南京邮电大学

实 验 报 告

（2024 / 2025 学年 第 一 学期）

课程名称	数据结构
实验名称	排序
实验时间	2024 年 12 月 26 日
指导单位	计算机学院
指导教师	孙海安

学生姓名	于明宏	班级学号	B23041011
学院(系)	计算机学院	专　　业	信息安全

实 验 报 告

实验名称	排序			指导教师	孙海安
实验类型	设计	实验学时	2	实验时间	2024.12.26

一、 实验目的和要求

1. 掌握各种内排序算法的实现方法。
2. 学会分析各种内排序算法的时间复杂度。

二、实验环境(实验设备)

硬件：微型计算机

软件：Ubuntu 22.04、gcc version 11.4.0

三、实验原理及内容

1、算法实现

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#define MaxSize 100000

typedef int KeyType;
typedef struct {
    KeyType key;
    char data[10];
} Entry;
```

```
typedef struct {
	int n;
	Entry* D;
} List;

void simpleSelectionSort(List* L) {
	int i, j, min;
	Entry temp;
	for (i = 0; i < L->n - 1; i++) {
		min = i;
		for (j = i + 1; j < L->n; j++) {
			if (L->D[j].key < L->D[min].key)
				min = j;
		}
		temp = L->D[i];
		L->D[i] = L->D[min];
		L->D[min] = temp;
	}
}

void directInsertionSort(List* L) {
	int i, j;
	Entry temp;
	for (i = 1; i < L->n; i++) {
		temp = L->D[i];
		j = i - 1;
		while (j >= 0 && L->D[j].key > temp.key) {
			L->D[j + 1] = L->D[j];
			j--;
		}
		L->D[j + 1] = temp;
	}
}
```

```
void bubbleSort(List* L) {
    int i, j;
    Entry temp;
    for (i = 0; i < L->n - 1; i++) {
        for (j = 0; j < L->n - i - 1; j++) {
            if (L->D[j].key > L->D[j + 1].key) {
                temp = L->D[j];
                L->D[j] = L->D[j + 1];
                L->D[j + 1] = temp;
            }
        }
    }
}

int partition(List* L, int low, int high) {
    Entry pivot = L->D[low];
    while (low < high) {
        while (low < high && L->D[high].key >= pivot.key)
            high--;
        L->D[low] = L->D[high];
        while (low < high && L->D[low].key <= pivot.key)
            low++;
        L->D[high] = L->D[low];
    }
    L->D[low] = pivot;
    return low;
}

void quickSortRecursive(List* L, int low, int high) {
    if (low < high) {
        int pivot = partition(L, low, high);
        quickSortRecursive(L, low, pivot - 1);
        quickSortRecursive(L, pivot + 1, high);
    }
}
```

```
void quickSort(List* L) {
    quickSortRecursive(L, 0, L->n - 1);
}

void merge(List* L, int left, int mid, int right) {
    int i, j, k;
    Entry* temp = (Entry*)malloc((right - left + 1) * sizeof(Entry));
    if (!temp) exit(EXIT_FAILURE);
    for (i = left, j = mid + 1, k = 0; i <= mid && j <= right;) {
        if (L->D[i].key <= L->D[j].key)
            temp[k++] = L->D[i++];
        else
            temp[k++] = L->D[j++];
    }
    while (i <= mid)
        temp[k++] = L->D[i++];
    while (j <= right)
        temp[k++] = L->D[j++];
    for (i = 0; i < k; i++)
        L->D[left + i] = temp[i];
    free(temp);
}

void mergeSortRecursive(List* L, int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSortRecursive(L, left, mid);
        mergeSortRecursive(L, mid + 1, right);
        merge(L, left, mid, right);
    }
}

void mergeSort(List* L) {
    mergeSortRecursive(L, 0, L->n - 1);
```

```
}

void heapify(List* L, int n, int i) {
    int largest = i, left = 2 * i + 1, right = 2 * i + 2;
    if (left < n && L->D[left].key > L->D[largest].key)
        largest = left;
    if (right < n && L->D[right].key > L->D[largest].key)
        largest = right;
    if (largest != i) {
        Entry temp = L->D[i];
        L->D[i] = L->D[largest];
        L->D[largest] = temp;
        heapify(L, n, largest);
    }
}

void heapSort(List* L) {
    int i;
    for (i = L->n / 2 - 1; i >= 0; i--)
        heapify(L, L->n, i);
    for (i = L->n - 1; i > 0; i--) {
        Entry temp = L->D[0];
        L->D[0] = L->D[i];
        L->D[i] = temp;
        heapify(L, i, 0);
    }
}

void generateRandomData(List* L, int n) {
    int i;
    L->n = n;
    srand((unsigned int)time(0));
    for (i = 0; i < n; i++) {
        L->D[i].key = rand() % 100000;
        snprintf(L->D[i].data, sizeof(L->D[i].data), "data%d", i);
```

```
        }
    }

    double measureSort(void (*sortFunc)(List*), List* L) {
        clock_t start, end;
        start = clock();
        sortFunc(L);
        end = clock();
        return (double)(((double)(end - start)) / (double)CLOCKS_PER_SEC) * 1000;
    }

    int main() {
        int sizes[] = { 500, 10000, 50000, 100000 };
        int i, j;
        double timeTaken[6][4];
        void (*sorts[])(List*) = { simpleSelectionSort, directInsertionSort, bubbleSort, quickSort, mergeSort, heapSort };
        char* names[] = { "Selection(ms):", "Insertion(ms):", "Bubble(ms):", "Quick(ms):", "Merge(ms):", "Heap(ms):" };

        List L;
        L.D = (Entry*)malloc(MaxSize * sizeof(Entry));
        if (!L.D) exit(EXIT_FAILURE);

        for (i = 0; i < 4; i++) {
            generateRandomData(&L, sizes[i]);
            for (j = 0; j < 6; j++) {
                List temp;
                temp.D = (Entry*)malloc(sizes[i] * sizeof(Entry));
                if (!temp.D) exit(EXIT_FAILURE);
                memcpy(temp.D, L.D, sizes[i] * sizeof(Entry));
                temp.n = L.n;
                timeTaken[j][i] = measureSort(sorts[j], &temp);
                free(temp.D);
            }
```

```
        }

        FILE* fp = fopen("sort_results.csv", "w");
        if (!fp) {
            free(L.D);
            return EXIT_FAILURE;
        }

        fprintf(fp, "Algorithm,500,10000,50000,100000\n");
        for (i = 0; i < 6; i++) {
            fprintf(fp, "%s", names[i]);
            for (j = 0; j < 4; j++) {
                fprintf(fp, ",%.3lf", timeTaken[i][j]);
            }
            fprintf(fp, "\n");
        }

        fclose(fp);
        free(L.D);
        return 0;
    }
```

2、复杂度分析

（1）simpleSelectionSort 函数

时间复杂度：$O(n^2)$

空间复杂度：O(1)

外层循环执行 n-1 次，内层循环每次需要遍历剩余的元素，时间复杂度为 $n + (n-1) + ... + 1 = O(n^2)$，使用常数空间进行交换操作，因此空间复杂度为 O(1)。

（2）directInsertionSort 函数

时间复杂度：$O(n^2)$（最坏情况），O(n)（最佳情况）

空间复杂度：O(1)

外层循环执行 n-1 次，内层循环在最坏情况下（数组是降序排列）执行 $1 + 2 + ... + (n-1) = O(n^2)$，最佳情况下（数组已排序），内层循环不会执行，时间复杂度为 O(n)，空间复杂度为 O(1)。

（3）bubbleSort 函数

时间复杂度：$O(n^2)$（最坏和平均情况），O(n)（最佳情况）

空间复杂度：O(1)

外层循环执行 n-1 次，内层循环在最坏情况下（数组是降序排列）执行 n-1 + n-2 + ... + 1 = O(n^2)，最佳情况下（数组已排序），只需执行一次检查，时间复杂度为 O(n)，空间复杂度为 O(1)。

（4）quickSort 函数

时间复杂度：O(n log n)（平均情况），O(n^2)（最坏情况）

空间复杂度：O(log n)（平均情况），O(n)（最坏情况）

平均情况下，每次分区将数组分成两个近似相等的子数组，分区过程的时间复杂度为 O(n)，递归深度为 O(log n)，因此总时间复杂度为 O(n log n)，最坏情况下（每次分区极不均衡，如已排序数组），递归深度为 O(n)，总时间复杂度为 O(n^2)，空间复杂度由递归栈深度决定，平均为 O(log n)，最坏为 O(n)。

（5）mergeSort 函数

时间复杂度：O(n log n)

空间复杂度：O(n)

每次将数组分为两部分需要 O(log n) 次递归，合并操作需要 O(n) 时间，总时间复杂度为 O(n log n)，额外分配临时数组存储数据，空间复杂度为 O(n)。

（6）heapSort 函数

时间复杂度：O(n log n)

空间复杂度：O(1)

建堆操作的时间复杂度为 O(n)，每次从堆中取最大元素并调整堆的复杂度为 O(log n)，需要进行 n 次调整，总复杂度为 O(n log n)，使用原地排序，没有额外空间需求，空间复杂度为 O(1)。

（7）generateRandomData 函数

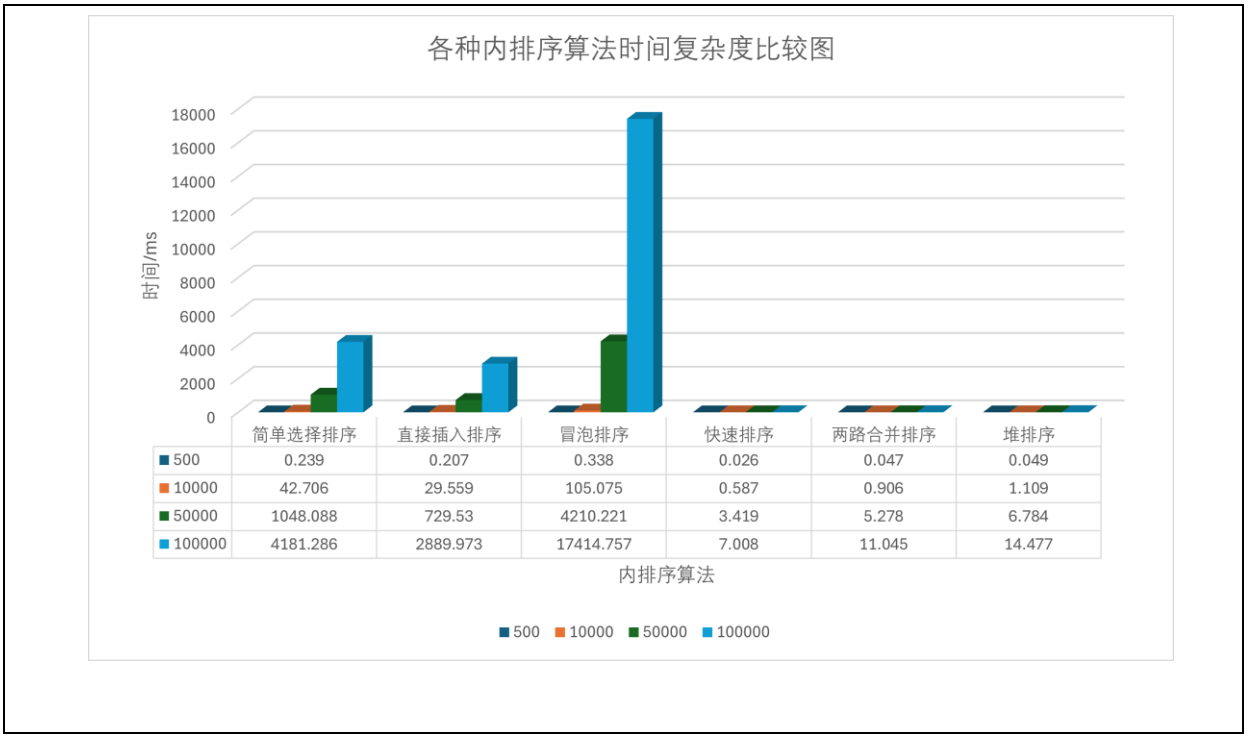
时间复杂度：O(n)

空间复杂度：O(1)

遍历 n 次生成随机数并赋值，总时间复杂度为 O(n)，仅使用常数空间。

3、实验结果与结论

关键字个数	500	10000	50000	100000
简单选择排序(ms)	0.239	42.706	1048.088	4181.286
直接插入排序(ms)	0.207	29.559	729.53	2889.973
冒泡排序(ms)	0.338	105.075	4210.221	17414.757
快速排序(ms)	0.026	0.587	3.419	7.008
两路合并排序(ms)	0.047	0.906	5.278	11.045
堆排序(ms)	0.049	1.109	6.784	14.477


各种内排序算法时间复杂度比较图
时间/ms
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
简单选择排序 直接插入排序 冒泡排序 快速排序 两路合并排序 堆排序
500 0.239 0.207 0.338 0.026 0.047 0.049
10000 42.706 29.559 105.075 0.587 0.906 1.109
50000 1048.088 729.53 4210.221 3.419 5.278 6.784
100000 4181.286 2889.973 17414.757 7.008 11.045 14.477
内排序算法
500 10000 50000 100000

四、实验小结（包括问题和解决方法、心得体会、意见与建议等）

（一）实验中遇到的主要问题及解决方法

1. 问题：在实现排序算法时，数组大小超过了预设的最大值，导致内存分配失败。

解决方法：检查内存分配语句，确保在分配内存前进行必要的边界检查，避免数组越界或内存溢出。可以增加异常处理机制来确保程序稳定运行。

2. 问题：在不同的排序算法中，时间复杂度分析时，未能准确计算最坏情况和最佳情况。

解决方法：通过手动推导和实验验证各排序算法的时间复杂度，特别是对于快速排序和归并排序，确保理解递归过程中的时间复杂度。

（二）实验心得

通过本次排序算法实验，我深入理解了不同内排序算法的实现及其优缺点。在实现选择排序、插入排序、冒泡排序、快速排序、归并排序和堆排序的过程中，我加深了对排序算法的理解，特别是它们的时间复杂度和空间复杂度分析。实验中，我还通过实际编程加深了对排序算法工作原理的理解，如快速排序的分治策略和归并排序的归并过程。此外，通过对不同算法在不同规模数据集上的性能测试，我体会到了时间复杂度对于算法效率的重要性，尤其是在大数据量的情况下，选择合适的排序算法至关重要。

（三）意见与建议（没有可省略）

可以提供更多的时间上机操作，以确保更多程序设计思路得以实现，提升面向对象语言的掌握程度和编程能力。

五、支撑毕业要求指标点

《数据结构》课程支撑毕业要求的指标点为：

1.2-M 掌握计算机软硬件相关工程基础知识，能将其用于分析计算机及应用领域的相关工程问题。

3.2-H 能够根据用户需求，选取适当的研究方法和技术手段，确定复杂工程问题的解决方案。

4.2-H 能够根据实验方案，配置实验环境、开展实验，使用定性或定量分析方法进行数据分析与处理，综合实验结果以获得合理有效的结论。

实验内容	支撑点 1.2	支撑点 3.2	支撑点 4.2
线性表及多项式的运算	√		
二叉树的基本操作及哈夫曼编码译码系统的实现		√	√
图的基本运算及智能交通中的最佳路径选择问题		√	√
各种内排序算法的实现及性能比较	√		√

六、指导教师评语 (含学生能力达成度的评价)

如评分细则所示

成　绩	XX	批阅人	XX	日　期	XXX

	评分项	优秀	良好	中等	合格	不合格
评	遵守实验室规章制度					
	学习态度					
	算法思想准备情况					
分	程序设计能力					
	解决问题能力					
	课题功能实现情况					
	算法设计合理性					
	算法效能评价					
	回答问题准确度					
细	报告书写认真程度					
	内容详实程度					
	文字表达熟练程度					
则	其它评价意见					
	本次实验能力达成评价（总成绩）					