Lab 5

Task 1 - 插入排序 Insertion Sort

完成排序算法以及部分辅助函数

Task 1.1 - 随机数组

> 生成用来排序的随机数组

给定随机数种子： `srand((unsigned) time(NULL)); //用时间做种，每次产生随机数不一样` 。

- 设计一个子函数，功能是产生长度为dim，数值在 [1-range] 区间的随机数。
- 函数形式：`int randnum(int *arr, int dim, int range)`。

```
// #define dim          //生成多大的数组
// #define range        //数组的元素大小从1到range
//! 产生长度为dim，数值在［1-range］区间的随机数组
void randnum(int *arr, int dim, int range)
{
    srand((unsigned) time(NULL));   //用时间做种，每次产生随机数不一样
    // TODO
}
```

Task 1.2 - 数组打印输出

> 打印数组

调用上节课的数组输出函数：`void ivec_print_pl(int *x, int n)`。

```
void ivec_print_pl(int *x, int n)
{
    int i;
    // TODO
    printf("\n");
}
```

Task 1.3 - 排序算法

> 实现插入排序

算法：

- 将第一待排序序列第一个元素看做一个有序序列，把第二个元素到最后一个元素当成是未排序序列。
- 从头到尾依次扫描未排序序列，将扫描到的每个元素插入有序序列的适当位置。（如果待插入的元素与有序序列中的某个元素相等，则将待插入元素插入到相等元素的后面。）

```
void insertion_sort(int arr[], int len)
{
    int i, j, key;
    for (i = 1; i < len; i++)
    {
        key = arr[i];
        j = i - 1;
        // TODO
        arr[j + 1] = key;
    }
}
```

Task 1.4 - 测量计算时间

> 输出排序时间

编写主函数，实现给定数组维度dim，范围[1,range]的整数排序，输出插入排序的排序时间。

```
int main()
{
    int dim = 10;
    int range = 8;
    clock_t begin, end; //! 计时
    int arr[dim];

    randnum(arr, dim, range);

    ivec_print_p1(arr, dim);

    // TODO
    insertion_sort(arr, dim);
    ivec_print_p1(arr, dim);

    double cost = end - begin;
    printf("dim=%d:\t", dim);
    printf("time=%f\n", cost);

    return 0;
}
```

Task 1.5 - 多次测量取平均值

重复100次，输出维度dim数组插入排序的平均时间t_avg，最大时间t_max。

```
int main()
{
    int origtable[10000]; // 待排数组
    int NTable = 10000;   // 元素个数
    double timeexe[100];  // 每次随机试验的时间
    int NTest = 100;      // 试验次数
    for (int k = 0; k < NTest; k++)
    {
        randnum(origtable, NTable, 10000); // 生成随机数数组
        // ivec_print_p1(origtable,NTable);
        clock_t begin, end;
        begin = clock();
        insertion_sort(origtable, NTable);
        end = clock();
        // ivec_print_p1(origtable,NTable);
        timeexe[k] = (double)(end - begin);
    }

    double average_time = 0;
    double min_time = timeexe[0], max_time = timeexe[0];

    // TODO

    average_time /= NTest;
    printf("Best: %lf\n", min_time);
    printf("Average: %lf\n", average_time);
    printf("Worst: %lf\n", max_time);
    return 0;
}
```

Task 1.6 - 观察计算时间增长趋势

改变`dim = 10, 100, 1000, 10000`，输出排序的平均时间`t_avg`，最大时间`t_max`，画出：

- 插入排序的维度dim - 平均时间t_avg关系图。
- 插入排序的维度dim - 最大时间t_max关系图。

Task 2 - 选择排序 Selection Sort

换成选择排序，重复1-6步。

选择排序算法：

- 首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置。
- 再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。
- 重复第二步，直到所有元素均排序完毕。

```
void selection_sort(int arr[], int len)
{
    int i, j;
    for (i = 0; i < len - 1; i++)
    {
        int min = i;
        // TODO
    }
}
```

Task 3 - 冒泡排序 Bubble Sort

换成冒泡排序，重复1-6步。

冒泡排序算法：

- 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
- 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。
- 针对所有的元素重复以上的步骤，除了最后一个。
- 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。

```
void bubble_sort(int arr[], int len)
{
    int i, j, temp;
    for (i = 0; i < len - 1; i++)
    {
        // TODO
    }
}
```

Task 4 - 快速排序 Quick Sort

换成快速排序，重复1-6步。

快速排序算法，是对冒泡排序的一种改进。

- 从数列中挑出一个元素，称为 "基准"（pivot）；
- 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作；
- 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

```c
void quick_sort_recursive(int arr[], int start, int end)
{
    if (start >= end)
        return;
    int mid = arr[end];
    int left = start, right = end - 1;
    while (left < right)
    {
        // TODO
        swap(&arr[left], &arr[right]);
    }
    if (arr[left] >= arr[end])
        swap(&arr[left], &arr[end]);
    else
        left++;
    if (left)
        quick_sort_recursive(arr, start, left - 1);
    quick_sort_recursive(arr, left + 1, end);
}

void quick_sort(int arr[], int len)
{
    quick_sort_recursive(arr, 0, len - 1);
}
```

Homework

Task 5 - 归并排序 Merge Sort

参考Task2-4提供的代码模板，实现归并排序（Merge Sort）算法

改变 `dim = 10，100，1000，10000,100000`，输出排序所需的平均时间`t_avg`与最大时间`t_max`。

Task 6 - 整合与分析（提交代码与实验报告）

在代码文件`task6.c`内，

- 以 作业Task5中实现的 归并排序 为基础（模板参考Task2-4提供的代码）
- 整合 课上Task1-4中实现的 四种排序算法
- 在main函数中依次测试 五种排序算法（插入排序、选择排序、冒泡排序、快速排序、归并排序）
- 输出 五种排序算法 在排序不同数量的随机数时所需的平均时间`t_avg`与最大时间`t_max`

在实验报告`report.pdf`内，

- 分析 归并排序 的算法原理
- 画出 五种排序算法 的 维度`dim` - 平均时间`t_avg`关系图
- 总结并分析 五种排序算法 的 时间复杂度（最优、平均、最坏） 和 空间复杂度，以及相应原因。

注意：本次作业报告请在学习后写下自己的理解，不要只是复制粘贴AI生成的结果或是网上搜索到的解析。排序算法是算法中很经典很重要的一章，力求掌握。

作业提交内容

作业完成后，将`task6.c`与`report.pdf`压缩为`zip`文件后提交至Canvas。

提交前请检查代码与报告文件，避免出现误交、错交的情况。