	插入排序	合并排序	快速排序
10	27	23	32
100	2438	560	722
1000	250464	8718	11586
10000	24999879	123688	164172
100000	INF	1566458	2108117
1000000	INF	18715675	36159221
10000000	INF	224002210	1733804951

插入排序的比较次数远大于合并排序与快速排序。合并排序腾挪次数小于快速排序，而且在表中最后一行可以看到，在N=1e7时，快速排序的比较次数显著地提高了。

运行时间比较(ms)

N	插入排序	合并排序	快速排序
10	0	0	0
100	0	0.0997	0
1000	7.334	0.699	0.598
10000	697.52	8.669	8.733
100000	INF	103.96	108.80
1000000	INF	1237.98	1608.01
10000000	INF	14576.1	57602.5

插入排序的运行时间远大于合并排序与快速排序。合并排序在N较小时的运行时间大于快速排序，在N较大时的运行时间小于快速排序。

总结

插入排序的效率远不如合并排序与快速排序。而合并排序与快速排序不相上下，十分接近。具体来说，在数据规模较小时快速排序效率较高，在数据规模较大时合并排序效率较高。这是由于快速排序在数据规模较大时的比较次数显著上升导致的。

六、实验总结

在本次排序算法比较实验中，我对三种算法的执行过程、运行效率都有了一个清晰的认知。其中给我留下最深刻印象的是快速排序的最优情况数据构造与合并排序的最差情况的数据构造。我反复思索这两种排序方式的排序过程，画出例子进行思考，多次修改程序，最终才成功地写出了数据构造程序。这次实验带给我的收获不仅在于加深了对于排序算法的理解，更在于对问题的思考与解决能力。