## 



Harbin Institute of Technology

实验报告

课程名称： 数据结构与算法

实验题目： 选择排序和堆排序的比较分析

实验类型： 必 修

所在院系： 计算机科学与技术学院

学生类别： 本科生

指导教师： 李秀坤

学生姓名： 易亚玲

班级和学号： 1703005 1170300511

实验时间： 2018 年 秋 季学期

哈尔滨工业大学

1. **实验目的**

深入理解选择排序和堆排序的时间性能与数据规模和数据分布的关系。

1. **实验内容和要求**

**2.1.实验内容**

实现一组经典的排序算法，通过实验数据的统计，考察不同规模和分布的数据对排序算法运行时间影响的规律，验证理论分析的正确性。

**2.2实验要求**

1.实现以下三种排序方法中的一组排序算法：

(1)冒泡排序和快速排序

(2)插入排序和希尔排序

(3)选择排序和堆排序

2.产生不同规模和分布的数据，以“图或表”的方式给出输入规模和分布对排序方法运用时间变化趋势的影响（画出T(n)的曲线）。并与理论分析结果比较。

3．将上诉“图或者表”采用图片等形式贴在实验报告中，并作适当分析和说明。

**2.3系统功能需求分析**

**（三）程序实现**

**3.1变量及函数**

int a[MAX\_SIZE+1]; //选择排序的测试数据

int b[MAX\_SIZE+1]; //堆排序的测试数据

void selsort(); //选择排序

void heapsort(); //堆排序，大根堆

void heap(int n,int k); //堆排序子函数

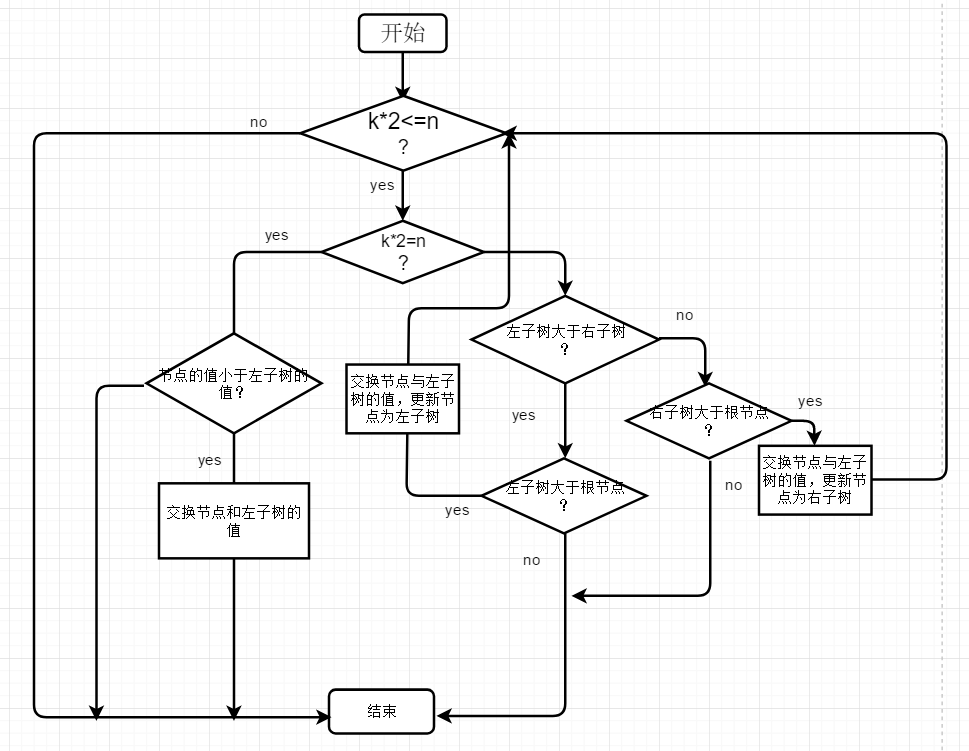
**3.2算法设计**

选择排序：每次选择一个最大的值，初始化最大值的下标k为i，如果有更大的值，更新最大值的下标k，扫描结束后，如果k不为i，那么交换下标为i和k的数据，否则不做更改。

堆排序：首先根据初始数据建立一个大根堆，然后开始排序过程：交换堆顶和最后一个叶节点，相当于把堆顶元素保存在数组中，采用从后往前存的方式，堆的元素减一，整理堆，即整理堆时不操作从堆顶交换下来的元素。整理堆的方式:如果节点的两倍小于或者等于堆的长度，继续整理。在整理过程中，如果节点的两倍等于堆的长度，那么节点只有左子树，如果左子树比节点的值大，交换，否则不交换，比较完以后结束；如果节点的两倍小于堆的长度，那么节点有两株子树，比较两株子树和根节点的大小，如果根节点小于左子树且左子树大于右子树，那么交换根节点和左子树，更新根节点下标为左子树下标；如果根节点小于右子树且右子树大于左子树，交换右子树和根节点，更新根节点下标为右子树下标；否则直接结束整理。

**3.3 数据结构设计**

堆排序整理堆流程图：



**（四）系统测试**

**4.1数据规模测试：**

数据量：500 s: 0 h：0







数据量:1000 s: 0.001 h：0







数据量：1500 s: 0.0035 h: 0.0005









数据量：2000 s: 0.003 h: 0







数据量：2500 s: 0.0057 h: 0







数据量：3000 s: 0.0123 h: 0.001







数据量：3500 s: 0.0123 h: 0.001







数据量：4000 s: 0.0223 h: 0.001







数据量：4500 s: 0.0193 h: 0.001







数据量：5000 s: 0.02725 h: 0.001









数据量：6000 s: 0.0333 h: 0.001







数据量：7000 s: 0.0463 h: 0.0013







数据量：10000 s: 0.0937 h: 0.0013







数据量：15000 s: 0.212 h: 0.002







数据量：20000 s: 0.352 h: 0.003







数据量：25000 s: 0.561 h: 0.0033







数据量：30000 s: 0.7983 h: 0.0043







数据量：35000 s: 1.089 h: 0.006







数据量：40000 s: 1.3973 h: 0.0063







数据量：50000 s: 2.182 h: 0.008









数据量：60000 s: 3.130 h: 0.01







数据量：70000 s: 4.324 h:0.0127







数据量：100000 s:8.31 h:0.0153







数据量：150000 s: 18.898 h:0.0257







数据量：200000 s: 33.686 h: 0.0353







数据量：250000 s: 52.444 h: 0.045







数据量：300000 s: 76.79 h: 0.060







数据量：350000 h:0.073







数据量：400000 h：0.0785





数据量：500000 h: 0.113







数据量：600000 h: 0.130













数据量：700000 h: 0.157





数据量：800000 h:0.185









数据量：900000 h: 0.21



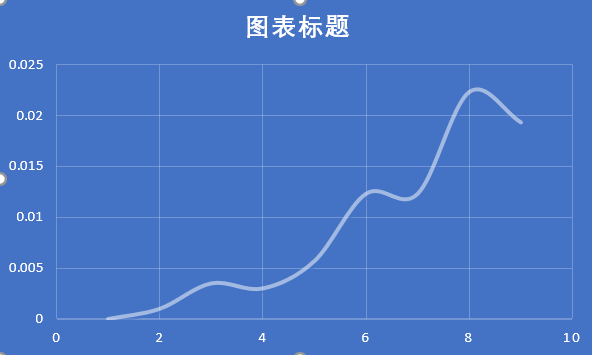




★图形展示：

1. 数据量为 500，1000，…,4500

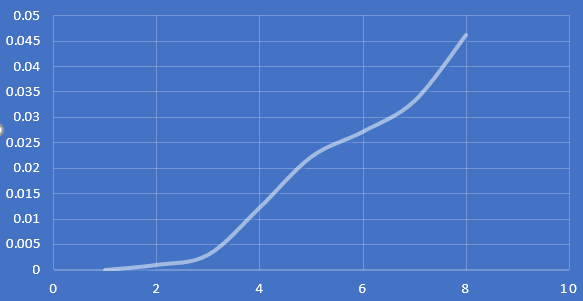
选择排序：



堆排序5000以下不超过0.001s

1. 数据量为 1000，2000，…,7000

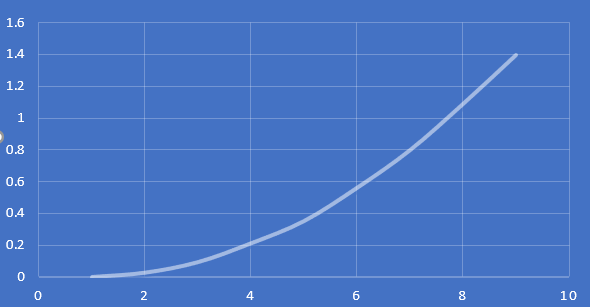
选择排序：



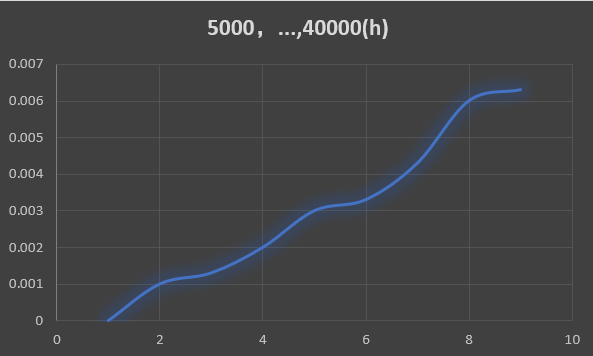
堆排序：7000以下不超过0.002s

1. 数据量为5000，10000，…,40000

选择排序：

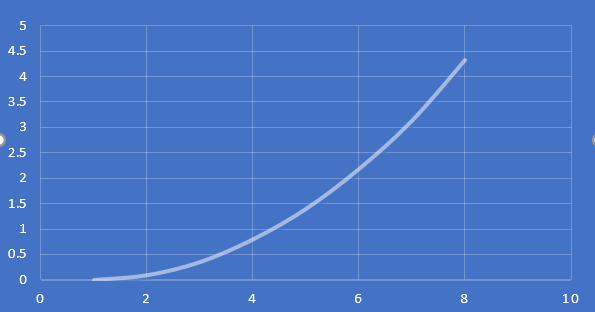


堆排序：

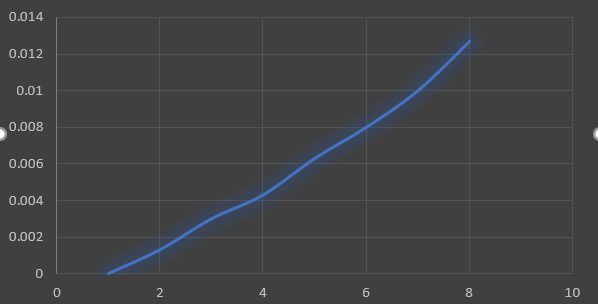


1. 数据量为 10000，20000，…,70000

选择排序：



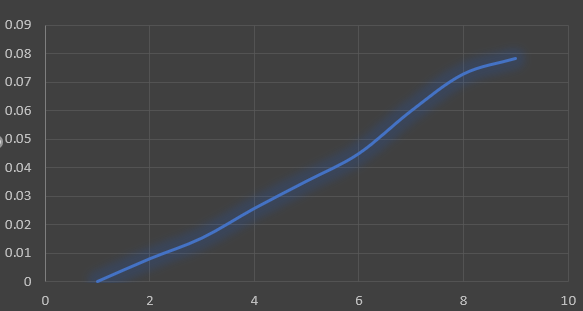
堆排序：



1. 数据量为 50000，100000，…,400000

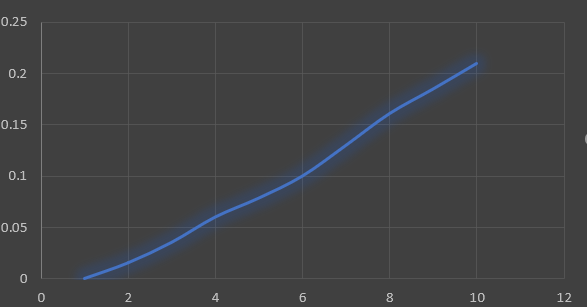
选择排序在之前已经基本与y = x^2重合，所以就不再重复绘制图像

堆排序：



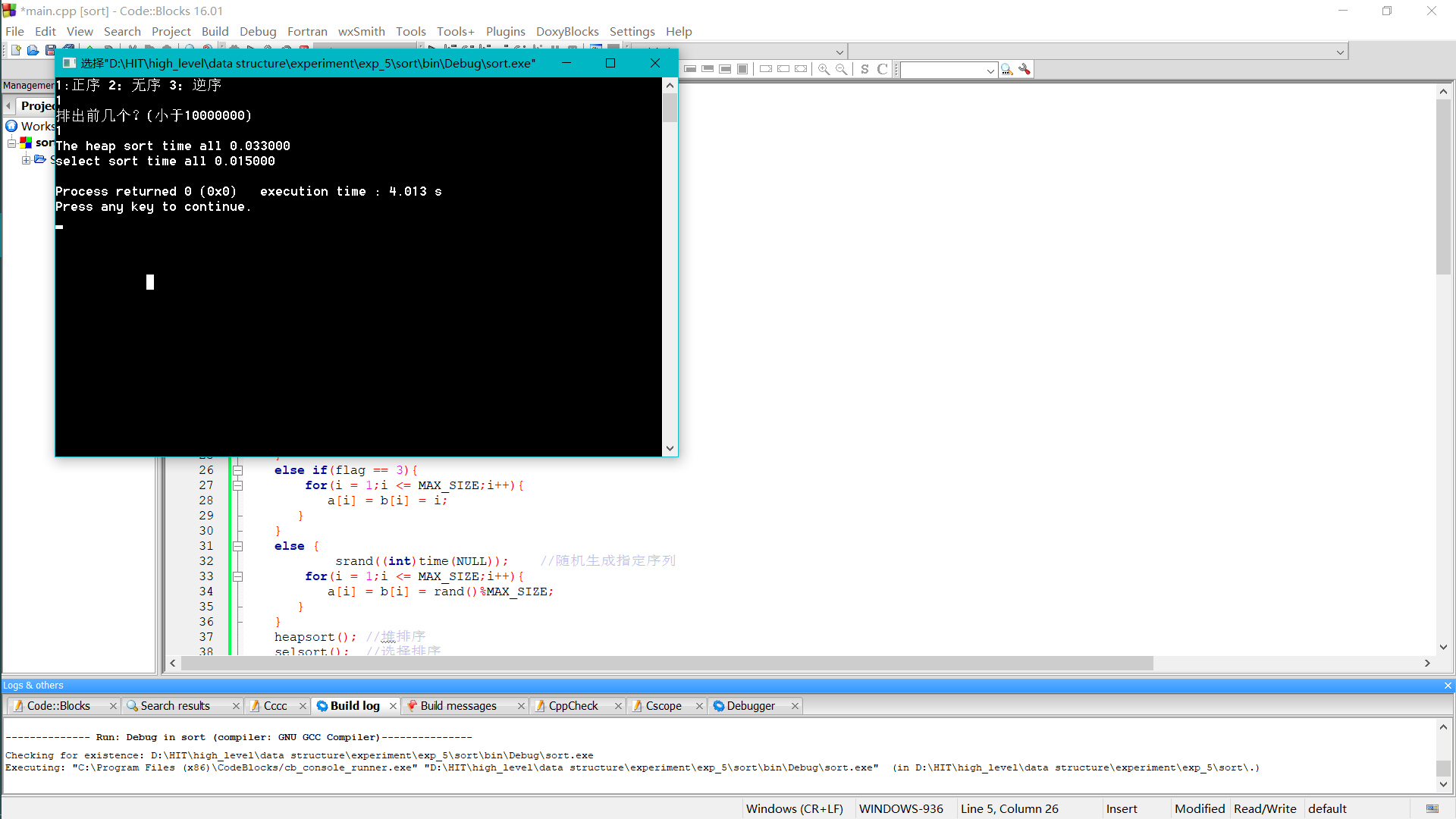
1. 数据量为 100000，200000，…,700000

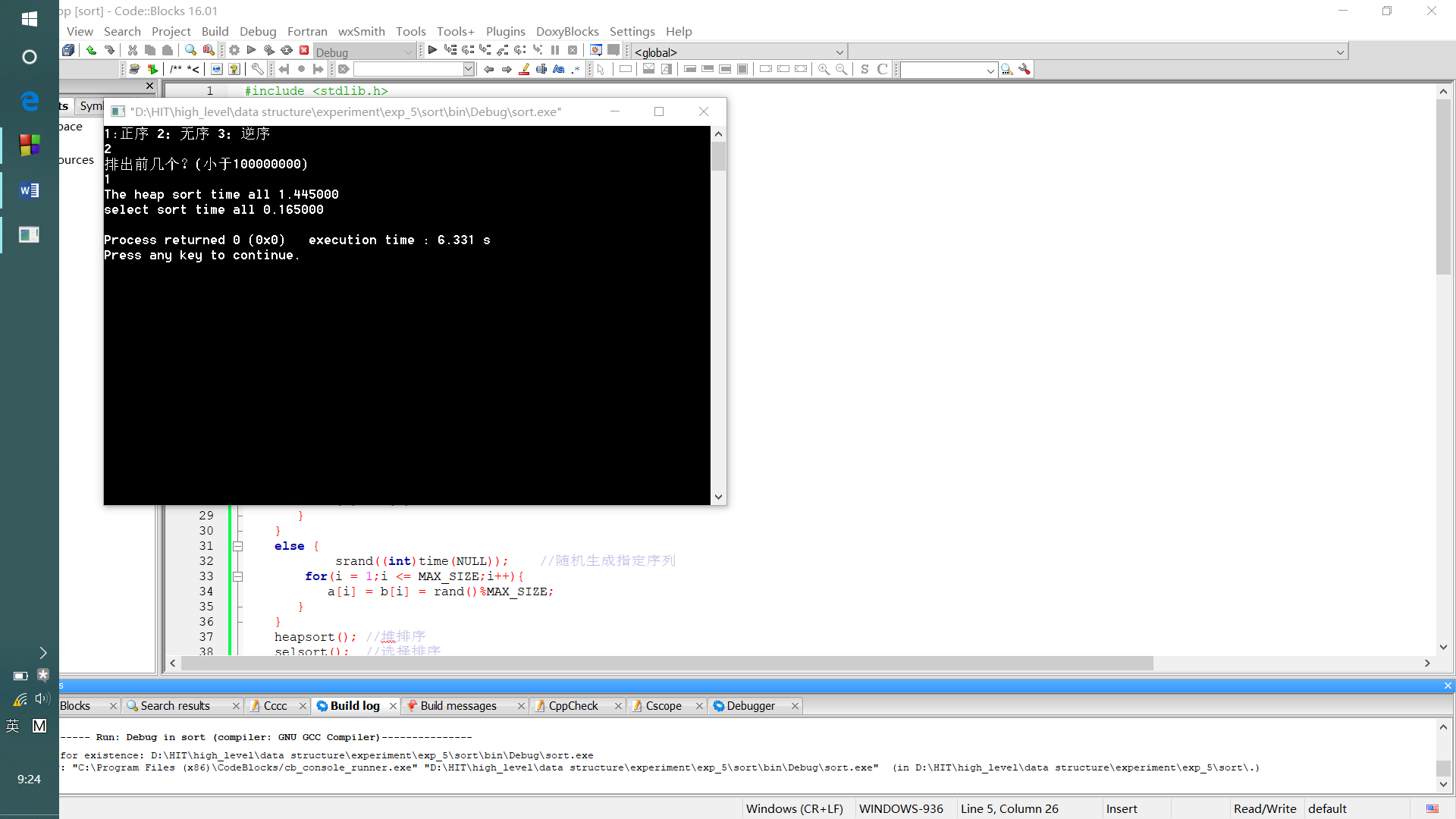
堆排序：

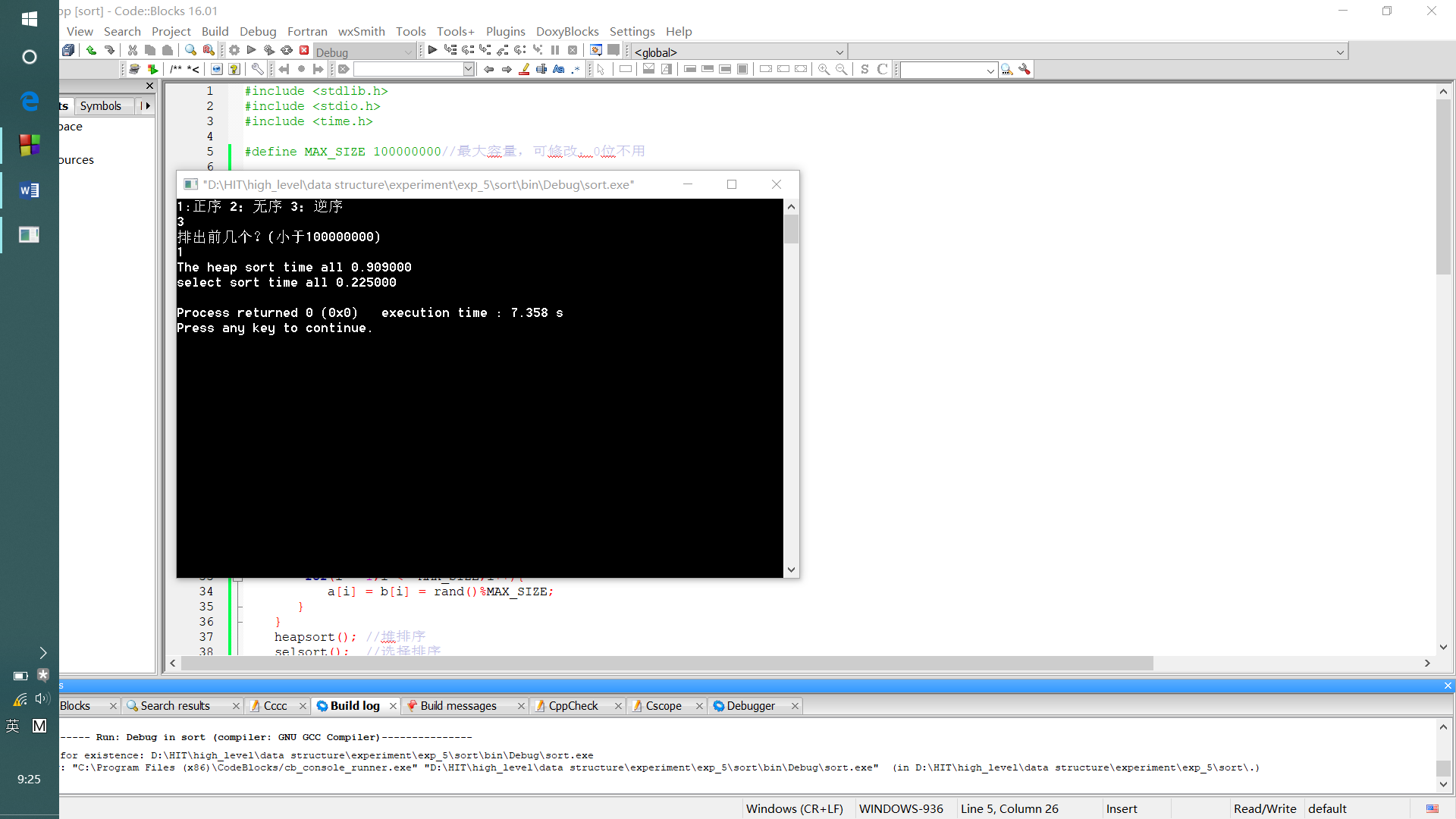


结论：当数据量小于17时，codeblock默认使用选择排序，所以两个时间相等。在数据量较小的时候两者差异不明显，当数据量超过1000时，堆排序明显优于选择排序。

**4.2数据分布测试：**







随着数据规模的增大，选择前n（n较小）个指定的数据，选择排序表现出比堆排序更好的性能，具体原因是建堆和整理的时间代价太大。

**（五）实验总结**

通过这次实验，更加深入地了解了简单排序算法和复杂排序算法受数据分布和数据量的影响，也在分析中提升了自己分析问题的能力。处理数据的过程很有趣，感觉处理完以后很有成就感，感觉就像在做一些大的研究项目一样，感觉非常快乐。

**（六）源代码**

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <windows.h>

#define MAX\_SIZE 10//最大容量，可修改，0位不用

int MAX;

int flag;

int a[MAX\_SIZE+1]; //选择排序的测试数据

int b[MAX\_SIZE+1]; //堆排序的测试数据

void selsort(); //选择排序

void heapsort(); //堆排序，大根堆

void heap(int n,int k); //堆排序子函数

int main()

{

int i;

printf("1:正序 2：无序 3：逆序\n");

scanf("%d",&flag);

printf("排出前几个？(小于%d)\n",MAX\_SIZE);

scanf("%d",&MAX);

if(flag == 1){

for(i = 1;i <= MAX\_SIZE;i++){

a[i] = b[i] = MAX\_SIZE - i;

}

}

else if(flag == 3){

for(i = 1;i <= MAX\_SIZE;i++){

a[i] = b[i] = i;

}

}

else {

srand((int)time(NULL)); //随机生成指定序列

for(i = 1;i <= MAX\_SIZE;i++){

a[i] = b[i] = rand()%MAX\_SIZE;

}

}

heapsort(); //堆排序

selsort(); //选择排序

return 0;

}

void selsort(){

int i,j,k,tmp;

//clock\_t start,finish;

double time;

LARGE\_INTEGER nFreq;

LARGE\_INTEGER nBeginTime;

LARGE\_INTEGER nEndTime;

QueryPerformanceFrequency(&nFreq);

QueryPerformanceCounter(&nBeginTime);

//start = clock(); //计时开始

for(i = 1;i < MAX\_SIZE && i <= MAX ;i++){

k = i;

for(j = i+1;j <= MAX\_SIZE;j++){

if(a[j] > a[k])

k = j;

}

if(i!=k){ //如果需要交换

tmp = a[i];

a[i] = a[k];

a[k] = tmp;

}

}

QueryPerformanceCounter(&nEndTime);

time = (double)(nEndTime.QuadPart - nBeginTime.QuadPart) / nFreq.QuadPart;

//finish = clock(); //计时结束

//time = (double)(finish - start);

//printf("select sort time all %lf\n",time / CLOCKS\_PER\_SEC);

printf("select sort time all %lf us\n",time\*1000);

}

//堆排序

void heapsort(){

int i,tmp;

//clock\_t start,finish;

double time;

LARGE\_INTEGER nFreq;

LARGE\_INTEGER nBeginTime;

LARGE\_INTEGER nEndTime;

QueryPerformanceFrequency(&nFreq);

QueryPerformanceCounter(&nBeginTime);

//start = clock(); //计时器开始

for(i = MAX\_SIZE >>1; i > 0;i--)

heap(MAX\_SIZE,i); //初始化堆

for(i = MAX\_SIZE;i > 1 && i >= (MAX\_SIZE - MAX + 1);i--){

tmp = b[i]; //交换堆顶和堆底，堆长度-1

b[i] = b[1];

b[1] = tmp;

heap(i - 1,1);

}

//finish = clock(); //计时结束

//time = (double)(finish - start); //计算总时间

//printf("The heap sort time all %lf\n",time / CLOCKS\_PER\_SEC);

QueryPerformanceCounter(&nEndTime);

time = (double)(nEndTime.QuadPart - nBeginTime.QuadPart) / nFreq.QuadPart;

printf("The heap sort time all %lf us\n",time\*1000);

}

//堆排序子函数

void heap(int n,int k){

int tmp;

while((k<<1) <= n){ //如果k是非叶节点

if(n == (k<<1)){ //如果k节点只有

if(b[n] < b[k]){ //如果叶节点比根节点的值大，需要交换

tmp = b[n];

b[n] = b[k];

b[k] = tmp;

}

break;

}

else{ //如果有两株子树

if(b[2\*k] > b[k] && b[2\*k+1] <= b[2\*k]){ //左大于根节点且，大于右，左子树和根节点交换

tmp = b[2\*k];

b[2\*k] = b[k];

b[k] = tmp;

k = k << 1;

}

else if((2\*k < n)&& b[2\*k+1] > b[k] && b[2\*k] <= b[2\*k+1]){ //右大于根节点，且右大于左，右子树和根节点交换

tmp = b[2\*k+1];

b[2\*k+1] = b[k];

b[k] = tmp;

k = (k << 1) + 1;

}

else //根节点大于两株子树，停止遍历

break;

}

}

}