哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业基础（必修）

实验项目：排序算法及其应用

实验题目：内存排序算法

实验日期：

班级：2237102

学号：2022110504

姓名：蓝鸿翔

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  |  |

**一、实验目的**

排序是计算机科学中的常见任务，它将一组无序的数据元素按照某种规则重新排列，以使得数据呈现有序的状态，便于后续的查找、统计和分析等操作。当数据量较小时，将数据全部读入内存并进行排序的算法称为内存排序算法，常见的内存排序算法有：插入排序、冒泡排序、归并排序、快速排序、堆排序、基数排序等。本实验要求设计并实现上述内存排序算法并比较其运行速度。

**二、实验要求及实验环境**

 实验要求: 从文本文件中将两行数据读入内存，其中第一行有一个整数n(n≤100000)，表示待排序序列的长度，第二行有 n 个整数，用空格隔开，表示待排序序列。

2. 实现归并排序、快速排序算法，输出排序好的序列，并记录算法运行时间。

3. 实现选择排序算法或插入排序算法，并将其运行时间与归并排序、快速排序算法比较，随机生成多个适当规模的数据进行实验并绘制折线图，反映不同算法运行时间随着输入规模的变化趋势，并与理论分析结果进行比较。

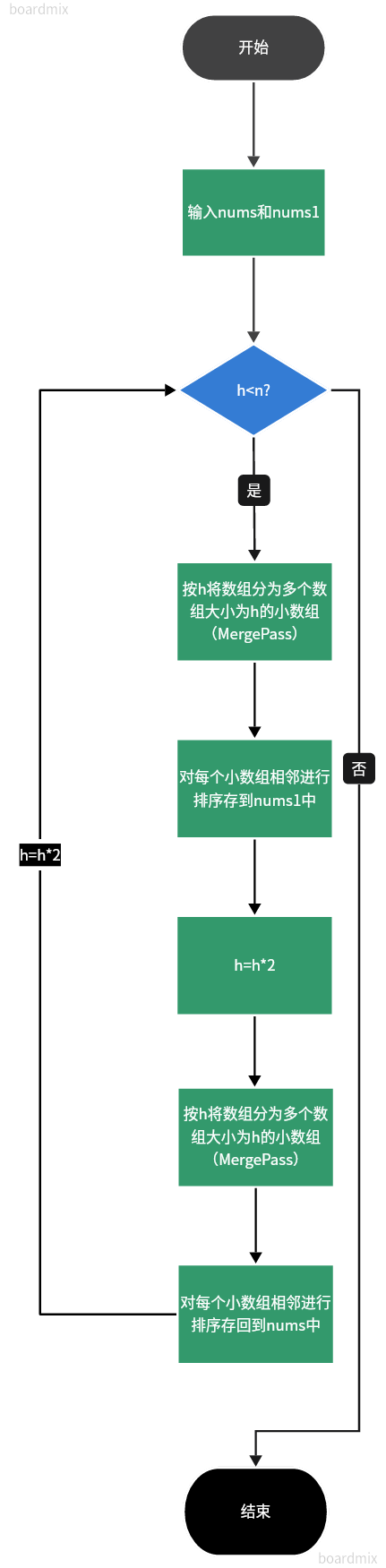
4. 选做: 实现堆排序算法（基于优先级队列）和基数排序算法，并将其运行时

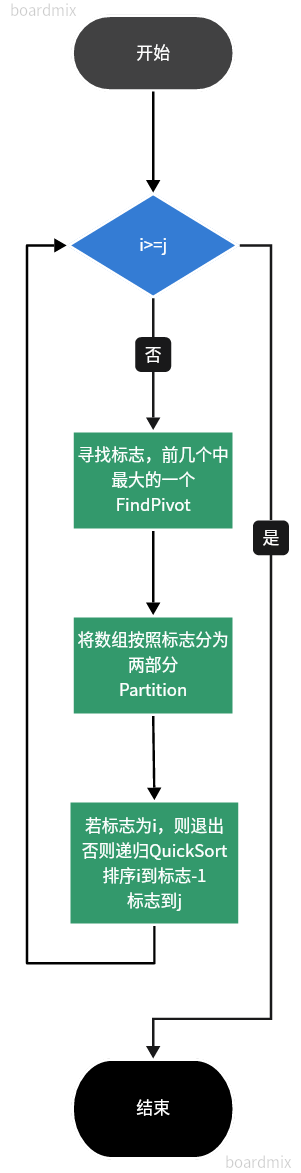
间与上述算法进行实验比较。

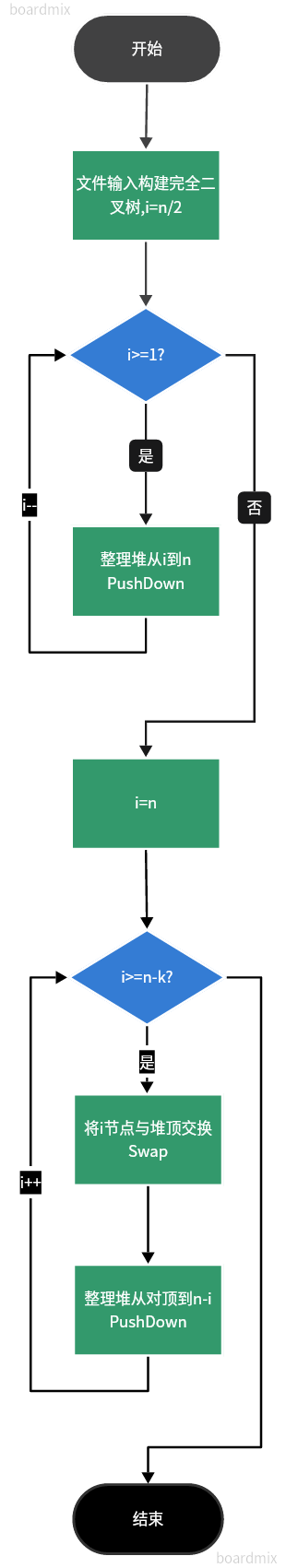
5. 选做: 对于序列 a1,a2,...an，若 ai>aj(1≤i<j≤n)则称(ai,aj)构成一个逆序对，请改写归并排序算法使其在对序列排序的同时计算原序列的逆序对数量并输出。

**三、设计思想**（本程序中的用到的所有数据类型的定义，各程序模块之间的调用关系、核心算法的流程图或主要步骤）

1．逻辑设计

归并排序：

快速排序：

堆排序：

2．物理设计

int frequency=0;//记录比较次数

void swap(int\* a, int\* b)//交换两个数

void shuffleArray(int disorderly[], int size)//产生随机数据

void GenerateRandom()//产生随机数据文件

int Getnums(int \*nums)//读取文件

void Swap(int\* nums,int x,int y) //交换数组中两个数

int FindPivot(int nums[],int i,int j) //寻找快排标志

int Partition(int i,int j,int \*nums) //将i到j部分分成两份

void QuickSort(int \*nums,int i,int j) //快排

void Show(int a[],int n) //显示排好的数组

void SelectSort(int\*nums,int n) //选择排序

void insertSort(int\*nums,int n) //插入法

void Merge(int\*nums,int \*nums1,int s,int m,int t) //将两个排好序的数组合并

void MergePass(int \*nums,int \*nums1,int n,int h) //合并排序

void MergeSort(int \*nums,int n,int \*nums1) //合并排序主程序

void PushDown(int\*h,int first,int last) //整理堆从first到last

int HeapSort(int\*h,int k) //堆排序

void ShowHeapSort(int\*h,int k,int n) //倒序显示堆排序

int nums[N]={0};//待排序数组

int nums1[N]={0};//合并排序辅助数组

int n;//待排序数组大小

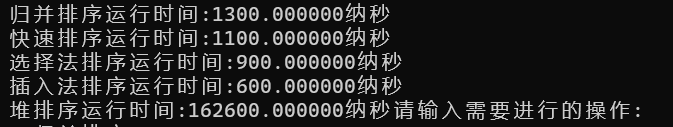
double duration=0;//记录运行时间

int k=0,h\_n=0;k://需要排序的数量；h\_n：堆大小

int h[N]={0};//堆

**四、测试结果**

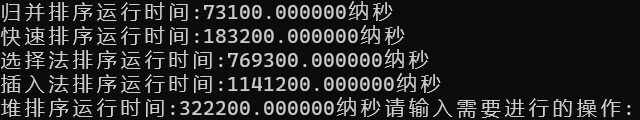
**数据量我10时：**

****

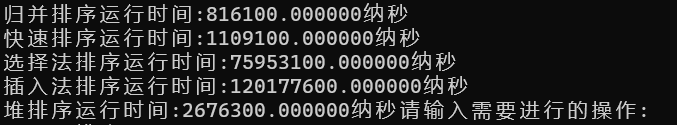
**数据量为100时：**

****

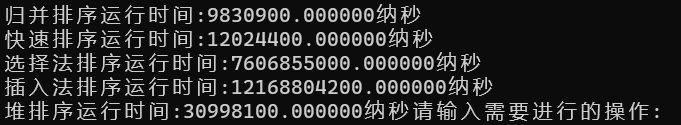
**数据量为1000时：**

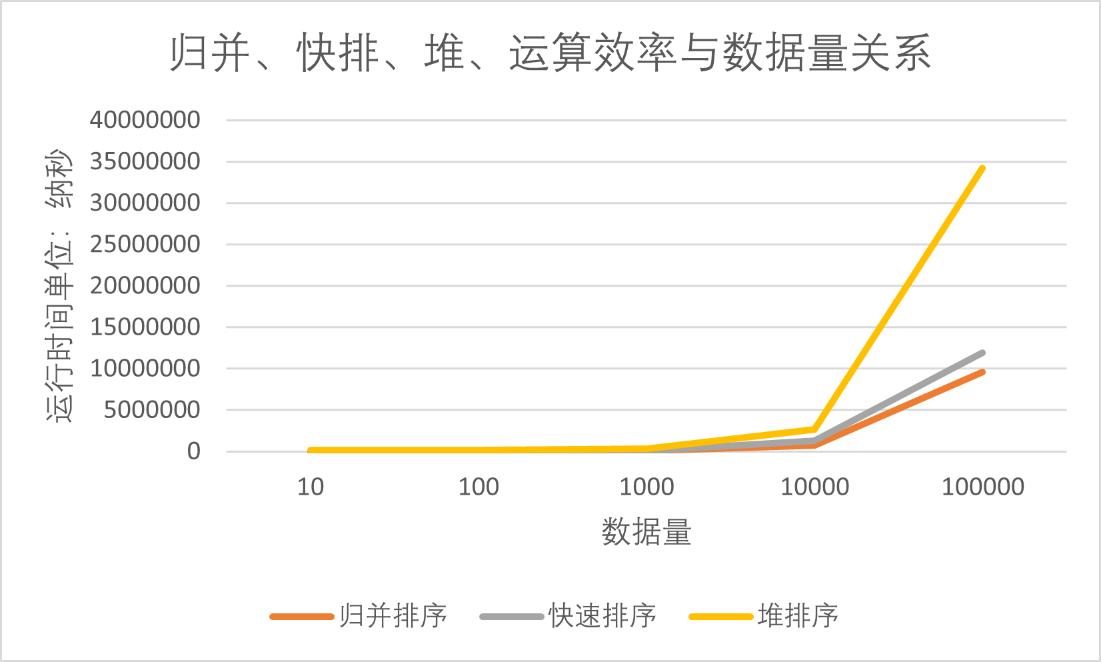
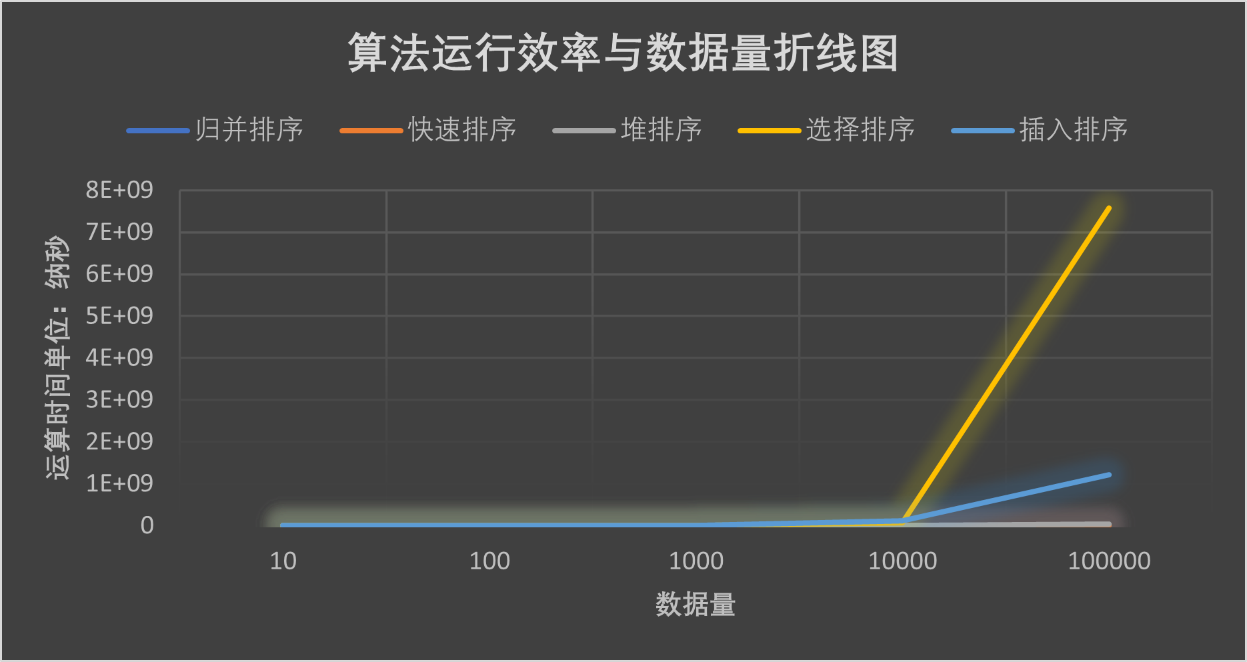
****

**数据量为10000时：**

****

**数据量为100000时：**

****



**分析：**数据量较少时运算时间相近，数据量较大时选择排序和插入排序运算时间明显较大，另外三种排序，堆排序时间相对更慢一些，快排与归并排序相差较小。

堆排序需要一开始将完全二叉树整理成堆并且每次将堆顶与最后元素交换后再整理为堆，蕴含了很多数量级较小的时间，导致时间大于归并排序和快排；

快排在寻找标志时可能会多耗一些时间，以及调用函数次数过多导致时间略大于归并排序；

归并排序两两个集合合并，原数组与辅助数组之间互相赋值，需要额外的内存；

**五、经验体会与不足**

经验体会：

各个排序各有特点，需要依照原理以及特点对排序方式进行选择，在排序时，细节是魔鬼，很可能因为一点点细节上的问题导致整个结果出错

不足：

快排由于运用了三个函数之间互相调用于是增加了运行时间，导致运行时间略大于归并排序，可以减少调用次数来减少运行时间

**六、附录：源代码（带注释）**

#include <stdio.h>

#include <stdlib.h>

#define N 100000

#include <time.h>

#define BILLION 1000000000L

struct timespec start, end;

long long elapsed;

int frequency=0;

void swap(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void shuffleArray(int disorderly[], int size)

{

srand(time(NULL)); // 初始化随机数生成器

for (int i = size - 1; i > 0; i--)

{

int j = rand() % (i + 1); // 生成一个随机索引

swap(&disorderly[i], &disorderly[j]); // 交换当前元素和随机索引对应的元素

}

}

void GenerateRandom()

{

int n=0;

int h[N]={0};

n=Getnums(h);

shuffleArray(h,n);

FILE\* fp=fopen("nums.txt","w");

fprintf(fp,"%d\n",n);

for(int i=0;i<n-1;i++)

{

fprintf(fp,"%d ",h[i]);

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

}fprintf(fp,"%d",h[n-1]);

fclose(fp);

}

int Getnums(int \*nums)

{

FILE\*fp;

int n=0;

fp=fopen("nums.txt","r");

fscanf(fp,"%d",&n);

fgetc(fp);

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

{

fscanf(fp,"%d",nums+i);fgetc(fp);

}

fclose(fp);

return n;

}

//交换数组中两个数

void Swap(int\* nums,int x,int y)

{

int t=nums[x];

nums[x]=nums[y];

nums[y]=t;

}

//寻找标志

int FindPivot(int nums[],int i,int j)

{

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

{

if(nums[k]>nums[k+1])

{

return nums[k];

}

else if(nums[k]<nums[k+1])

{

return nums[k+1];

}

}

return 0;

}

//将i到j部分分成两份

int Partition(int i,int j,int \*nums)

{

int p=FindPivot(nums,i,j);

int m=i,n=j;

do

{

while(nums[m]<p&&m<n)

{

m++;

frequency++;

}

while(nums[n]>=p&&n>m)

{

n--;

frequency++;

}

if(m<n)

{

Swap(nums,m,n);

}

}while(n>m);

return m;

}

//快排

void QuickSort(int \*nums,int i,int j)

{

if(i>=j)

{

return;

}

else

{

int cut=Partition(i,j,nums);

if(cut==i)return;

QuickSort(nums,i,cut-1);

QuickSort(nums,cut,j);

}

}

//显示排好的数组

void Show(int a[],int n)

{

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

{

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

}

printf("frequency=%d",frequency);

frequency=0;

}

//选择排序

void SelectSort(int\*nums,int n)

{

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

{

int min=i;

for(int j=i+1;j<n;j++)

{

frequency++;

if(nums[min]>nums[j])

{

min=j;

}

}

Swap(nums,min,i);

}

}

//插入法

void insertSort(int\*nums,int n)

{

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

{

int k=i;

for(int j=i-1;j>=0;j--)

{

frequency++;

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

{

Swap(nums,k,j);k--;

}

else

{

break;

}

}

}

}

//将两个排好序的数组合并

void Merge(int\*nums,int \*nums1,int s,int m,int t)

{

int i=s,j=m+1,k=s;

while(i<=m&&j<=t)

{

nums1[k++]=(nums[i]<nums[j])?nums[i++]:nums[j++];frequency++;

}

while(i<=m)

{

nums1[k++]=nums[i];i++;

}

while(j<=t)

{

nums1[k++]=nums[j];j++;

}

}

//合并排序

void MergePass(int \*nums,int \*nums1,int n,int h)

{

int i=0;

while(i+2\*h-1<n)

{

Merge(nums,nums1,i,i+h-1,i+2\*h-1);

i+=2\*h;

}

if(i+h-1<n)

{

Merge(nums,nums1,i,i+h-1,n-1);

}

else

{

for(int j=i;j<n;j++)

{

nums1[j]=nums[j];

}

}

}

//主程序

void MergeSort(int \*nums,int n,int \*nums1)

{

int h=1;

for(;h<n;h=h\*2)

{

MergePass(nums,nums1,n,h);h=h\*2;

MergePass(nums1,nums,n,h);

}

}

//整理堆从first到last

void PushDown(int\*h,int first,int last)/\*整理堆:把h[first]下推到完全二元树的适当位置\*/

{

int r=first; /\* r是被下推到的适当位置，初始值为根first\*/

while(r<=last/2)

{

frequency++;

if((r==last/2) && (last%2==0))/\* r有一个儿子在2\*r上且为左儿子\*/

{

if(h[r]>h[2\*r])

{

Swap(h,r,2\*r);/\*下推\*/

}

r=last;/\*交换后到叶，循环结束\*/

}

else if((h[r]>h[2\*r])&&(h[2\*r]<=h[2\*r+1]))/\*根大于左儿子，且左儿子小于或等于右儿子\*/

{

Swap(h,r,2\*r); /\*与左儿子交换\*/

r=2\*r;/\*下推到的位置也是下次考虑的根\*/

}

else if((h[r]>h[2\*r+1])&&(h[2\*r+1]<h[2\*r]))/\*根大于右儿子，且右儿子小于左儿子\*/

{

Swap(h,r,2\*r+1); /\*与右儿子交换\*/

r=2\*r+1;/\*下推到的位置也是下次考虑的根\*/

}

else/\*符合堆的定义，不必整理，循环结束\*/

{

r=last;

}

}

}

//堆排序

int HeapSort(int\*h,int k)

{

//文件输入

FILE\* fp=fopen("nums.txt","r");

int n=0;

if(fp==NULL)

{

printf("打开文件失败.");

}

else

{

fscanf(fp,"%d",&n);

n=n+1;

fgetc(fp);

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

{

fscanf(fp,"%d",h+i);fgetc(fp);

}

}

fclose(fp);

int i;

for( i=n/2; i>=1; i--) /\*初始建堆，从最右非叶结点开始\*/

{

PushDown(h,i, n); /\*整理堆，把以i为根，最大下标的叶为n\*/

}

//排序

for( i=n; i>n-k; i--)

{

Swap(h,i,1); //堆顶与当前堆中的下标最大的叶结点交换

PushDown(h,1, i-1 );/\*整理堆把以1为根，最大叶下标为i-1的完全二元树整理成堆\*/

}

return n;

}

//倒序显示堆排序

void ShowHeapSort(int\*h,int k,int n)

{

for(int i=n;i>n-k;i--)

{

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

}

printf("frequency=%d",frequency);

frequency=0;

}

int main()

{

int nums[N]={0};int nums1[N]={0};

int n=Getnums(nums);

double duration\_1=0,duration\_2=0,duration\_3=0,duration\_4=0,duration\_5=0;

int op=-1;

do{

printf("请输入需要进行的操作:\n0:归并排序\n1:快速排序\n2:选择法排序\n3:插入法排序\n4:堆排序;\n5:生成一组随机数据;\n6:运行完0到4后显示各排序运行时间;\n-1:退出");

scanf("%d",&op);

switch(op)

{

case -1:

break;

case 0:

clock\_gettime(CLOCK\_MONOTONIC, &start);

MergeSort(nums,n,nums1);

clock\_gettime(CLOCK\_MONOTONIC, &end);

duration\_1=BILLION \* (end.tv\_sec - start.tv\_sec) + end.tv\_nsec - start.tv\_nsec;

printf("运行时间:%f纳秒",duration\_1);

break;

case 1:

clock\_gettime(CLOCK\_MONOTONIC, &start);

QuickSort(nums,0,n-1);

clock\_gettime(CLOCK\_MONOTONIC, &end);

duration\_2=BILLION \* (end.tv\_sec - start.tv\_sec) + end.tv\_nsec - start.tv\_nsec;

Show(nums,n);

printf("运行时间:%f纳秒",duration\_2);

break;

case 2:

clock\_gettime(CLOCK\_MONOTONIC, &start);

SelectSort(nums,n);

clock\_gettime(CLOCK\_MONOTONIC, &end);

duration\_3=BILLION \* (end.tv\_sec - start.tv\_sec) + end.tv\_nsec - start.tv\_nsec;

Show(nums,n);

printf("运行时间:%f纳秒",duration\_3);

break;

case 3:

clock\_gettime(CLOCK\_MONOTONIC, &start);

insertSort(nums,n);

clock\_gettime(CLOCK\_MONOTONIC, &end);

duration\_4=BILLION \* (end.tv\_sec - start.tv\_sec) + end.tv\_nsec - start.tv\_nsec;

Show(nums,n);

printf("运行时间:%f纳秒",duration\_4);

break;

case 4:

{

int k=0,h\_n=0;

int h[N]={0};

printf("请输入K值:");

scanf("%d",&k);

clock\_gettime(CLOCK\_MONOTONIC, &start);

h\_n=HeapSort(h,k);

clock\_gettime(CLOCK\_MONOTONIC, &end);

duration\_5=BILLION \* (end.tv\_sec - start.tv\_sec) + end.tv\_nsec - start.tv\_nsec;

ShowHeapSort(h,k,h\_n);

printf("运行时间:%f纳秒",duration\_5);

}

break;

case 5:

GenerateRandom();

break;

case 6:

printf("归并排序运行时间:%f纳秒\n快速排序运行时间:%f纳秒\n选择法排序运行时间:%f纳秒\n插入法排序运行时间:%f纳秒\n堆排序运行时间:%f纳秒",duration\_1,duration\_2,duration\_3,duration\_4,duration\_5);

break;

}

n=Getnums(nums);

}while(op!=-1);

return 0;

}