排序

```c
#define _CRT_SECURE_NO_WARNINGS 1
#pragma once
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

void arrPrint(int a[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    printf("\n");
}

//升序
void InsertRange(int a[], int n)
{

    //摸牌
    for (int i = 0; i < n - 1; i++)
    {
        //找位置
        int end = i;
        int temp = a[end + 1];
        while (end >= 0)
        {
            if (a[end] > temp)
            {
                a[end + 1] = a[end];//把打牌往后面抓
                end--;//向后移动
            }
            else {
                break;
            }
        }
        //放牌
        a[end + 1] = temp;
```

```
        arrPrint(a, n);
    }

}
void Test1()
{
    int a[] = { 4 ,5, 7, 6, 3, 2, 1 };
    //arrPrint(a, sizeof(a) / sizeof(a[0]));
    InsertRange(a, sizeof(a) / sizeof(a[0]));
    //arrPrint(a, sizeof(a) / sizeof(a[0]));
}

//希尔排序
void ShellSort(int a[], int n)
{
    int gap=n;
    //当gap==1时就结束，即做完最后的直接插入排序结束
    while (gap > 1) {
        gap = gap/3+1;
        for (int i = 0; i < n - gap; i++)
        {
            int end = i;
            while (end >= 0) {
                int temp = a[end + gap];

                if (a[end] > temp)
                {
                    a[end + gap] = a[end];
                    end -= gap;
                }
                else
                {
                    break;
                }

                a[end + gap] = temp;
            }
        }
    }

}
void Test2()
{
```

```
    int a[] = { 4 ,5, 7, 6, 3, 2, 1 };
    arrPrint(a, sizeof(a) / sizeof(a[0]));
    ShellSort(a, sizeof(a) / sizeof(a[0]));
    arrPrint(a, sizeof(a) / sizeof(a[0]));
}

//堆排序
void Swap(int* p1, int* p2)
{
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}

void AdjustDown(int a[], int size, int root)
{
    int parent = root;
    int child = 2*parent + 1;//默认为左child
    while (child < size)
    {
        //找左右孩子大的那一个
        //child+1<size是防止没有→孩子
        if (child + 1 < size && a[child] < a[child + 1])
        {
            child += 1;
        }
        if (a[child] > a[parent])
        {
            Swap(a + child, a + parent);
            parent = child;
            child = parent * 2 + 1;
        }
        else
        {
            break;
        }
    }
}

//堆排序
void HeapSort(int a[], int size)
{
    //建堆
```

```
    for (int i = (size - 1 - 1) / 2; i >= 0; i--)
    {
        AdjustDown(a, size, i);
    }
    arrPrint(a, size);
    //排序
    //将第一个数和最后一个数(end)对换，然后再对前end个数以0为root进行向下调整

    int end = size - 1;
    while (end>0)
    {
        Swap(&a[0], &a[end]);
        AdjustDown(a, end, 0);
        end--;
    }
    arrPrint(a, size);
}

void Test3()
{
    int a[] = { 1,5,6,3,2,4,8,4,5,6 };
    HeapSort(a, sizeof(a) / sizeof(a[0]));
}
//直接选择排序
void SelectSort(int a[], int size)
{
    int begin = 0, end = size - 1;
    while (begin < end)
    {
        int mini = begin;
        int maxi = end;
        for (int i = begin; i < end; i++)
        {
            if (a[i] < a[mini])
            {
                mini = i;
            }
            if (a[i] > a[maxi])
            {
                maxi = i;
            }
        }
```

```
        Swap(a + begin, a + mini);
        if (begin == maxi)
        {
            maxi = mini;
        }
        Swap(a + end, a + maxi);
        end--;
        begin++;
    }
}
void Test4()
{
    int a[] = { 9,5,6,3,2,4,8,4,5,6 };
    SelectSort(a, sizeof(a) / sizeof(a[0]));
    arrPrint(a, sizeof(a) / sizeof(a[0]));
}

int PartSort1(int a[], int left, int right)
{
    int begin = left, end = right;
    int pivot = begin;
    int key = a[begin];
    while (begin < end)
    {
        //右边找小
        while (begin < end && a[end] >= key)
        {
            end--;
        }
        //填坑
        a[pivot] = a[end];
        pivot = end;
        //左边找大
        while (begin < end && a[begin] <= key)
        {
            begin++;
        }
        //放坑
        a[pivot] = a[begin];
        pivot = begin;
    }
    //填坑
    pivot = begin;
```

```
    a[pivot] = key;
    return pivot;
}

void QuickSort(int a[], int left,int right)
{
    if (left >= right)
    {
        return;
    }
    int keyindex = PartSort1(a, left, right);
    //递归左右区间
    QuickSort(a, left, keyindex - 1);
    QuickSort(a, keyindex + 1, right);;
}

void Test5()
{
    int a[] = { 9,5,6,3,2,4,8,4,5,6 };
    QuickSort(a, 0,sizeof(a) / sizeof(a[0])-1);
    arrPrint(a, sizeof(a) / sizeof(a[0]));
}

void _MergeSort(int a[], int left, int right,int *temp)
{
    //结束递归条件
    if (left >= right)return;

    int mid = (left + right) >> 1;

    //左右排序
    _MergeSort(a, left, mid, temp);
    _MergeSort(a, mid+1, right, temp);

    //归并
    int end1 = mid, begin1 = left;
    int begin2 = mid+1, end2 = right;
    int index = left;
    while(begin1 <= end1 && begin2 <= end2)
    {
        if (a[begin1] < a[begin2])
        {
            temp[index++] = a[begin1++];
```

```
        }
        else {
            temp[index++] = a[begin2++];
        }
    }
    while (begin1 <= end1)
    {
        temp[index++] = a[begin1++];
    }
    while (begin2 <= end2)
    {
        temp[index++] = a[begin2++];
    }
    //拷贝回去
    for (int i = left; i <= right; i++)
    {
        a[i] = temp[i];
    }


}
void MergeSort(int a[], int n)
{
    int* temp = (int*)malloc(sizeof(int) * n);
    _MergeSort(a, 0, n - 1, temp);
    free(temp);
}

//直接并
void MergeSort2(int a[], int n)
{
    int* temp = (int*)malloc(sizeof(int) * n);
    int gap = 1;
    while (gap < n)
    {
        for (int i = 0; i < n; i += 2 * gap)
        {
            //归并
            //[i,i+gap-1][i+gap,i+2*gap-1]
            int end1 = i+gap-1, begin1 = i;
            int begin2 = i+gap, end2 = i+2*gap-1;
            int index = begin1;
            //当左边区间数量小于gap时
```

```
        //  if (end1 > n - 1)break;
            //当右边区间不存在时
            if (begin2 > n - 1)break;
                //当右边区间不足时
            if (end2 > n - 1)end2 = n - 1;

            while (begin1 <= end1 && begin2 <= end2)
            {
                if (a[begin1] < a[begin2])
                {
                    temp[index++] = a[begin1++];
                }
                else {
                    temp[index++] = a[begin2++];
                }
            }
            while (begin1 <= end1)
            {
                temp[index++] = a[begin1++];
            }
            while (begin2 <= end2)
            {
                temp[index++] = a[begin2++];
            }
            //拷贝回去
            for (int i = begin1; i <= end2; i++)
            {
                a[i] = temp[i];
            }
        }
        gap *= 2;
    }

    free(temp);

}
void Test6()
{
    int a[] = { 9,5,6,3,2,4,8,4,5,6 };
    MergeSort(a, sizeof(a) / sizeof(a[0]) );
    arrPrint(a, sizeof(a) / sizeof(a[0]));
}
```

```c
int main()
{
    //Test1();
    //Test2();
    /*Test3();
    Test4();
    Test5();
    Test6();*/
    Test6();
    return 0;
}
```

@title: 排序
@date: 2025-01-08 19:00:00
@version: 1.0.0

@author：软件工程宋浩元