### 三种排序性能的比较

1. **题目要求**

将一个整数序列按升序排列

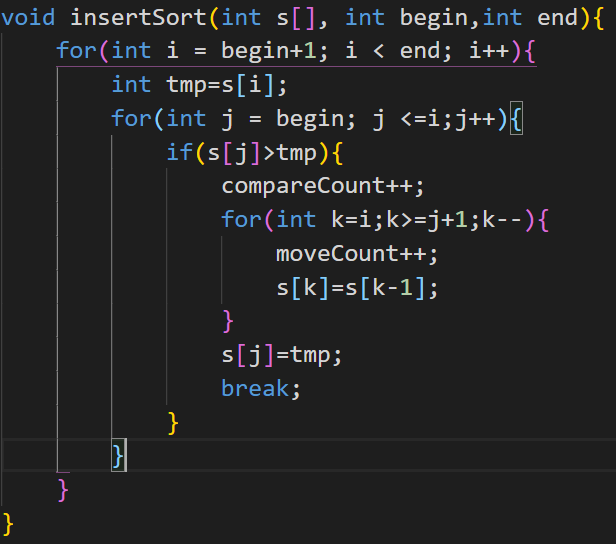
1. **实现方法**

* **插入排序**

思想：利用有序表的插入操作进行排序

平均时间复杂度为：

代码实现：



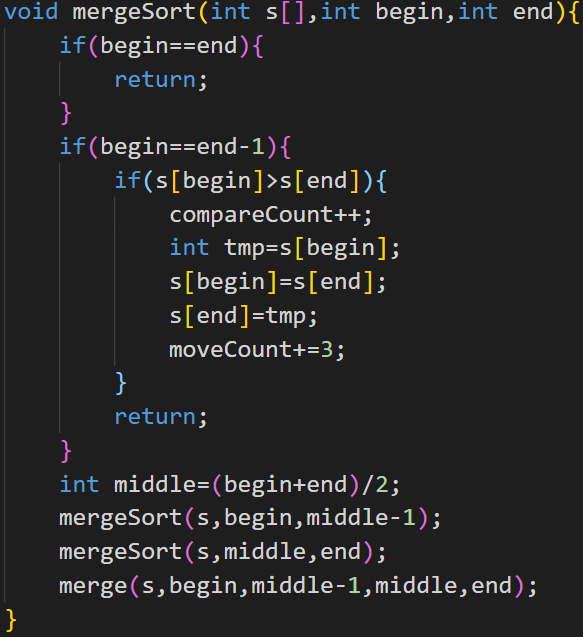
* **归并排序**

思想：将个元素分成各含个元素的子序列，递归地用归并排序函数对两个子序列排序，合并两个已经排好序的子序列

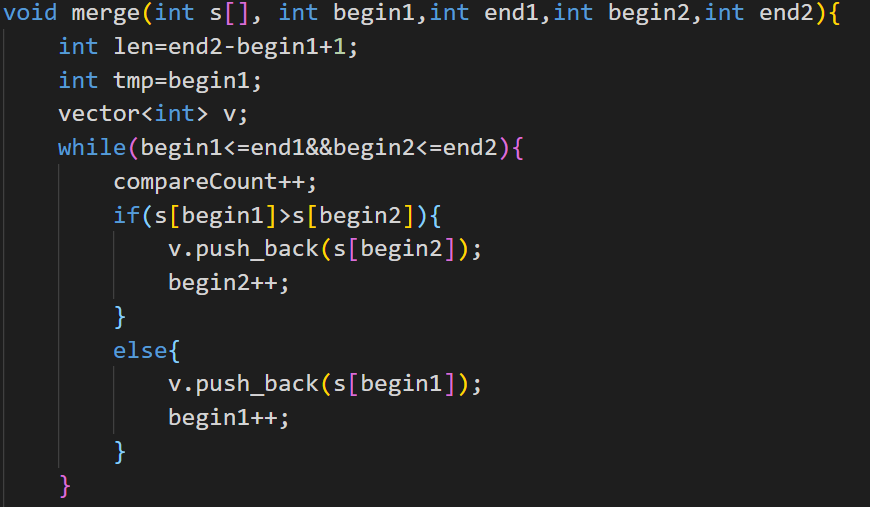
平均时间复杂度为：

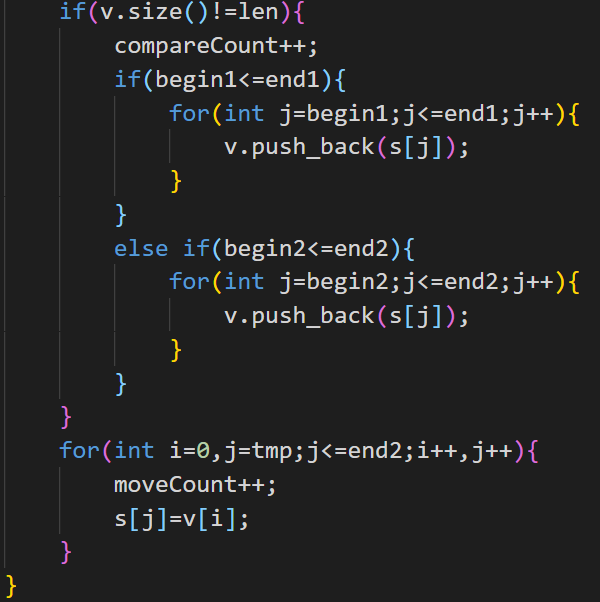
代码实现：

1.分解部分：



1. 合并部分



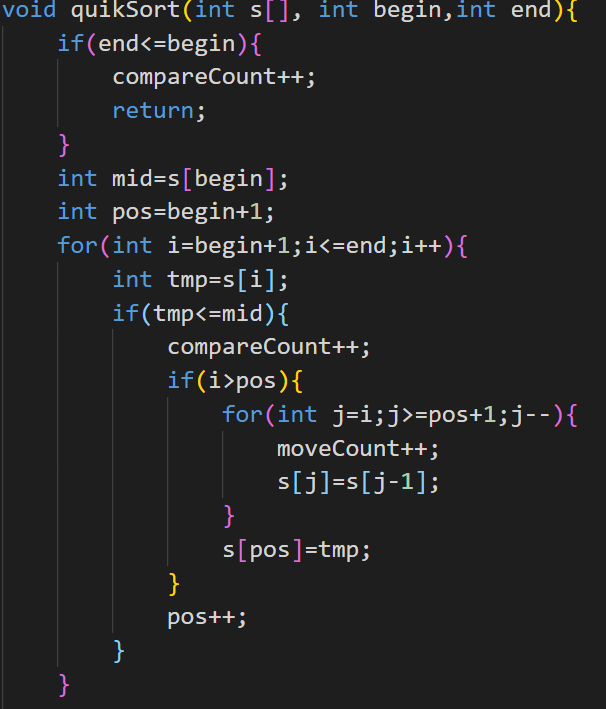


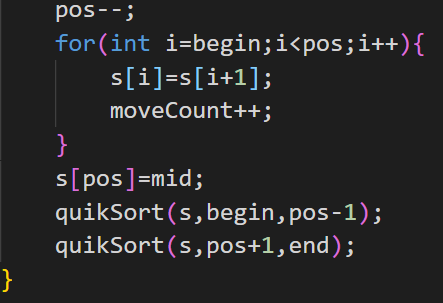
* **快速排序**

思路：对于的子序列，将以为标杆元素，分为两个子序列，即，其中中的值均小于，中的值均大于，对递归地调用快速排序函数，不需要合并。

平均时间复杂度为：

代码实现：

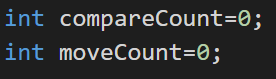




1. **白盒测试**

**思路：**

**设置全局变量存储一种算法的比较次数和腾挪次数**



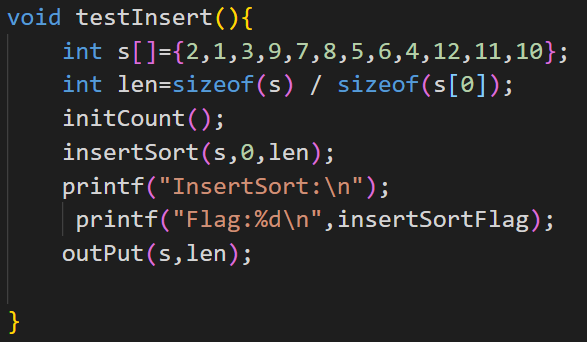
对于每种算法的每个分支设置标志位，当进入到那个分支时，标志位置1

* **插入排序**

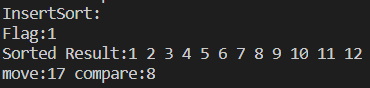
有1个分支，设置1个标志位：



输出标志位情况：

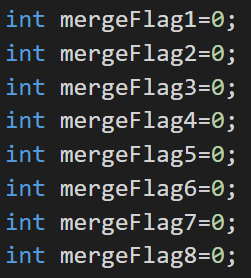


输出结果：

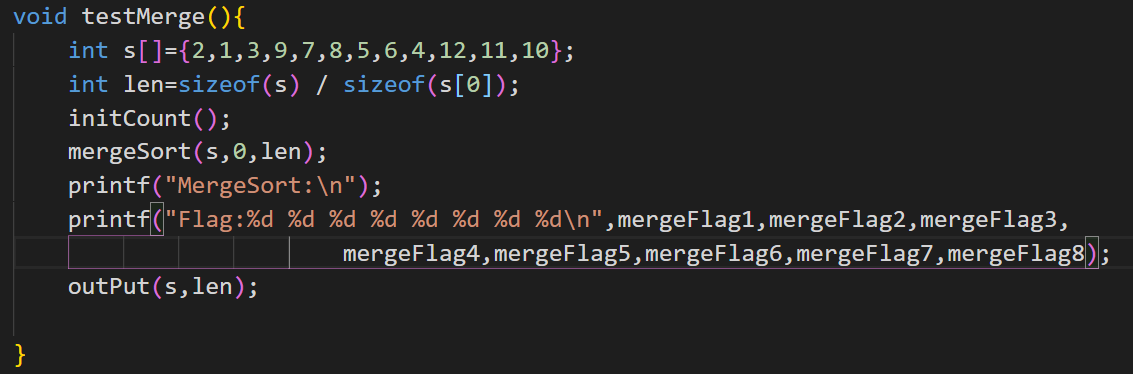


* **归并排序**

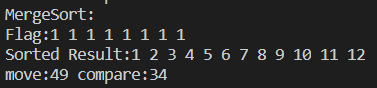
有8个分支，设置8个标志位：



输出标志位情况：

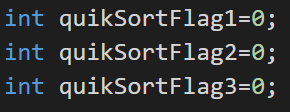


输出结果：

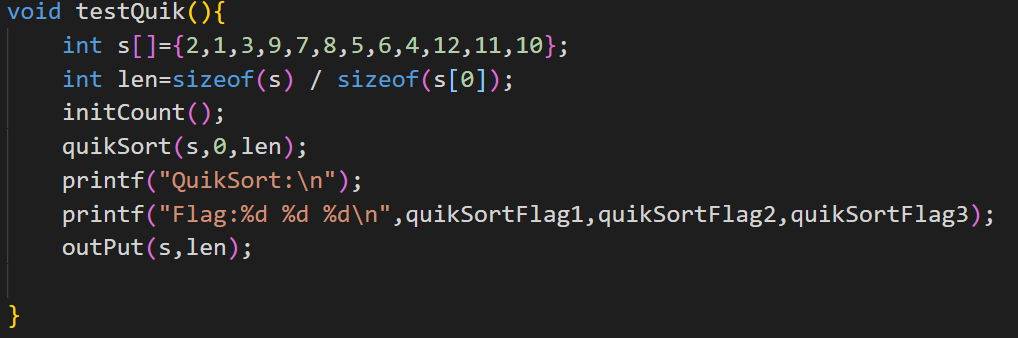


* **快速排序**

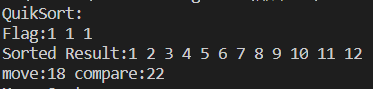
有3个分支，设置3个标志位：



输出标志位情况：



输出结果：

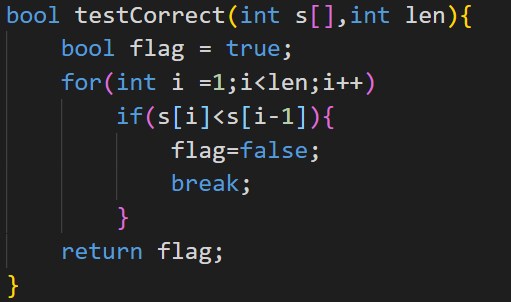


1. **黑盒测试**

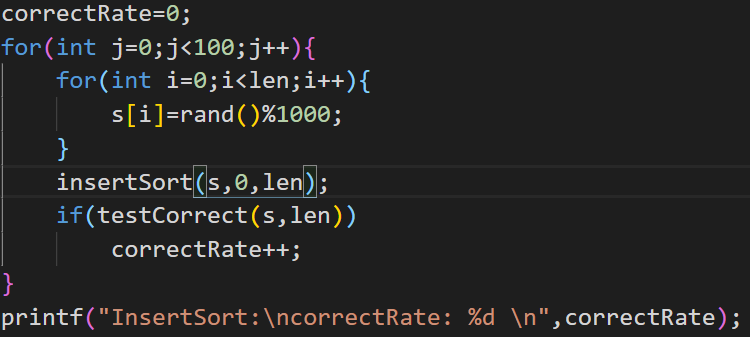
随机生成整数序列，检测排序结果是否正确

对于每种算法，随机生成100组测试样例，每组测试样例包含100个0~1000的整数序列

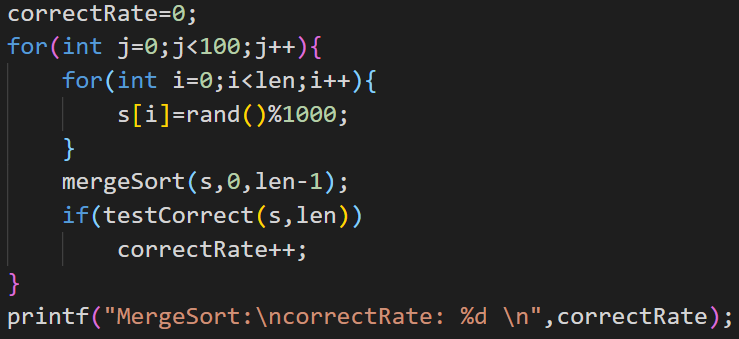
* **验证函数：用来验证排序是否正确**



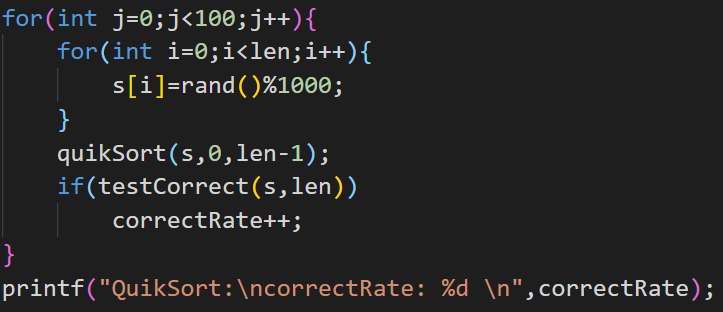
* **插入排序**



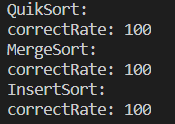
* **归并排序**



* **快速排序**

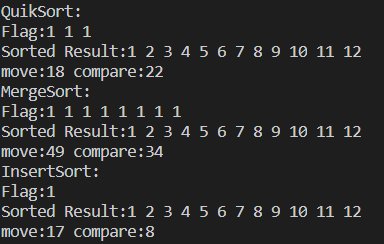


**输出结果：**



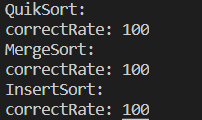
1. **结果分析**

* **白盒测试结果分析：**



测试数据遍历了三种算法的所有分支，由上图的输出可看出，腾挪次数和比较次数均呈现出插入排序<快速排序<归并排序的规律

* **黑盒测试结果分析：**



随机产生100组测试数据

由上图结果可看出，正确率均为100%

1. **实验总结**

通过这次对三种排序算法的比较以及白盒、黑盒测试的代码实现，让我对这三种算法之间的区别有了更清晰的认识，也让我感受到白盒测试可以更直观地体现出算法的特性。虽然快速排序和归并排序平均时间复杂度都为

但相对于插入排序有更多的比较次数和腾挪次数，且快速排序在最坏情况下的时间复杂度同插入排序相同，这三种算法各有特点，需要根据具体情况灵活使用。

1. **实现代码**

#include<cstdio>

#include<vector>

#include <stdlib.h>

#include <time.h>

using namespace std;

int compareCount=0;

int moveCount=0;

int quikSortFlag1=0;

int quikSortFlag2=0;

int quikSortFlag3=0;

int mergeFlag1=0;

int mergeFlag2=0;

int mergeFlag3=0;

int mergeFlag4=0;

int mergeFlag5=0;

int mergeFlag6=0;

int mergeFlag7=0;

int mergeFlag8=0;

int insertSortFlag=0;

void quikSort(int s[], int begin,int end){

if(end<=begin){

quikSortFlag1=1;

compareCount++;

return;

}

int mid=s[begin];

int pos=begin+1;

for(int i=begin+1;i<=end;i++){

int tmp=s[i];

if(tmp<=mid){

quikSortFlag2=1;

compareCount++;

if(i>pos){

quikSortFlag3=1;

for(int j=i;j>=pos+1;j--){

moveCount++;

s[j]=s[j-1];

}

s[pos]=tmp;

}

pos++;

}

}

pos--;

for(int i=begin;i<pos;i++){

s[i]=s[i+1];

moveCount++;

}

s[pos]=mid;

quikSort(s,begin,pos-1);

quikSort(s,pos+1,end);

}

void merge(int s[], int begin1,int end1,int begin2,int end2){

int len=end2-begin1+1;

int tmp=begin1;

vector<int> v;

while(begin1<=end1&&begin2<=end2){

compareCount++;

if(s[begin1]>s[begin2]){

mergeFlag1=1;

v.push\_back(s[begin2]);

begin2++;

}

else{

mergeFlag2=1;

v.push\_back(s[begin1]);

begin1++;

}

}

if(v.size()!=len){

mergeFlag3=1;

compareCount++;

if(begin1<=end1){

mergeFlag4=1;

for(int j=begin1;j<=end1;j++){

v.push\_back(s[j]);

}

}

else if(begin2<=end2){

mergeFlag5=1;

for(int j=begin2;j<=end2;j++){

v.push\_back(s[j]);

}

}

}

for(int i=0,j=tmp;j<=end2;i++,j++){

moveCount++;

s[j]=v[i];

}

}

void mergeSort(int s[],int begin,int end){

if(begin==end){

mergeFlag6=1;

return;

}

if(begin==end-1){

mergeFlag7=1;

if(s[begin]>s[end]){

mergeFlag8=1;

compareCount++;

int tmp=s[begin];

s[begin]=s[end];

s[end]=tmp;

moveCount+=3;

}

return;

}

int middle=(begin+end)/2;

mergeSort(s,begin,middle-1);

mergeSort(s,middle,end);

merge(s,begin,middle-1,middle,end);

}

void insertSort(int s[], int begin,int end){

for(int i = begin+1; i < end; i++){

int tmp=s[i];

for(int j = begin; j <=i;j++){

if(s[j]>tmp){

insertSortFlag=1;

compareCount++;

for(int k=i;k>=j+1;k--){

moveCount++;

s[k]=s[k-1];

}

s[j]=tmp;

break;

}

}

}

}

void initCount(){

compareCount=0;

moveCount=0;

}

void outPut(int s[],int len){

printf("Sorted Result:");

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

printf("%d ",s[i]);

printf("\n");

printf("move:%d compare:%d\n",moveCount,compareCount);

}

void testQuik(){

int s[]={2,1,3,9,7,8,5,6,4,12,11,10};

int len=sizeof(s) / sizeof(s[0]);

initCount();

quikSort(s,0,len);

printf("QuikSort:\n");

printf("Flag:%d %d %d\n",quikSortFlag1,quikSortFlag2,quikSortFlag3);

outPut(s,len);

}

void testMerge(){

int s[]={2,1,3,9,7,8,5,6,4,12,11,10};

int len=sizeof(s) / sizeof(s[0]);

initCount();

mergeSort(s,0,len);

printf("MergeSort:\n");

printf("Flag:%d %d %d %d %d %d %d %d\n",mergeFlag1,mergeFlag2,mergeFlag3,

mergeFlag4,mergeFlag5,mergeFlag6,mergeFlag7,mergeFlag8);

outPut(s,len);

}

void testInsert(){

int s[]={2,1,3,9,7,8,5,6,4,12,11,10};

int len=sizeof(s) / sizeof(s[0]);

initCount();

insertSort(s,0,len);

printf("InsertSort:\n");

printf("Flag:%d\n",insertSortFlag);

outPut(s,len);

}

bool testCorrect(int s[],int len){

bool flag = true;

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

if(s[i]<s[i-1]){

flag=false;

break;

}

return flag;

}

void whiteTest(){

testQuik();

testMerge();

testInsert();

}

void blackTest(){

srand(time(NULL));

int correctRate=0;

int len=100;

int s[100];

for(int j=0;j<100;j++){

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

s[i]=rand()%1000;

}

quikSort(s,0,len-1);

if(testCorrect(s,len))

correctRate++;

}

printf("QuikSort:\ncorrectRate: %d \n",correctRate);

correctRate=0;

for(int j=0;j<100;j++){

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

s[i]=rand()%1000;

}

mergeSort(s,0,len-1);

if(testCorrect(s,len))

correctRate++;

}

printf("MergeSort:\ncorrectRate: %d \n",correctRate);

correctRate=0;

for(int j=0;j<100;j++){

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

s[i]=rand()%1000;

}

insertSort(s,0,len);

if(testCorrect(s,len))

correctRate++;

}

printf("InsertSort:\ncorrectRate: %d \n",correctRate)；

}

int main(){

blackTest();

whiteTest();

}