《数据结构与算法实验》第 15 次实验

学院： 专业： 年级：
姓名： 学号： 日期： 2022 年 6 月 28 日

第一部分 验证实验

一、 实验目的

1. 掌握归并排序的基本思想与实现方法。
2. 掌握基数排序的基本思想与实现方法。

二、 实验内容

1. 归并排序算法的实现

- 对同一组数据进行归并排序，输出排序结果。
- 为了避免每次运行程序都从键盘上输入数据，可以设计一个函数自动生成待排序记录。

2. 基数排序算法的实现

- 对同一组数据进行基数排序，输出排序结果。
- 为了避免每次运行程序都从键盘上输入数据，可以设计一个函数自动生成待排序记录。

三、代码实现（注：实验编码格式为 UTF-8）

1. 归并排序算法的实现

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;
const int Max = 20;

void Creat(int r[], int n);
void Merge(int r[], int r1[], int s, int m, int t);
void MergeSort(int r[], int r1[], int n);
void MergePass(int r[], int r1[], int s, int t);
```

```
int main()
{
    int r[Max + 1] = {0}, r1[Max + 1] = {0};
    int r2[Max + 1] = {0};
    Creat(r, Max);
    cout<<"for unordered list: "<<endl;
    for (int i = 1; i <= Max ; i++)
        r2[i] = r[i];
    for (int i = 1; i <= Max ; i++)
        cout<<r[i]<<" ";
    cout<<endl;

    MergeSort(r2, r1, Max);
    cout<<"after perform MergeSort: "<<endl;
    for (int i = 1; i <= Max ; i++)
        cout<<r2[i]<<" ";
    cout<<endl;
    return 0;
}

void Creat(int r[], int n)
{
    int i = 0;
    srand(time(NULL));
    for (i = 1; i <= n; i++)
        r[i] = 1 + rand() % 100;
}

void Merge(int r[], int r1[], int s, int m, int t)
{
    int i = s, j = m + 1, k = s;
    while (i <= m && j <= t)
    {
        if (r[i] <= r[j])  r1[k++] = r[i++];
        else  r1[k++] = r[j++];
    }
    if (i <= m)
        while (i <= m)
            r1[k++] = r[i++];
    else
        while (j <= t)
            r1[k++] = r[j++];
}
```

```
void MergePass(int r[],int r1[],int n,int h)
{
    int i=1;
    while(i<=n-2*h+1)
    {
        Merge(r,r1,i,i+h-1,i+2*h-1);
        i+=2*h;
    }
    if(i<n-h+1)
        Merge(r,r1,i,i+h-1,n);
    else
        for(int k=i;k<=n;k++)
            r1[k]=r[k];
}


void MergeSort(int r[],int r1[],int n)
{
    int h=1;
    while(h<n)
    {
        MergePass(r,r1,n,h);
        h=2*h;
        MergePass(r1,r,n,h);
        h=2*h;
    }
}
```

2. 基数排序算法的实现

```
#include<iostream>
#include<ctime>
#include<cstdlib>
using namespace std;
const int Maxd = 5;
const int MaxRange = 10;
const int MaxNode = 20;

struct Node
{
    int key[Maxd];
    int next;
};
```

```
struct QueueNode
{
    int front;
    int rear;
};

void Creat(Node r[], int n,int m,int d);
void Print(Node r[], int n,int d,int first);
void Print1(Node r[], int n, int d);
void Distribute(Node r[], int n, QueueNode q[], int m, int first, int j);
void Collect(Node r[], int n, QueueNode q[], int m, int &first);
void RadixSort(Node r[], int n, int m, int d,int &first);

int main()
{
    Node r[MaxNode];
    cout<<"please enter the number of record, Maxsize = "<<MaxNode<<endl;
    int n;
    cin>>n;
    cout<<"please enter the number of subkey, Maxsize = "<<Maxd<<endl;
    int d;
    cin>>d;
    cout<<"please enter the limits of subkey (0~m-1), Maxsize = "<<MaxRange<<endl;
    int m;
    cin>>m;
    Creat(r, n, m,d);
    cout<<"origin subkey: "<<endl;
    Print1(r, n, d);
    cout<<endl;
    int first = 0;
    RadixSort(r, n, m, d,first);//必须把 first 带出来才能得到排序
    system("pause");
    return 0;
}


void Creat(Node r[], int n,int m,int d)
{
    srand(time(NULL));
    for(int i=0;i<n;i++)
        for (int j = 0; j < d; j++)
        {
            r[i].key[j] = rand() % m;
        }
```

```
}

void Print(Node r[], int n,int d,int first)
{
    int count = 0;
    while (count<n)
    {
        for (int j = 0; j < d; j++)
        {
            cout<<r[first].key[j]<<' ';
        }
        cout<<endl;
        first = r[first].next;
        count++;
    }
    cout<<endl;
}

void Print1(Node r[], int n, int d)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < d; j++)
        {
            cout<<r[i].key[j]<<' ';
        }
        cout<<endl;
    }

}

void Distribute(Node r[], int n, QueueNode q[], int m, int first, int j)
{
    int i = first;
    int count=0;
    int k;
    while (count<n)
    {
        k = r[i].key[j];
        if (q[k].front == -1)q[k].front = i;
        else r[q[k].rear].next = i;
        q[k].rear = i;
        i = r[i].next;
        count++;
```

```
    }
}

void Collect(Node r[], int n, QueueNode q[], int m, int &first)
{
    int k = 0;
    while (q[k].front == -1)
        k++;
    first = q[k].front;//带出头记录所在下标
    int last = q[k].rear;
    while (k<m)
    {
        k++;
        if (k >= m)break;//k 必须小于 m
        if (q[k].front != -1)
        {
            r[last].next = q[k].front;
            last = q[k].rear;
        }
    }
    r[last].next = -1;
}

void RadixSort(Node r[], int n, int m, int d,int &first)
{
    for (int i = 0; i < n; i++)
        r[i].next = i + 1;
    r[n - 1].next = -1;
    QueueNode q[MaxRange];
    for (int j = d - 1; j > -1; j--)
    {
        for (int i = 0; i < m; i++)//对每个子关键码分配前要清空桶
            q[i].front = q[i].rear = -1;
        Distribute(r, n, q, m, first, j);
        Collect(r, n, q, m, first);
        cout<<"after sorting for "<<d - j<<" times, r["<<n<<"]= "<<endl;
        Print(r, n, d,first);
    }
}
```

四、运行与测试

1. 归并排序算法的实现

```
for unordered list:
82 20 16 22 93 94 40 63 7 77 66 58 51 32 82 24 65 56 9 59
after perform MergeSort:
7 9 16 20 22 24 32 40 51 56 58 59 63 65 66 77 82 82 93 94
PS F:\DataStructure>
```

2. 基数排序算法的实现

```
please enter the number of record, Maxsize = 20
5
please enter the number of subkey, Maxsize = 5
2
please enter the limits of subkey (0~m-1), Maxsize = 10
5
origin subkey:
0 3
3 1
0 4
2 4
1 0

after sorting for 1 times, r[5]=
1 0
3 1
0 3
0 4
2 4

after sorting for 2 times, r[5]=
0 3
0 4
1 0
2 4
3 1

Press any key to continue . . .
```

第二部分 综合实验 1：各种排序算法的时间性能比较

一、问题描述

对本章的各种排序方法（直接插入排序、希尔排序、起泡排序、快速排序、直接选择排序、堆排序和归并排序）的时间性能进行比较。

二、基本要求

- 设计并实现上述各种排序算法。
- 产生正序和逆序的初始排列分别调用上述排序算法并比较时间性能。
- 产生随机的初始排列分别调用上述排序算法，并比较时间性能。

三、设计编码（注：实验编码格式为 UTF-8）

1. 算法设计

设计思想：

上述各种排序方法都是基于比较的内排序，其时间主要消耗在排序过程中进行的记录的比较和移动，因此，统计在相同数据状态下不同排序算法的比较次数和移动次数，即可实现比较各种排序算法的目的。各种排序算法在本章的验证实验中已经完成，为了实现比较不同排序算法的比较次数和移动次数，在算法中的适当位置插入两个计数器分别统计记录的比较次数和移动次数。

思考题：如果测算每种排序算法所用实际的时间， 应如何修改排序算法？ 我们可以多次生成随机数进行排序，计算排序相同次数的总时间。

2. 代码实现

(1)次数 Compare_fre.cpp

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <iomanip>
using namespace std;
const int Max = 10;
void Creat(int r[], int n);
```

```
void Creat1(int r[], int n);
void Creat2(int r[], int n);
void InsertSort(int r[], int n);
void ShellSort(int r[], int n);
void BubbleSort(int r[], int n);

int Partition(int r[], int first, int end,int& tcom,int &tmove);
void QuickSort(int r[], int first, int end,int& tcom,int& tmove);

void SelectSort(int r[], int n);

void Sift(int r[], int k, int m,int& tcom,int& tmove);
void HeapSort(int r[], int n,int& tcom,int& tmove);

void Merge(int r[], int r1[], int s, int m, int t,int& tcom,int& tmove);
void MergeSort2(int r[], int r1[], int s,int t,int& tcom,int& tmove);

void Copy(int a[],int b[])
{
    for (int i = 1; i <= Max; i++)
        b[i] = a[i];
}

void show(int a[])
{
    for (int i = 1; i <= Max; i++)
        cout<<a[i]<<"  ";
    cout<<endl;
}

void timecom(int a[])
{
    int b[Max + 1];

    //对与 a 相同的临时数组 b 进行排序
    Copy(a,b);
    cout<<"直接插入排序:   ";
    InsertSort(b, Max);
    Copy(a,b);
    cout<<"希尔排序:       ";
    ShellSort(b, Max);
    Copy(a,b);
    cout<<"起泡排序:       ";
    BubbleSort(b, Max);
```

```
    Copy(a,b);
    cout<<"快速排序:       ";
    int tcom=0,tmove=0;
    QuickSort(b,1,Max,tcom,tmove);
    cout<<"比较次数为"<<setw(3)<<tcom<<",移动次数为"<<setw(3)<<tmove<<endl;

    Copy(a,b);
    cout<<"直接选择排序:   ";
    SelectSort(b, Max);

    Copy(a,b);
    cout<<"堆排序(大根堆): ";
    tcom=0;tmove=0;
    HeapSort(b, Max,tcom,tmove);
    cout<<"比较次数为"<<setw(3)<<tcom<<",移动次数为"<<setw(3)<<tmove<<endl;

    Copy(a,b);
    cout<<"归并排序:       ";
    tcom=0;tmove=0;
    int r1[Max + 1];
    MergeSort2(b, r1, 1, Max,tcom,tmove);
    cout<<"比较次数为"<<setw(3)<<tcom<<",移动次数为"<<setw(3)<<tmove<<endl;
}

int main( )
{
    srand(time(NULL));
    int a[Max + 1] = {0}, b[Max + 1] = {0};
    cout<<"对于正序序列: ";
    Creat1(a, Max);
    show(a);
    timecom(a);
    cout<<endl<<"对于逆序序列: ";
    Creat2(a, Max);
    show(a);
    timecom(a);
    cout<<endl<<"对于随机序列: ";
    Creat(a, Max);
    show(a);
    timecom(a);
    return 0;
}
```

```
void Creat(int r[], int n)
{
    int i = 0;
    for (i = 1; i <= n; i++) //r[0]没用，可能用作哨兵
        r[i] = 1 + rand() % 100;
}

void Creat1(int r[], int n)
{
    int i = 0;
    r[0]=0;
    for (i = 1; i <= n; i++) //r[0]没用，可能用作哨兵
        r[i] = r[i-1] + rand() % 100;
}

void Creat2(int r[], int n)
{
    int i = 0;
    r[0]=1000;
    for (i = 1; i <= n; i++) //r[0]没用，可能用作哨兵
        r[i] = r[i-1] -10- rand() % 100;
}

void InsertSort(int r[ ], int n)   //0 号单元用作暂存单元和监视哨
{
    int tcom=0,tmove=0;
    for (int i = 2; i <= n; i++)
    {
        r[0]=r[i];  //暂存待插关键码，设置哨兵
        tmove++;
        int j;
        for ( j = i - 1; (++tcom)&&(r[0] < r[j]); j--)   //寻找插入位置
        {
            r[j + 1] = r[j];            //记录后移
            tmove++;
        }
        r[j + 1] = r[0];
        tmove++;
    }
    cout<<"比较次数为"<<setw(3)<<tcom<<",移动次数为"<<setw(3)<<tmove<<endl;
}

void ShellSort(int r[ ], int n)    //0 号单元用作暂存单元
{
```

```
    int tcom=0,tmove=0;
    for (int d = n/2; d >= 1; d = d / 2)   //以增量为 d 进行直接插入排序
    {
        for (int i = d + 1; i <= n; i++)
        {
            r[0] = r[i];    //暂存被插入记录
            ++tmove;
            int j ;
            for ( j = i - d; ++tcom&&j > 0 && r[0] < r[j]; j = j - d)
            {
                r[j + d] = r[j];         //记录后移 d 个位置
                ++tmove;
            }
            r[j + d] = r[0];
            ++tmove;
        }
    }
    cout<<"比较次数为"<<setw(3)<<tcom<<",移动次数为"<<setw(3)<<tmove<<endl;
}

void BubbleSort(int r[ ], int n)    //0 号单元用作交换操作的暂存单元
{
    int tcom=0,tmove=0;
    int exchange = n, bound = n;   //第一趟起泡排序的区间是[1, n]
    while (exchange != 0)        //当上一趟排序有记录交换时
    {
        bound = exchange; exchange = 0;
        for (int j = 1; j < bound; j++)     //一趟起泡排序，排序区间是[1, bound]
            if (++tcom&&r[j] > r[j+1])
            {
                r[0] = r[j]; r[j] = r[j + 1]; r[j + 1] = r[0];
                exchange = j;         //记载每一次记录交换的位置
                tmove+=3;
            }
    }
    cout<<"比较次数为"<<setw(3)<<tcom<<",移动次数为"<<setw(3)<<tmove<<endl;
}

int Partition(int r[ ], int first, int end,int& tcom,int& tmove)
{
    int i = first, j = end;         //初始化
    while (i < j)
    {
        while (i < j&&++tcom  && r[i] <= r[j]) j--;  //右侧扫描
```

```
        if (i < j)
        {
            r[0] = r[i]; r[i] = r[j]; r[j] = r[0];
            tmove+=3;
            i++;
        }
        while (i < j&&++tcom  && r[i] <= r[j]) i++;  //左侧扫描
        if (i < j)
        {
            r[0] = r[i]; r[i] = r[j]; r[j] = r[0];
            tmove+=3;
            j--;
        }
    }
    return i;    //i 为轴值记录的最终位置
}

void QuickSort(int r[ ], int first, int end,int &tcom,int &tmove)
{

    if (first < end)
    {      //区间长度大于 1，执行一次划分，否则递归结束
        int pivot=Partition(r, first, end,tcom,tmove);    //一次划分
        QuickSort(r, first, pivot - 1,tcom,tmove);     //递归地对左侧子序列进行快速排
序
        QuickSort(r, pivot + 1, end,tcom,tmove);     //递归地对右侧子序列进行快速排序
    }
}

void SelectSort(int r[ ], int n)   //0 号单元用作交换操作的暂存单元
{
    int tcom=0,tmove=0;

    for (int i = 1; i < n; i++)      //对 n 个记录进行 n-1 趟简单选择排序
    {
        int index = i;
        for (int j = i + 1; j <= n; j++)   //在无序区中选取最小记录
            if (++tcom && r[j] < r[index]) index = j;
        if (index != i)
        {
            r[0] = r[i]; r[i] = r[index]; r[index] = r[0];
            tmove+=3;
        }
    }
```

```
    cout<<"比较次数为"<<setw(3)<<tcom<<",移动次数为"<<setw(3)<<tmove<<endl;
}

void Sift(int r[ ], int k, int m,int& tcom,int& tmove)   //0 号单元用作交换操作的暂存单元
{
    int i = k, j = 2 * i;     //i 指向被筛选结点，j 指向结点 i 的左孩子
    while (j <= m)             //筛选还没有进行到叶子
    {
        if (++tcom && j < m && r[j] < r[j+1]) j++;  //比较 i 的左右孩子，j 指向较大者
        if (++tcom && r[i] > r[j]) break;         //根结点已经大于左右孩子中的较大者
        else
        {
            r[0] = r[i]; r[i] = r[j]; r[j] = r[0];   //将根结点与结点 j 交换
            tmove+=3;
            i = j; j = 2 * i;           //被筛结点位于原来结点 j 的位置
        }
    }
}

void HeapSort(int r[ ], int n,int& tcom,int& tmove)      //0 号单元用作交换操作的暂存单元
{

    int i = 0;
    for (i = n/2; i >= 1; i--)       //初始建堆，从最后一个分支结点至根结点
        Sift(r, i, n,tcom,tmove);
    for (i=1; i<n; i++)         //重复执行移走堆顶及重建堆的操作
    {
        r[0] = r[1]; r[1] = r[n - i + 1]; r[n - i + 1] = r[0];
        tmove+=3;
        Sift(r, 1, n-i,tcom,tmove);
    }
}

void Merge(int r[], int r1[], int s, int m, int t,int& tcom,int& tmove)
{
    int i = s, j = m + 1, k = s;
    while (i <= m && j <= t)
    {
        if (++tcom && r[i] <= r[j])
            r1[k++] = r[i++];
        else r1[k++] = r[j++];
        tmove++;
```

```
    }
    if (i <= m)
        while (i <= m)
        {
            r1[k++] = r[i++];
            tmove++;
        }
    else
        while (j <= t)
        {
            r1[k++] = r[j++];
            tmove++;
        }
}

void MergeSort2(int r[], int r1[], int s,int t,int& tcom,int& tmove)
{
    if (s != t)
    {
        int m = (s + t) / 2;
        MergeSort2(r, r1, s, m,tcom,tmove);
        MergeSort2(r, r1, m + 1, t,tcom,tmove);
        for (int i = s; i <= t; i++)
        {
            r1[i] = r[i];
            tmove++;
        }
        Merge(r1, r, s, m, t,tcom,tmove);
    }
}
```

(2)时间 Compare_time.cpp

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <iomanip>
using namespace std;
const int Max = 10000;

void Creat(int r[], int n);
void Creat1(int r[], int n);
void Creat2(int r[], int n);
void InsertSort(int r[], int n);
```

```
void ShellSort(int r[], int n);
void BubbleSort(int r[], int n);

int Partition(int r[], int first, int end);
void QuickSort(int r[], int first, int end);

void SelectSort(int r[], int n);

void Sift(int r[], int k, int m);
void HeapSort(int r[], int n);

void Merge(int r[], int r1[], int s, int m, int t);
void MergeSort2(int r[], int r1[], int s,int t);

void Copy(int a[],int b[])
{
    for (int i = 1; i <= Max; i++)
        b[i] = a[i];
}

void show(int a[])
{
    for (int i = 1; i <= Max; i++)
        cout<<a[i]<<"  ";
    cout<<endl;
}

void timecom()
{
    int k=50,A[Max+1],b[Max+1];
    cout<<"对"<<k<<"组数排序，每组数有"<<Max<<"个"<<endl;
    clock_t start_t, end_t;
    double total_t=0;
    for(int i=1;i<=k;i++)
    {
        Creat(b,Max);
        start_t = clock();
        InsertSort(b,Max);
        end_t = clock();
        total_t=total_t-start_t+end_t;
    }
    total_t = (double) total_t / CLOCKS_PER_SEC;
    cout<<"直接插入排序:   "<<setw(3)<<total_t<<"秒"<<endl;
```

```
    for(int i=1;i<=k;i++)
    {
        Creat(b,Max);
        start_t = clock();
        ShellSort(b, Max);
        end_t = clock();
        total_t=total_t-start_t+end_t;
    }
    total_t = (double) total_t / CLOCKS_PER_SEC;
    cout<<"希尔排序:       "<<setw(3)<<total_t<<"秒"<<endl;

    for(int i=1;i<=k;i++)
    {
        Creat(b,Max);
        start_t = clock();
        BubbleSort(b, Max);
        end_t = clock();
        total_t=total_t-start_t+end_t;
    }
    total_t = (double) total_t / CLOCKS_PER_SEC;
    cout<<"起泡排序:       "<<setw(3)<<total_t<<"秒"<<endl;

    for(int i=1;i<=k;i++)
    {
        Creat(b,Max);
        start_t = clock();
        QuickSort(b,1,Max);
        end_t = clock();
        total_t=total_t-start_t+end_t;
    }
    total_t = (double) total_t / CLOCKS_PER_SEC;
    cout<<"快速排序:       "<<setw(3)<<total_t<<"秒"<<endl;

    for(int i=1;i<=k;i++)
    {
        Creat(b,Max);
        start_t = clock();
        SelectSort(b, Max);
        end_t = clock();
        total_t=total_t-start_t+end_t;
    }
    total_t = (double) total_t / CLOCKS_PER_SEC;
    cout<<"直接选择排序:   "<<setw(3)<<total_t<<"秒"<<endl;
```

```
    for(int i=1;i<=k;i++)
    {
        Creat(b,Max);
        start_t = clock();
        HeapSort(b, Max);
        end_t = clock();
        total_t=total_t-start_t+end_t;
    }
    total_t = (double) total_t / CLOCKS_PER_SEC;
    cout<<"堆排序(大根堆)：  "<<setw(3)<<total_t<<"秒"<<endl;

    for(int i=1;i<=k;i++)
    {
        Creat(b,Max);
        start_t = clock();
        MergeSort2(b, A, 1, Max);
        end_t = clock();
        total_t=total_t-start_t+end_t;
    }
    total_t = (double) total_t / CLOCKS_PER_SEC;
    cout<<"归并排序：        "<<setw(3)<<total_t<<"秒"<<endl;
}

int main( )
{
    srand(time(NULL));
    timecom();
    return 0;
}

void Creat(int r[], int n)
{
    int i = 0;
    for (i = 1; i <= n; i++)        //r[0]可能用作哨兵
        r[i] = 1 + rand() % 100;
}

void InsertSort(int r[], int n)    //0 号单元用作暂存单元和监视哨
{
    for (int i = 2; i <= n; i++)
    {
        r[0]=r[i];  //暂存待插关键码，设置哨兵
        int j;
        for ( j = i - 1; (r[0] < r[j]); j--)   //寻找插入位置
```

```
        {
            r[j + 1] = r[j];            //记录后移
        }
        r[j + 1] = r[0];
    }
}

void ShellSort(int r[], int n)    //0 号单元用作暂存单元
{
    for (int d = n/2; d >= 1; d = d / 2)   //以增量为 d 进行直接插入排序
    {
        for (int i = d + 1; i <= n; i++)
        {
            r[0] = r[i];    //暂存被插入记录
            int j ;
            for ( j = i - d; j > 0 && r[0] < r[j]; j = j - d)
            {
                r[j + d] = r[j];         //记录后移 d 个位置
            }
            r[j + d] = r[0];
        }
    }
}

void BubbleSort(int r[], int n)    //0 号单元用作交换操作的暂存单元
{
    int exchange = n, bound = n;   //第一趟起泡排序的区间是[1, n]
    while (exchange != 0)        //当上一趟排序有记录交换时
    {
        bound = exchange; exchange = 0;
        for (int j = 1; j < bound; j++)     //一趟起泡排序，排序区间是[1, bound]
            if (r[j] > r[j+1])
            {
                r[0] = r[j]; r[j] = r[j + 1]; r[j + 1] = r[0];
                exchange = j;         //记载每一次记录交换的位置
            }
    }
}

int Partition(int r[], int first, int end)
{
    int i = first, j = end;         //初始化
    while (i < j)
    {
```

```
        while (i < j && r[i] <= r[j]) j--;  //右侧扫描
        if (i < j)
        {
            r[0] = r[i]; r[i] = r[j]; r[j] = r[0];
            i++;
        }
        while ( i < j && r[i] <= r[j]) i++;  //左侧扫描
        if (i < j)
        {
            r[0] = r[i]; r[i] = r[j]; r[j] = r[0];
            j--;
        }
    }
    return i;    //i 为轴值记录的最终位置
}

void QuickSort(int r[], int first, int end)
{

    if (first < end)
    {      //区间长度大于 1，执行一次划分，否则递归结束
        int pivot=Partition(r, first, end);    //一次划分
        QuickSort(r, first, pivot - 1);     //递归地对左侧子序列进行快速排序
        QuickSort(r, pivot + 1, end);     //递归地对右侧子序列进行快速排序
    }
}

void SelectSort(int r[], int n)   //0 号单元用作交换操作的暂存单元
{

    for (int i = 1; i < n; i++)      //对 n 个记录进行 n-1 趟简单选择排序
    {
        int index = i;
        for (int j = i + 1; j <= n; j++)   //在无序区中选取最小记录
            if ( r[j] < r[index]) index = j;
        if (index != i)
        {
            r[0] = r[i]; r[i] = r[index]; r[index] = r[0];
        }
    }
}

void Sift(int r[], int k, int m)   //0 号单元用作交换操作的暂存单元
{
```

```
    int i = k, j = 2 * i;     //i 指向被筛选结点，j 指向结点 i 的左孩子
    while (j <= m)             //筛选还没有进行到叶子
    {
        if (  j < m && r[j] < r[j+1]) j++;  //比较 i 的左右孩子，j 指向较大者
        if ( r[i] > r[j]) break;         //根结点已经大于左右孩子中的较大者
        else
        {
            r[0] = r[i]; r[i] = r[j]; r[j] = r[0];   //将根结点与结点 j 交换
            i = j; j = 2 * i;           //被筛结点位于原来结点 j 的位置
        }
    }
}

void HeapSort(int r[], int n)      //0 号单元用作交换操作的暂存单元
{

    int i = 0;
    for (i = n/2; i >= 1; i--)       //初始建堆，从最后一个分支结点至根结点
        Sift(r, i, n);
    for (i=1; i<n; i++)         //重复执行移走堆顶及重建堆的操作
    {
        r[0] = r[1]; r[1] = r[n - i + 1]; r[n - i + 1] = r[0];
        Sift(r, 1, n-i);
    }
}

void Merge(int r[], int r1[], int s, int m, int t)
{
    int i = s, j = m + 1, k = s;
    while (i <= m && j <= t)
    {
        if (r[i] <= r[j])
            r1[k++] = r[i++];
        else r1[k++] = r[j++];
    }
    if (i <= m)
        while (i <= m)
            r1[k++] = r[i++];
    else
        while (j <= t)
            r1[k++] = r[j++];
}

void MergeSort2(int r[], int r1[], int s,int t)
```

```
{
    if (s != t)
    {
        int m = (s + t) / 2;
        MergeSort2(r, r1, s, m);
        MergeSort2(r, r1, m + 1, t);
        for (int i = s; i <= t; i++)
        {
            r1[i] = r[i];
        }
        Merge(r1, r, s, m, t);
    }
}
```

四、运行结果

1. 次数比较

```
对于正序序列：17  24  110  134  229  280  300  323  392  447
直接插入排序：   比较次数为  9,移动次数为 18
希尔排序：       比较次数为 22,移动次数为 44
起泡排序：       比较次数为  9,移动次数为  0
快速排序：       比较次数为 45,移动次数为  0
直接选择排序：   比较次数为 45,移动次数为  0
堆排序(大根堆)：比较次数为 46,移动次数为 90
归并排序：       比较次数为 19,移动次数为 68

对于逆序序列：945  907  841  808  732  710  630  591  557  482
直接插入排序：   比较次数为 54,移动次数为 63
希尔排序：       比较次数为 35,移动次数为 57
起泡排序：       比较次数为 45,移动次数为135
快速排序：       比较次数为 45,移动次数为 15
直接选择排序：   比较次数为 45,移动次数为 15
堆排序(大根堆)：比较次数为 38,移动次数为 63
归并排序：       比较次数为 15,移动次数为 68

对于随机序列：76  90  2  78  56  39  20  63  86  15
直接插入排序：   比较次数为 36,移动次数为 45
希尔排序：       比较次数为 39,移动次数为 61
起泡排序：       比较次数为 45,移动次数为 81
快速排序：       比较次数为 22,移动次数为 27
直接选择排序：   比较次数为 45,移动次数为 15
堆排序(大根堆)：比较次数为 44,移动次数为 87
归并排序：       比较次数为 24,移动次数为 68

--------------------------------
Process exited after 0.2852 seconds with return value 0
请按任意键继续. . .
```

另：由于本次代码只能在 dev c++中正常运行，VSCode 的环境不支持代码运行，下次可以尝试重新配置 VSCode 环境。

2.时间比较

```
对50组数排序，每组数有10000个
直接插入排序：   2.886秒
希尔排序：       0.071886秒
起泡排序：       8.77007秒
快速排序：       0.10077秒
直接选择排序：   5.6431秒
堆排序(大根堆)：0.0696431秒
归并排序：       0.0550696秒
```

第三部分 综合实验 2：机器调度问题

一、问题描述

机器调度是指有 m 台机器需要处理 n 个作业，设作业 i 的处理时间为 t_i，则对 n 个作业进行机器分配，使得：

(1)一台机器在同一时间内只能处理一个作业；

(2)一个作业不能同时在两台机器上处理；

(3)作业 i 一旦运行，则需要 t_i 个连续时间单位。

设计算法进行合理调度，使得在 m 台机器上处理 n 个作业所需要的处理时间最短。

二、基本要求

- 建立问题模型，设计数据结构。
- 设计调度算法，为每个作业分配一台可用机器。
- 给出外配方案。

三、设计编码（注：实验编码格式为 UTF-8）

1.算法设计

设计思想：

对于机器调度问题的处理，我们应该采用最长时间优先(LPT)的简单调度策略。在 LPT 算法中，作业按其所需时间的递减顺序排列，在外配一个作业时，将其外配给最先变为空闲的机器。

2. 代码实现

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include <stdio.h>
#include <time.h>
using namespace std;
const int Max = 100;

struct MachineNode
{
    int ID;
    int avail;
}M[Max+1];

struct JobNode
{
    int ID;
    int time;
}J[Max+1];

void Sift(JobNode r[], int k, int m)//大根堆的建立
{
    int i = k,j = 2*i;
    while(j <= m)
    {
        if( (j < m) && (r[j].time < r[j+1].time))
            j++;
        if( (r[i].time) > (r[j].time))
            break;
        else
        {
            JobNode t = r[i];
            r[i] = r[j];
            r[j] = t;
            i = j;
            j = 2*i;
        }
    }
}

void Sift2(MachineNode r[], int k, int m)//小根堆的建立
{
    int i = k, j = 2*i;
```

```
    while(j <= m)
    {
        if((j < m) && (r[j].avail > r[j+1].avail))
            j++;
        if((r[i].avail) < (r[j].avail))
            break;
        else
        {
            MachineNode t = r[i];
            r[i] = r[j];
            r[j] = t;
            i = j;
            j = 2*i;
        }
    }
}

void LPT(int n, int m)
{
    if(n <= m)
    {
        int maxtime = J[1].time;
        for(int i = 1; i <= m; i++)
        {
            cout<<"把第"<<i<<"个作业分配给机器"<<i<<",用时"<<J[i].time<<endl;
            if( J[i].time > maxtime)
                maxtime = J[i].time;
        }
        cout<<"最短调度长度为"<<maxtime<<endl;
        return ;
    }
    else
    {
        //n个作业的大根堆 建立
        int i = 0;
        for(i = n/2; i >= 1; i--)
            Sift(J, i, n);
        i = 1;
        int maxtime = 0;
        while(i <= n)
        {
            //机器建立小根堆
            Sift2(M, 1, m);
            int t1 = J[1].time;
```

```
            int n1 = J[1].ID;
            int n2 = M[1].ID;
            M[1].avail += t1;
            int t2 = M[1].avail;
            cout<<"把第"<<n1<<"个作业分配给机器"<<n2<<",用时"<<t1<<",完成时刻为
"<<t2<<endl;
            //H1 的堆顶作业取走,再建立大根堆
            J[1] = J[n - i + 1];
            Sift(J, 1, n-i);
            i++;
        }
        cout<<"最短调度长度为"<<M[1].avail<<endl;
        return ;
    }
}

int main()
{
    int m, n, i;
    cout<<"请输入机器台数 m"<<endl;
    cin>>m;
    for(int i = 1; i <= m; i++)
    {
        M[i].ID = i;
        M[i].avail = 0;
    }
    cout<<"请输入作业总数 n 和每个作业所需时间"<<endl;
    cin>>n;
    cout<<"请输入每个作业所需时间"<<endl;
    for(int i = 1; i <= n; i++)
    {
        cin>>J[i].time;
        J[i].ID = i;
    }
    LPT(n, m);
    return 0;
}
```

四、运行与测试

```
请输入机器台数m
7
请输入作业总数n和每个作业所需时间
7
请输入每个作业所需时间
2 14 4 16 6 5 3
把第1个作业分配给机器1,用时2
把第2个作业分配给机器2,用时14
把第3个作业分配给机器3,用时4
把第4个作业分配给机器4,用时16
把第5个作业分配给机器5,用时6
把第6个作业分配给机器6,用时5
把第7个作业分配给机器7,用时3
最短调度长度为16
PS F:\DataStructure>
```

```
请输入机器台数m
3
请输入作业总数n和每个作业所需时间
7
请输入每个作业所需时间
2 14 4 16 6 5 3
把第4个作业分配给机器2,用时16,完成时刻为16
把第2个作业分配给机器1,用时14,完成时刻为14
把第5个作业分配给机器3,用时6,完成时刻为6
把第6个作业分配给机器3,用时5,完成时刻为11
把第3个作业分配给机器3,用时4,完成时刻为15
把第7个作业分配给机器1,用时3,完成时刻为17
把第1个作业分配给机器3,用时2,完成时刻为17
最短调度长度为17
PS F:\DataStructure>
```

五、总结与心得

最后一个综合实验主要是考察如何正确设计数据结构（用结构体储存机器和作业的 ID）。其次是根据算法的核心要求建立相应的堆。大、小根堆的调整算法与是书上的桶排序算法的调整方式相同。。

另外，最后一个综合实验的实现过程中本来想使用 txt 文档进行输入，但是没有成功，对用文档输入和修改的方式还不太熟悉。