第六次实验报告

实验题目

要求采用三种不同的初始数据测试：1、基本无序。2、递增。3、递减。

测试比较各种排序算法。测试数据一般在5000-10000之间，以保证测试时间足够大，测试每种算法的执行时间并对结果进行分析。

实验过程

1. 使用生成器生成三种序列。
2. 应用各种算法，并使用clock进行时间分析。

效果截图

test size:6000

\*random asc desc

bubble 165 92 116

insert 31 6 59

select 62 60 59

quick 1 2 1

rx 1 0 0

heap 1 1 1

test size:10000

\*random asc desc

bubble 417 280 368

insert 86 11 154

select 168 161 156

quick 1 3 2

rx 0 0 0

heap 3 2 1

test size:14000

\*random asc desc

bubble 812 468 625

insert 162 19 298

select 310 308 302

quick 2 5 3

rx 1 1 1

heap 3 3 3

test size:18000

\*random asc desc

bubble 1324 768 1017

insert 262 33 489

select 530 516 510

quick 2 7 4

rx 1 2 1

heap 4 4 4

test size:22000

\*random asc desc

bubble 1983 1136 1523

insert 390 48 728

select 774 766 767

quick 4 9 4

rx 1 1 2

heap 6 5 4

test size:26000

\*random asc desc

bubble 2809 1593 2124

insert 543 69 1010

select 1096 1068 1069

quick 5 12 5

rx 2 2 1

heap 6 5 5

test size:30000

\*random asc desc

bubble 3725 2116 2822

insert 720 90 1344

select 1437 1420 1451

quick 5 10 7

rx 1 2 2

heap 8 7 6

test size:34000

\*random asc desc

bubble 5182 2777 3884

insert 966 118 1830

select 1912 1860 1888

quick 6 16 6

rx 2 2 2

heap 8 8 7

test size:38000

\*random asc desc

bubble 6113 3425 4562

insert 1159 145 2169

select 2319 2275 2333

quick 7 17 8

rx 2 2 2

heap 10 9 8

test size:42000

\*random asc desc

bubble 7328 4156 5575

insert 1417 179 2653

select 2827 2763 2795

quick 7 18 14

rx 3 3 3

heap 11 9 9

test size:46000

\*random asc desc

bubble 8764 4986 6678

insert 1691 207 3184

select 3373 3307 3352

quick 8 17 9

rx 2 3 3

heap 12 10 10

test size:50000

\*random asc desc

bubble 10441 5901 7899

insert 2017 253 3819

select 3994 3932 3991

quick 8 23 10

rx 3 3 3

heap 13 11 10

test size:54000

\*random asc desc

bubble 12141 6882 9428

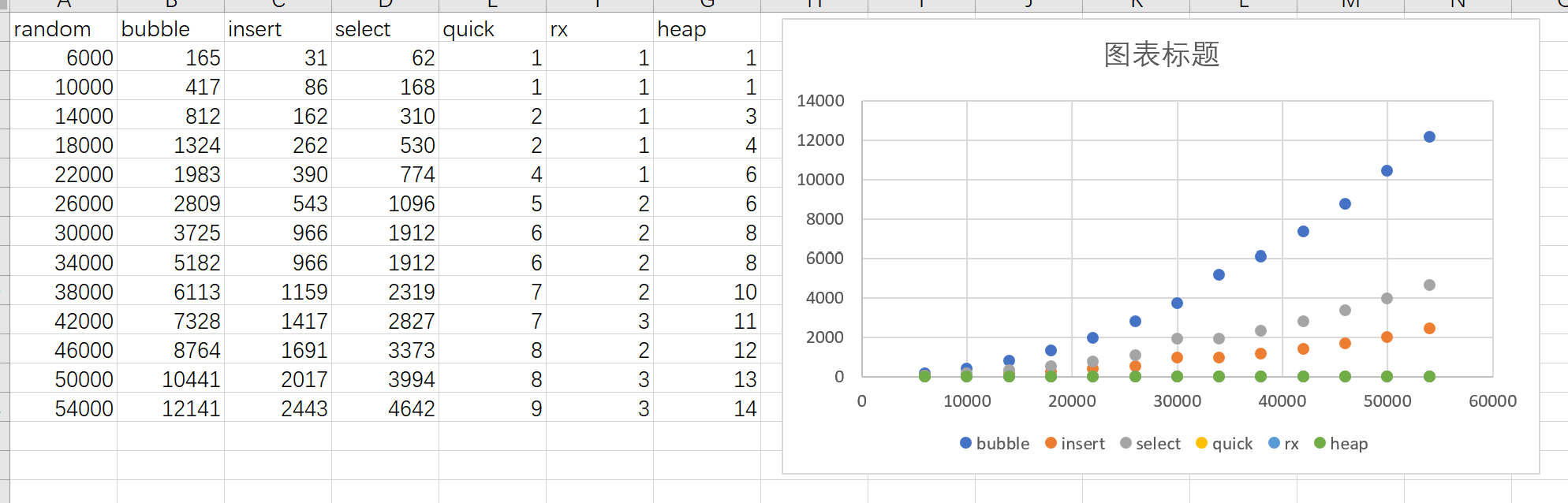
insert 2443 291 4383

select 4642 4580 4603

quick 9 23 15

rx 3 3 3

heap 14 12 11



如图分析，冒牌排序，插入排序，选择排序都比较满，且呈现明显的*O*(*n*2)的指数上升的趋势。

代码

main.cpp

#include <iostream>

#include <ctime>

#include <sys/utime.h>

#include "sorthelper.h"

#include "msort.h"

using namespace std;

int main() {

for(int i = 6000; i < 55000; i+= 4000){

test\_a\_size(i);

cout << endl;

}

}

sorthelper.h

*//*

*// Created by cht on 2019/12/18.*

*//*

#pragma once

#include <random>

#include <ctime>

#include "msort.h"

#include <iostream>

#include <iomanip>

using namespace std;

enum class sort\_type{

bubble,

insert,

select,

quick,

rx,

heap

};

using namespace std;

default\_random\_engine e;

int rand\_int(int start, int end){

uniform\_int\_distribution<unsigned> u(start, end -1);

return u(e);

}

void build\_data(int\* a,int length, int way){

*//int length = 200;*

int max\_n = 10000;

if(way == 0){ *//random generate*

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

a[i] = rand\_int(0,max\_n);

}

} else {

double step = max\_n / (double)length;

if(way == 1){

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

a[i] = (int)( i \* step);

}

} else {

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

a[i] = (int)( (length - 1 - i) \* step);

}

}

int swap\_count = length / 20;

for(int i = 0; i< swap\_count; ++i){

int swap\_i = rand\_int(0, length);

int swap\_j = rand\_int(0, length);

swap(a[swap\_i], a[swap\_j]);

}

}

}

long test\_sort(int\* a ,int length, sort\_type type){

clock\_t start = clock();

if(type == sort\_type::bubble){

m\_bubble\_sort(a, a + length);

} else if(type == sort\_type::select){

m\_select\_sort(a, a + length);

} else if(type == sort\_type::insert){

m\_insert\_sort(a, a + length);

} else if(type == sort\_type::quick){

m\_quick\_sort(a, a + length);

} else if(type == sort\_type::rx){

m\_rx\_sort(a, a + length, 4);

} else {

m\_heap\_sort(a, a + length);

}

clock\_t end = clock();

return end-start;

}

void test\_a\_size(int size){

int column\_size = 6;

cout << "test size:" << size << endl;

cout << setw(column\_size) << "\*" << setw(column\_size) << "random"

<< setw(column\_size) << "asc" << setw(column\_size) << "desc" << endl;

sort\_type st[] = {sort\_type::bubble, sort\_type::insert, sort\_type::select,

sort\_type::quick, sort\_type::rx, sort\_type::heap};

string sn[] = {"bubble","insert","select","quick","rx","heap"};

for(int i = 0; i < 6; i++){

int\* random\_data = new int[size];

int\* asc\_data = new int[size];

int\* des\_data = new int[size];

build\_data(random\_data, size, 0);

build\_data(asc\_data, size, 1);

build\_data(des\_data, size, 2);

cout << setw(column\_size) << sn[i];

cout << setw(column\_size) << test\_sort(random\_data, size, st[i]);

cout << setw(column\_size) << test\_sort(asc\_data, size, st[i]);

cout << setw(column\_size) << test\_sort(des\_data, size, st[i]);

cout << endl;

delete[] random\_data;

delete[] asc\_data;

delete[] des\_data;

}

}

msort.h

*//*

*// Created by cht on 2019/12/18.*

*//*

#pragma once

void m\_swap(int\* first, int\* second){

int temp = \*first;

\*first = \*second;

\*second = temp;

}

bool compare(int first, int second, int direction = 0){

if(direction >= 0 && first > second)

return true;

if(direction < 0 && first < second)

return true;

return false;

}

void m\_bubble\_sort(int\* start, int\* end, int direction = 0){

for(int\* pi = start; pi < end - 1 ; ++pi){

for(int\* pj = end - 1; pj > pi; --pj){

if(compare(\*(pj - 1),\*pj, direction))

m\_swap(pj - 1, pj);

}

}

}

void m\_select\_sort(int\* start, int\* end, int direction = 0){

for(int\* pi = start; pi < end - 1; ++pi){

int\* temp = pi;

for(int\* pj = pi + 1; pj < end; ++pj){

if(compare(\*temp, \*pj))

temp = pj;

}

m\_swap(pi, temp);

}

}

void m\_insert\_sort(int\* start, int\* end, int direction = 0){

for(int\* pi = start + 1; pi < end; ++pi){

int temp = \*pi;

int\* pj = pi;

for(pj = pi; pj >= start; pj--){

if(compare (\*(pj-1), temp) )

\*pj = \*(pj-1);

else

break;

}

\*(pj) = temp;

}

}

int\* quick\_v(int\* start, int\* end, int direction = 0){

int temp = \*start;

while(start < end){

while(start < end && !compare(temp,\*end)){

end--;

}

\*start = \*end;

while(start < end && !compare(\*start, temp)){

start++;

}

\*end = \*start;

}

\*start = temp;

return start;

}

void \_m\_quick\_sort(int\* start, int\* end, int direction = 0){

if(start < end){

int\* middle = quick\_v(start, end, direction);

\_m\_quick\_sort(start, middle-1 ,direction);

\_m\_quick\_sort(middle+1, end, direction);

}

}

void m\_quick\_sort(int\* start, int\* end, int direction = 0){

\_m\_quick\_sort(start, end-1);

}

void adjust\_heap(int\* a, int node, int size){

int left = 2\*node + 1;

int right = 2\*node +2;

int max = node;

if(left < size && a[left] > a[max])

max = left;

if(right < size && a[right] > a[max])

max = right;

if(max != node){

m\_swap(a+max,a+node);

adjust\_heap(a, max, size);

}

}

void m\_heap\_sort(int\* start, int\* end, int direction = 0){

int len = end - start;

for(int i = len/2 - 1; i >= 0; --i)

adjust\_heap(start, i, len);

for(int i = len-1; i >= 0; --i){

m\_swap(start, start+i);

adjust\_heap(start, 0, i);

}

}

void m\_rx\_sort(int\* start, int\* end, int d){

int k = 10;

int length = end - start;

int\* counts = new int[k];

int\* temp = new int[length];

int index;

int pval = 1;

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

for (int j = 0; j < k; ++j)

counts[j] = 0;

for(int\*p = start; p < end; ++p){

index = (\*p / pval) % k;

counts[index]++;

}

for(int j = 1; j < k;++j){

counts[j] += counts[j-1];

}

for(int\*p = end-1; p >= start;--p){

index = (\*p / pval) % k;

temp[counts[index] - 1] = \*p;

counts[index]--;

}

for(int \*p =start; p < end; ++p) {

\*p = temp[p - start];

}

pval\*= k;

}

delete[] counts;

delete[] temp;

}