浙江大学实验报告

专业：微电子科学与技术
姓名：焦天晟
学号：3220105664
日期：2023.06.24

课程名称：C 程序设计专题　　指导老师：翁恺　　成绩：

实验名称：作业 4：　比较自己实现的归并和快排的性能

一、实验题目要求：

比较自己实现的归并和快排的性能

二、实验思路

这是一个比较快排和归并排序性能的实验，所以首先要写好快排和归并排序的函数：

首先是快排，这个程序里我写了三种快排：

第一种：单指针循环快排：

单指针循环快排的基本思想是将数组分成两部分：小于基准元素和大于等于基准元素的部分。具体实现方式是以数组的第一个元素为基准元素，从数组的第二个元素开始遍历数组，如果当前元素小于基准元素，则将其交换到指针所在位置（即指针所在位置的元素一定小于基准元素），然后将指针向右移动一位；否则，当前元素大于等于基准元素，将指针继续向右移动一位即可。遍历完成后，将基准元素交换到指针所在位置，这样就将数组分成了小于和大于等于两部分。然后分别对这两部分递归进行快排即可。

```
void Multisort(int arr[], int l, int r) {
    if (l < r) {
        int pivot = arr[r];
        int i = l - 1;
        for (int j = l; j <= r - 1; j++) {
            if (arr[j] < pivot) {
                i++;
                swap(&arr[i], &arr[j]);
            }
        }
        swap(&arr[i + 1], &arr[r]);
        int p = i + 1;
        Multisort(arr, l, p - 1);
        Multisort(arr, p + 1, r);
```

```
    }
}
```

第二种是双指针循环快排：

双指针循环快排的基本思想是将数组分成三部分：小于第一个基准元素、大于等于第一个基准元素且小于第二个基准元素、大于等于第二个基准元素的部分。具体实现方式是选择数组的两个元素作为基准元素，分别称为 p 和 q，然后以 p 和 q 为分界点，使用两个指针 i 和 j 来遍历数组。初始时，i 和 j 都指向数组的起始位置，然后逐个检查数组中的元素，并根据其大小关系将其划分到相应的区间中。遍历完成后，将小于 p 和大于 q 的部分分别递归进行快排，等于 p 和等于 q 的部分不需要排序，将其留在数组中即可。对于大于等于 p 且小于 q 的部分，也需要递归进行快排。

```
void Doublesort(int arr[], int l, int r) {
    if (l < r) {
        int pivot = arr[l];
        int i = l, j = r;
        while (i < j) {
            while (i < j && arr[j] >= pivot) j--;
            if (i < j) arr[i++] = arr[j];
            while (i < j && arr[i] < pivot) i++;
            if (i < j) arr[j--] = arr[i];
        }
        arr[i] = pivot;
        Doublesort(arr, l, i - 1);
        Doublesort(arr, i + 1, r);
    }
}
```

第三种是三指针循环快排：

三路循环快排的基本思想是将数组分成三部分：小于基准元素、等于基准元素、大于基准元素的部分。具体实现方式是选择数组的一个元素作为基准元素，然后使用三个指针（lt、i 和 gt）来遍历数组。初始时，lt 和 i 都指向数组的起始位置，gt 指向数组的末尾位置。然后逐个检查数组中的元素，并根据其大小关系将其划分到相应的区间中。遍历完成后，将小于和大于基准元素的部分分别递归进行快排，等于基准元素的部分不需要排序，将其留在数组中即可。

```
void Threesort(int arr[], int l, int r) {
```

```
    if (l < r) {
        int pivot = arr[l];
        int lt = l, i = l + 1, gt = r;
        while (i <= gt) {
            if (arr[i] < pivot) {
                swap(&arr[i], &arr[lt]);
                lt++;
                i++;
            } else if (arr[i] > pivot) {
                swap(&arr[i], &arr[gt]);
                gt--;
            } else {
                i++;
            }
        }
        Threesort(arr, l, lt - 1);
        Threesort(arr, gt + 1, r);
    }
}
```

接下来是归并排序的实现：

归并排序的基本思想是，将待排序的数组不断地对半分，直到每个子数组只包含一个元素。然后将相邻的两个子数组合并成一个有序数组，逐步将子数组合并成规模更大的有序数组，直到最终得到一个排好序的完整数组。

```
void merge(int arr[], int left, int mid, int right) {
    int len = right - left + 1;
    int* temp = (int*)malloc(sizeof(int) * len);
    int i = left, j = mid + 1, k = 0;
    while (i <= mid && j <= right) {
        if (arr[i] <= arr[j]) {
            temp[k++] = arr[i++];
        } else {
            temp[k++] = arr[j++];
        }
```

```
    }
    while (i <= mid) {
        temp[k++] = arr[i++];
    }
    while (j <= right) {
        temp[k++] = arr[j++];
    }
    for (int p = 0; p < len; p++) {
        arr[left + p] = temp[p];
    }
    free(temp);
}
void MergeSort(int arr[], int left, int right) {
    if (left >= right) return;
    int mid = (left + right) / 2;
    MergeSort(arr, left, mid);
    MergeSort(arr, mid + 1, right);
    merge(arr, left, mid, right);
}
```

接下来就是要测试这四种排序方法的性能

为此编写了一个测试函数：

```
void test2() {
    srand(time(NULL));
    const int n = 100000000;
    printf("用这四个排序方法处理%d 个数据\n",n);
    int* arr1 = (int*)malloc(sizeof(int) * n);
    int* arr2 = (int*)malloc(sizeof(int) * n);
    int* arr3 = (int*)malloc(sizeof(int) * n);
    int* arr4 = (int*)malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++) {
        arr1[i] = arr2[i] = arr3[i] = arr4[i] = rand() % n;
    }
```

```
    printf("原始数组：");
    printArray(arr1, 20);

    clock_t start = clock();
    Multisort(arr1, 0, n - 1);
    clock_t end = clock();
    printf("单边循环快排用时：%lf 秒\n", (double)(end - start) /
CLOCKS_PER_SEC);
    //printf("排序后的数组：");
    //printArray(arr1, 500);
    free(arr1);

    start = clock();
    Doublesort(arr2, 0, n - 1);
    end = clock();
    printf("双边循环快排用时：%lf 秒\n", (double)(end - start) /
CLOCKS_PER_SEC);
    //printf("排序后的数组：");
    //printArray(arr2, 500);
    free(arr2);

    start = clock();
    Threesort(arr3, 0, n - 1);
    end = clock();
    printf("三路循环快排用时：%lf 秒\n", (double)(end - start) /
CLOCKS_PER_SEC);
    //printf("排序后的数组：");
    //printArray(arr3, 500);
    free(arr3);

    start = clock();
    MergeSort(arr4, 0, n - 1);
    end = clock();
```

```
    printf("归并排序用时：%lf 秒\n", (double)(end - start) /
CLOCKS_PER_SEC);
    //printf("排序后的数组：");
    //printArray(arr4, 500);
    free(arr4);
}
```

在使用了静态数组和链表这两种数据存储方法后发现静态数组不能储存较大数量的数据，而链表的排序处理有比较麻烦，因此采用了动态数组这一数据存储方法。

第一次：处理 1000000 个数据：

1. 用这四个排序方法处理 1000000 个数据

   单边循环快排用时：0.099000 秒

   双边循环快排用时：0.098000 秒

   三路循环快排用时：0.105000 秒

   归并排序用时：0.172000 秒

2. 用这四个排序方法处理 1000000 个数据

   单边循环快排用时：0.104000 秒

   双边循环快排用时：0.097000 秒

   三路循环快排用时：0.106000 秒

   归并排序用时：0.181000 秒

3. 用这四个排序方法处理 1000000 个数据

   单边循环快排用时：0.113000 秒

   双边循环快排用时：0.090000 秒

   三路循环快排用时：0.101000 秒

   归并排序用时：0.169000 秒

4. 用这四个排序方法处理 1000000 个数据

   单边循环快排用时：0.106000 秒

   双边循环快排用时：0.095000 秒

   三路循环快排用时：0.107000 秒

   归并排序用时：0.193000 秒

5. 用这四个排序方法处理 1000000 个数据

   单边循环快排用时：0.113000 秒

   双边循环快排用时：0.106000 秒

   三路循环快排用时：0.110000 秒

归并排序用时：0.191000 秒

可以看见，在处理较小量数据时双路循环快排用时是最快的，单路和三路相差无几，归并则处理较慢

第二次：处理 10000000 个数据：

1. 用这四个排序方法处理 10000000 个数据

   单边循环快排用时：1.896000 秒

   双边循环快排用时：1.796000 秒

   三路循环快排用时：1.011000 秒

   归并排序用时：1.733000 秒

2. 用这四个排序方法处理 10000000 个数据

   单边循环快排用时：1.892000 秒

   双边循环快排用时：1.810000 秒

   三路循环快排用时：0.993000 秒

   归并排序用时：1.756000 秒

3. 用这四个排序方法处理 10000000 个数据

   单边循环快排用时：1.918000 秒

   双边循环快排用时：1.821000 秒

   三路循环快排用时：1.018000 秒

   归并排序用时：1.793000 秒

4. 用这四个排序方法处理 10000000 个数据

   单边循环快排用时：1.902000 秒

   双边循环快排用时：1.788000 秒

   三路循环快排用时：1.009000 秒

   归并排序用时：1.764000 秒

5. 用这四个排序方法处理 10000000 个数据

   单边循环快排用时：1.938000 秒

   双边循环快排用时：1.812000 秒

   三路循环快排用时：0.996000 秒

   归并排序用时：1.750000 秒

比较上述数据可以发现，数据量变大时三路快排的优势开始展现，多次随机排序均展现了较好的性能。归并排序也开始用时小于单边与双边，且用时比较稳定。双边稍快于单边，但用时均较长。

第三次：处理 100000000 个数据：

1. 用这四个排序方法处理 100000000 个数据

   单边循环快排用时：135.635000 秒

   双边循环快排用时：141.175000 秒

   三路循环快排用时：13.584000 秒

   归并排序用时：25.716000 秒

2. 用这四个排序方法处理 100000000 个数据

   单边循环快排用时：115.301000 秒

   双边循环快排用时：105.446000 秒

   三路循环快排用时：10.710000 秒

   归并排序用时：19.135000 秒

3. 用这四个排序方法处理 100000000 个数据

   单边循环快排用时：111.646000 秒

   双边循环快排用时：103.887000 秒

   三路循环快排用时：10.787000 秒

   归并排序用时：19.708000 秒

4. 用这四个排序方法处理 100000000 个数据

   单边循环快排用时：130.906000 秒

   双边循环快排用时：148.661000 秒

   三路循环快排用时：13.409000 秒

   归并排序用时：25.450000 秒

5. 用这四个排序方法处理 100000000 个数据

   单边循环快排用时：139.640000 秒

   双边循环快排用时：138.837000 秒

   三路循环快排用时：13.187000 秒

   归并排序用时：24.269000 秒

单边循环快排和双边循环快排在处理 1000000 个数据时效率较高，但随着数据规模增大，它们的性能逐渐下降，而且在处理 100000000 个数据时，它们的用时非常长，分别需要几十秒至数分钟的时间。这是因为快速排序的最坏时间复杂度为 O(n^2)，当数据规模较大且分割不均匀时，会退化为该复杂度。

三路循环快排在处理 1000000 个数据时的效率比较接近单边循环快排和双边循环快排，但是随着数据规模的增大，它的性能优势逐渐显现，处理 100000000 个数据时的用时大约为 10-15 秒左右，远远快于单边循环快排和双边循环快排。这是因为三路快排能够更好地处理重复元素，避免了分割不均匀的问题，因此在大数据集上的效率更高。

归并排序在处理 1000000 个数据时的用时相对较长，但是它的时间复杂度始终为 O(n log n)，不会退化为 O(n^2)，因此随着数据规模的增大，其性能表现相对稳定。在处理 100000000 个数据时，归并排序的用时大约为 20-25 秒左右，比起单边循环快排和双边循环快排要快得多。

如果数据规模较小，可以使用单边循环快排、双边循环快排或三路循环快排。如果数据规模较大，特别是当数据存在大量重复元素时，三路循环快排的效率最高；如果需要保证稳定的性能表现，可以使用归并排序。

四、实验体会与心得：

在完成本次实验时，我深刻地认识到了算法对程序性能的影响。通过比较自己实现的归并排序和快速排序算法的性能，我发现不同的算法在处理不同规模的数据时，表现出了截然不同的性能。在处理较小规模的数据时，快速排序的效率要优于归并排序，而在处理较大规模的数据时，归并排序的效率则更高。同时，三路循环快排在处理存在大量重复元素的数据时，表现出了更好的性能。

另外，本次实验也让我更加深入地了解了快速排序和归并排序算法的具体实现方式。通过自己动手实现这些算法，我更好地理解了它们的核心思想和运行机制。同时，在实现算法的过程中，我也发现了一些容易出错的细节，例如数组越界等问题，这些问题在实现算法时需要特别注意。

本次实验让我更深入地理解了常用的排序算法，并且通过实际测试比较它们的性能，让我更加明确了在不同场景下选择不同算法的重要性。同时，通过实践，我也更加熟练地掌握了 C 语言的编程技巧和调试方法，对我的编程能力提升有很大的帮助。