哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业核心基础课（必修）

实验项目：内存排序方法及其应用

实验题目：内存排序算法实验比较

实验日期：2024 年 5 月 19 日

班级：22WL022

学号：2022110829

姓名：杨明达

设计成绩	报告成绩	任课老师
		张岩

一、实验目的

排序是计算机科学中的常见任务，它将一组无序的数据元素按照某种规则重新排列，以使得数据呈现有序的状态，便于后续的查找、统计和分析等操。当数据量较小时，将数据全部读入内存并进行排序的算法称为内存排序算法，常见的内存排序算法有：插入排序、冒泡排序、归并排序、快速排序、堆排序、基数排序等。本实验要求设计并实现上述内存排序算法并比较其运行速度。

二、实验要求及实验环境

实验要求：

1. 从文本文件中将两行数据读入内存，其中第一行有一个整数 n（n≤100000），表示待排序序列的长度，第二行有 n 个整数，用空格隔开，表示待排序序列。

2. 实现归并排序、快速排序算法，输出排序好的序列，记录算法运行时间。

3. 实现选择排序算法或插入排序算法，并将其运行时间与归并排序、快速排序算法比较，随机生成多个适当规模的数据进行实验并绘制折线图，反映不同算法运行时间随着输入规模的变化趋势，并与理论分析结果进行比较。

实验环境：Windows 11 && Visual Studio Code

三、设计思想（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系、核心算法的主要步骤）

1．逻辑设计

1.1 主程序的流程图

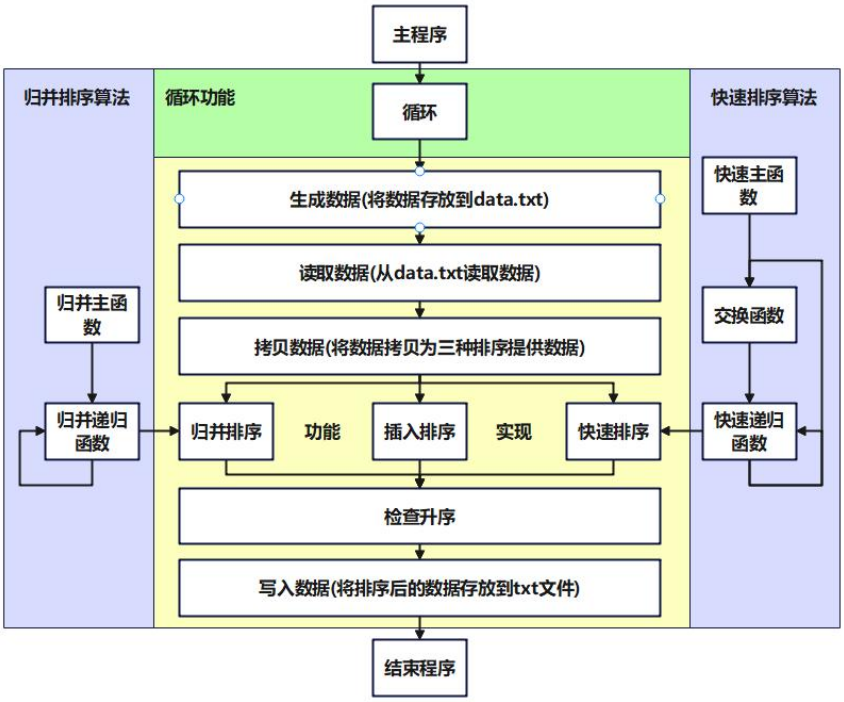

图 1 主程序流程图

1.2 各程序模块的调用关系

（1）主函数调用数据生成、数据读取、数据拷贝、归并排序、快速排序、插入排序、检查升序、数据写入函数模块。

（2）归并主函数调用归并递归函数模块。

（3）归并递归函数模块调用归并递归函数模块。

（4）快速主函数调用快速递归函数模块。

（5）快速递归函数调用快速递归函数、交换函数模块。

1.3 核心算法

（1）归并排序算法

在对数据进行排序是，归并排序算法是核心算法。在算法中，首先在归并主函数中声明一个数组用于存放归并后的数据，之后进入归并递归函数。在函数中，首先判断其实位置是否小于等于结尾位置，当满足条件，更新起始、中间、结尾位置，随后两次调用归并递归函数，进行中间位置左半部分和右半部分的排序，之后合并两个有序数组，并将合并后的数据存放在 reg 中，再将 reg 中的数据复制到 arr 中，以便下一次递归调用，知道递归结束，将排序后的数据存放在 arr 中，作为最终结果。

```
1.  void Merge_Sort_Recursive(int arr[], int reg[], int start, int end) {
2.      if (start >= end) {
3.          return;
4.      }
5.      int len = end - start, mid = (len >> 1) + start;
6.      int start1 = start, end1 = mid;
7.      int start2 = mid + 1, end2 = end;
8.      Merge_Sort_Recursive(arr, reg, start1, end1);
9.      Merge_Sort_Recursive(arr, reg, start2, end2);
10.     int k = start;
11.     while (start1 <= end1 && start2 <= end2) {
12.         reg[k++] = arr[start1] < arr[start2] ? arr[start1++] : arr[start2++];
13.     }
14.     while (start1 <= end1) {
15.         reg[k++] = arr[start1++];
16.     }
17.     while (start2 <= end2) {
18.         reg[k++] = arr[start2++];
```

```
    }
    for (k = start; k <= end; k++) {
        arr[k] = reg[k];
    }
}
void Merge_Sort(int arr[], const int len) {
    int reg[len];
    Merge_Sort_Recursive(arr, reg, 0, len - 1);
}
```

（2）快速排序算法

在对数据进行排序是，快速排序算法是核心算法。在算法中，快速主函数调用快速递归函数。在函数中，首先判断其实位置是否小于等于结尾位置，当满足条件，更新左端、中间、右端位置，交换左右两边的数据，使得左边的数据都小于 mid，右边的数据都大于 mid，然后递归调用，直到 start 大于等于 end，排序完成。

```
void swap(int *x, int *y) {
    int t = *x;
    *x = *y;
    *y = t;
}
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) {
        while (arr[left] < mid && left < right) {
            left++;
        }
        while (arr[right] >= mid && left < right) {
            right--;
        }
        swap(&arr[left], &arr[right]);
    }
    if (arr[left] >= arr[end]) {
        swap(&arr[left], &arr[end]);
    } else {
        left++;
    }
```

```
26.       if (left) {
27.           Quick_Sort_Recursive(arr, start, left - 1);
28.       }
29.       Quick_Sort_Recursive(arr, left + 1, end);
30.   }
31.   void Quick_Sort(int arr[], int len) { Quick_Sort_Recursiv
    e(arr, 0, len - 1); }
```

（3）插入排序算法

在对数据进行排序是，插入排序算法是核心算法。在算法中，从第二个元素开始，在待排序的数据中，直到找到它合适的位置，即按升序合适的位置，并将该元素插入到这个位置，并将后面的元素依次后移。

```
1.   void Insert_Sort(int arr[], int high) {
2.       for (int i = 1; i < high; i++) {
3.           int temp = arr[i];
4.           int j = i - 1;
5.           while (j >= 0 && arr[j] > temp) {
6.               arr[j + 1] = arr[j];
7.               j--;
8.           }
9.           arr[j + 1] = temp;
10.      }
11.  }
```

2．物理设计（即存储结构设计）

本实验主要采用一维数组存储待排序的数据，该存储结构的数据类型为 int。数组的最大空间为 MAX，并在此基础上将 arr[0]至 arr[high]空间存放数据。待排序数据存储结构的示意图如下图所示。

图 2 待排序数据存储结构

四、测试结果（包括测试数据、结果数据及结果的简单分析和结论，可以用截图得形式贴入此报告）

1.归并排序、快速排序、插入排序在不同数据规模下排序运行时间

由于数据规模在 10000 之前的时间较小以至于无法比较，故选取数据规模在 10000 至 100000，每隔 10000 测试一次。

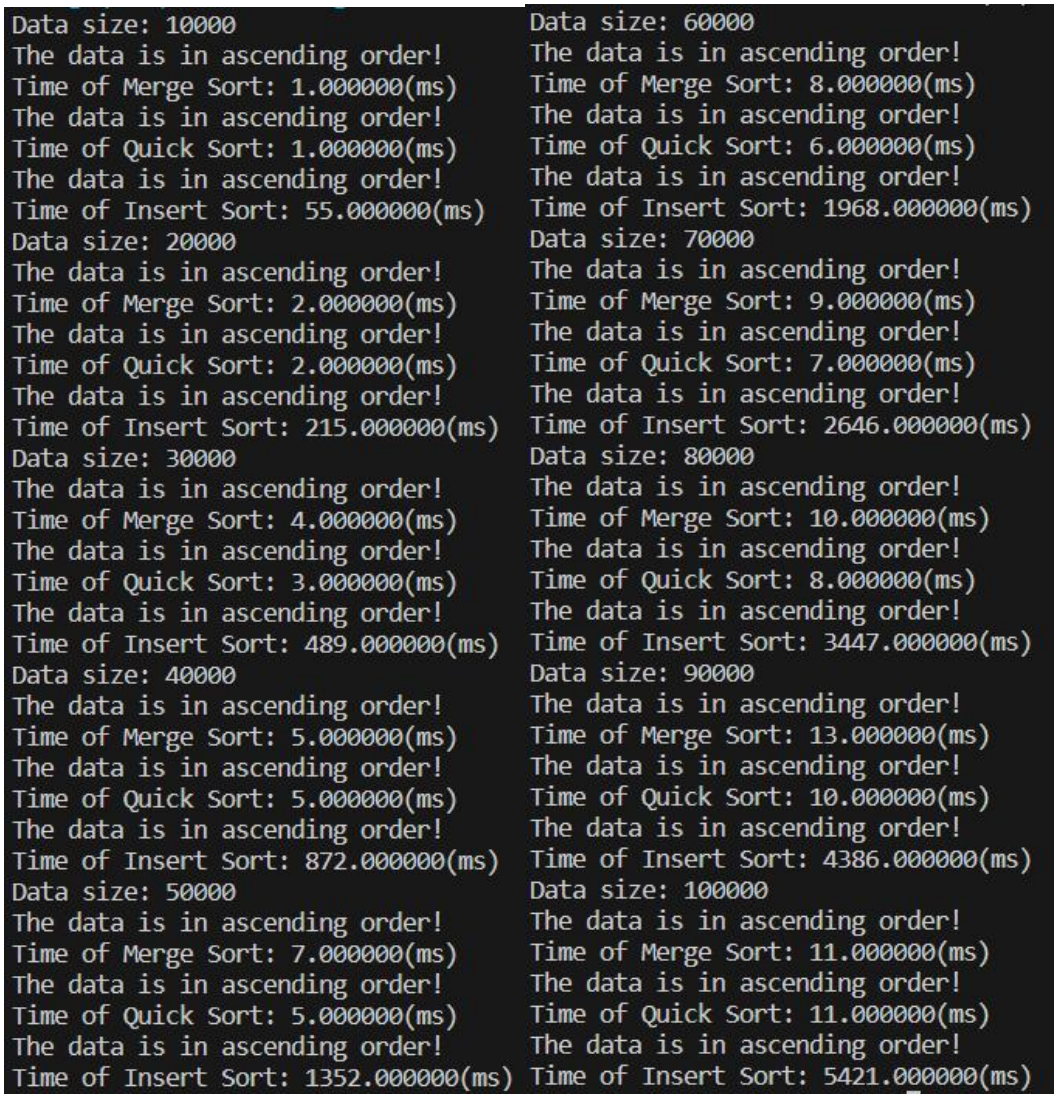

图 3 归并排序、快速排序、插入排序在不同数据规模下排序运行时间

2.时间复杂度

归并排序： $T(n)=n*T(1)+(\log_2 n)*n, T(1)=1$，**故** $T(n)=O(n\log_2 n)$

快速排序： $T(n)=O(n\log_2 n)$

插入排序：两层循环，故 $T(n)=O(n^2)$

经过实验验证，发现归并排序、快速排序和插入排序在数据规模较小时排序时间差不多，但随着数据规模逐渐增大，归并排序和快速排序的排序时间略微增加，而插入排序的排序时间增长幅度巨大，符合理论分析结果。

3. 绘制折线图

图 4 排序算法性能图

五、经验体会与不足

经验体会：通过实现归并排序、快速排序和插入排序的算法实现，我更好地掌握了排序算法的编写，并且通过进行排序算法的时间复杂度的理论分析和实验验证，我充分了解不同排序算法的排序性能差别也很大。通过在文件中读取数据等操作，我也掌握了 c 语言的文件操作。

不足：程序中的一些算法编写的不够高效简洁，还存在时间和空间的不必要消耗，并且程序的输出端口编写的也存在不规范之处。

六、附录：源代码（带注释）

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAX 100000
// 生成数据，第一行为一个整数 n(表示带排列序列的长度)，第二行有 n 个整数，表示待排序的序列
void Data_generation(int data_size) {
    srand((unsigned)time(NULL));
    FILE *fp = fopen("data.txt", "w");
    if (fp == NULL) {
        printf("File open failed!\n");
        exit(0);
    }
    fprintf(fp, "%d\n", data_size);
    for (int i = 0; i < data_size; i++) {
        fprintf(fp, "%d ", rand() % 100000);
    }
    fclose(fp);
}
// 读取数据
void Data_retrieve(int array[], int *high) {
    FILE *fp = fopen("data.txt", "r");
    if (fp == NULL) {
        printf("File open failed!\n");
        exit(0);
    }
    fscanf(fp, "%d", high);
    for (int i = 0; i < *high; i++) {
        fscanf(fp, "%d", &array[i]);
    }
    fclose(fp);
}
// 复制数据
void Data_Copy(int array[], int a[], int b[], int c[], int high) {
    for (int i = 0; i < high; i++) {
        a[i] = array[i];
        b[i] = array[i];
        c[i] = array[i];
    }
}
// 归并排序(分支算法)
```

```
41.  void Merge_Sort_Recursive(int arr[], int reg[], int start,
     int end) {
42.      if (start >= end) {
43.          return;
44.      }
45.      int len = end - start, mid = (len >> 1) + start;
46.      int start1 = start, end1 = mid;
47.      int start2 = mid + 1, end2 = end;
48.      Merge_Sort_Recursive(arr, reg, start1, end1);
49.      Merge_Sort_Recursive(arr, reg, start2, end2);
50.      int k = start;
51.      while (start1 <= end1 && start2 <= end2) {
52.          reg[k++] = arr[start1] < arr[start2] ? arr[start1
    ++] : arr[start2++];
53.      }
54.      while (start1 <= end1) {
55.          reg[k++] = arr[start1++];
56.      }
57.      while (start2 <= end2) {
58.          reg[k++] = arr[start2++];
59.      }
60.      for (k = start; k <= end; k++) {
61.          arr[k] = reg[k];
62.      }
63.  }
64.  void Merge_Sort(int arr[], const int len) {
65.      int reg[len];  // 用于存放归并后的数据
66.      Merge_Sort_Recursive(arr, reg, 0, len - 1);  // 递归调
    用
67.  }
68.  // 快速排序(分治算法)
69.  void swap(int *x, int *y) {
70.      int t = *x;
71.      *x = *y;
72.      *y = t;
73.  }
74.  void Quick_Sort_Recursive(int arr[], int start, int end)
    {
75.      if (start >= end) {
76.          return;
77.      }
78.      int mid = arr[end];
79.      int left = start, right = end - 1;
80.      while (left < right) {
```

```
81.             while (arr[left] < mid && left < right) {
82.                 left++;
83.             }
84.             while (arr[right] >= mid && left < right) {
85.                 right--;
86.             }
87.             swap(&arr[left], &arr[right]);
88.         }
89.         if (arr[left] >= arr[end]) {
90.             swap(&arr[left], &arr[end]);
91.         } else {
92.             left++;
93.         }
94.         if (left) {
95.             Quick_Sort_Recursive(arr, start, left - 1);
96.         }
97.         Quick_Sort_Recursive(arr, left + 1, end);
98.  }
99.  void Quick_Sort(int arr[], int len) { Quick_Sort_Recursiv
   e(arr, 0, len - 1); }
100. // 插入排序
101. void Insert_Sort(int arr[], int high) {
102.     for (int i = 1; i < high; i++) {
103.         int temp = arr[i];
104.         int j = i - 1;
105.         while (j >= 0 && arr[j] > temp) {
106.             arr[j + 1] = arr[j];
107.             j--;
108.         }
109.         arr[j + 1] = temp;
110.     }
111. }
112. // 将排序后的数据写入文件(从函数的 sortname 参数中获取排序方法
   的名称，并将排序后的数据写入文件 sortname.txt 中)
113. void Data_Write(int arr[], int high, int sort_number) {
114.     FILE *fp;
115.     if (sort_number == 1) {
116.         fp = fopen("Merge_Sort.txt", "w");
117.     } else if (sort_number == 2) {
118.         fp = fopen("Quick_Sort.txt", "w");
119.     } else if (sort_number == 3) {
120.         fp = fopen("Insert_Sort.txt", "w");
121.     }
122.     if (fp == NULL) {
```

```
123.         printf("File open failed!\n");
124.         exit(0);
125.     }
126.     fprintf(fp, "%d\n", high);
127.     for (int i = 0; i < high; i++) {
128.         fprintf(fp, "%d ", arr[i]);
129.     }
130.     fclose(fp);
131. }
132. // 检测是否为升序
133. void Detecting_ascending_order(int arr[], int high) {
134.     for (int i = 0; i < high - 1; i++) {
135.         if (arr[i] > arr[i + 1]) {
136.             printf("The data is not in ascending order!\n
   ");
137.             return;
138.         }
139.     }
140.     printf("The data is in ascending order!\n");
141. }
142. int main() {
143.     int array[MAX], a[MAX], b[MAX], c[MAX];
144.     int high;
145.     int data_size;
146.     double sum_time;
147.     clock_t start_t, end_t;
148.     for (int data_size = 10000; data_size <= 100000;
149.          data_size = data_size + 10000) {
150.         printf("Data size: %d\n", data_size);
151.         Data_generation(data_size);
152.         Data_retrieve(array, &high);
153.         Data_Copy(array, a, b, c, high);
154.
155.         start_t = clock();
156.         Merge_Sort(a, high);
157.         end_t = clock();
158.         Detecting_ascending_order(a, high);
159.         sum_time = ((double)(end_t - start_t)) / CLOCKS_P
   ER_SEC;
160.         printf("Time of Merge Sort: %f(ms)\n", sum_time *
    1000);
161.         Data_Write(a, high, 1);
162.
163.         start_t = clock();
```

```
164.            Quick_Sort(b, high);
165.            end_t = clock();
166.            Detecting_ascending_order(b, high);
167.            sum_time = ((double)(end_t - start_t)) / CLOCKS_P
    ER_SEC;
168.            printf("Time of Quick Sort: %f(ms)\n", sum_time *
     1000);
169.            Data_Write(b, high, 2);
170.
171.            start_t = clock();
172.            Insert_Sort(c, high);
173.            end_t = clock();
174.            Detecting_ascending_order(c, high);
175.            sum_time = ((double)(end_t - start_t)) / CLOCKS_P
    ER_SEC;
176.            printf("Time of Insert Sort: %f(ms)\n", sum_time
    * 1000);
177.            Data_Write(c, high, 3);
178.        }
179.        return 0;
180. }
```