**项目说明文档**

**数据结构课程设计**

**——排序算法比较**

### 作 者 姓 名 尹诚成

学 号 2351279

指 导 教 师 张 颖

学 院 专 业 计算机科学与技术学院 软件工程



二〇二三 年 十二 月 十三 日

**目录**

[1.项目分析 1](#_bookmark0)

* 1. .[项目背景分析 1](#_bookmark1)
  2. .[项目需求分析 1](#_bookmark2)
  3. .[项目功能分析 1](#_bookmark3)

1.3.1.冒泡排序算法 1

1.3.2.选择排序算法 1

1.3.3.插入排序算法 1

1.3.4.希尔排序算法 1

1.3.5.快速排序算法 1

1.3.6.堆排序算法 1

1.3.7.归并排序算法 1

1.3.8.基数排序算法 1

1.3.9.算法性能评测功能 1

1.3.10.异常处理功能 2

[2.项目设计 2](#_bookmark12)

* 1. .[数据结构设计 2](#_bookmark13)
  2. .[泛型化设计 2](#_bookmark14)
  3. 结构体设计 2
  4. .[项目主体架构设计 4](#_bookmark32)

[3.项目功能实现 4](#_bookmark33)

3.1.项目主体架构的实现 4

3.2.冒泡排序算法的实现 5

3.3.选择排序算法的实现 5

3.4.插入排序算法的实现 5

3.5.希尔排序算法的实现 5

3.6.快速排序算法的实现 5

3.7.堆排序算法的实现 5

3.8.归并排序算法的实现 5

3.9.基数排序算法的实现 5

3.10.算法性能评测功能的实现 5

3.11.异常处理功能的实现 5

4.项目测试 7

4.1.输入生成随机数的个数功能测试 5

4.2.排序算法功能正确性测试 5

1. **项目分析**
   1. 项目背景分析

通过比较不同的排序算法，可以加深对它们性能差异的理解。排序是数据处理的基础操作之一，不同的排序算法在不同应用场景下有着不同的适用性和效率。通过比较算法的时间复杂度、空间复杂度和稳定性，可以选择适合特定应用场景的排序方法。随着硬件和编程语言的发展，旧算法可能被优化或新算法可能被发明，因此持续的性能评估是必要的。

* 1. 项目需求分析

基于以上背景分析，本项目需要实现需求如下：

(1)多种排序算法：本项目需要实现多种排序算法；

(2)算法性能评测：本项目需要能够测量和比较每种排序算法的性能，包括但不限于运行时间和比较次数等；

(3)数据生成：能够生成不同规模和特性的数据以进行排序；

(4)异常处理：需要妥善处理异常情况，如内存分配失败和输入数据非法等情况。

* 1. 项目功能分析

本项目旨在通过实现多种排序算法，并进行算法性能评测，进行排序算法的比较，同时需要考虑用户交互、数据生成和异常处理等功能。通过本项目，可以更好地理解不同排序算法在处理大量数据时的效率和适用性。下面对项目的功能进行详细分析。

* + 1. 冒泡排序算法

冒泡排序是一种比较排序算法。它的基本思想是重复地遍历待排序的元素列表，比较相邻的两个元素，并根据比较结果交换它们的位置，直到整个列表变得有序为止。

* + 1. 选择排序算法

选择排序是一种比较排序算法。选择排序的基本思想是将待排序的元素分成两部分：已排序部分和未排序部分。初始时，已排序部分为空，而未排序部分包含所有待排序的元素。在每一轮中，选择排序会从未排序部分中找到最小（或最大）的元素，然后将其与未排序部分的第一个元素交换位置，将其放入已排序部分的末尾。

* + 1. 插入排序算法

插入排序是一种比较排序算法。插入排序的基本思想是将待排序的元素逐个插入到已经排好序的部分中，从而逐步构建有序的序列。

* + 1. 希尔排序算法

希尔排序是一种改进的插入排序算法，也被称为缩小增量排序。希尔排序通过将待排序的元素分成多个子序列，对每个子序列进行插入排序，然后逐渐缩小子序列的间隔，最终完成整体的排序。希尔排序的主要思想是通过较大的步长将元素移动到合适的位置，从而提前部分有序性，最终减少插入排序的工作量。

* + 1. 快速排序算法

快速排序是一种高效的分治排序算法，它常用于对大规模数据集进行排序。快速排序的基本思想是选择一个元素作为基准（通常选择最后一个元素），然后将数组分成两部分，使得左边的部分都小于基准，右边的部分都大于基准。接着，递归地对左右两部分进行排序，最终得到整个数组有序。

* + 1. 堆排序算法

堆排序是一种基于二叉堆数据结构的排序算法，它具有稳定的时间复杂度和较好的性能，通常用于对大规模数据集进行排序。堆是一种特殊的树形数据结构，分为最大堆和最小堆两种类型。最大堆要求父节点的值大于或等于其子节点的值，而最小堆则要求父节点的值小于或等于其子节点的值。在堆排序中，通常使用最大堆。

* + 1. 归并排序算法

归并排序是一种分治排序算法，它将待排序的数组分成两个部分，分别对这两部分进行排序，然后将它们合并成一个有序的数组。归并排序的核心思想是分而治之，通过将问题拆分为子问题并解决子问题，最终达到整体有序的目标。

* + 1. 基数排序算法

基数排序是一种非比较排序算法，它通过将待排序的元素按照各个位数的值（或者其他进制的位数）进行分组和排序，最终得到有序的结果。基数排序通常用于处理整数或字符串等数据类型。

* + 1. 算法性能评测功能

本项目需要能够测量和比较每种排序算法的性能，包括但不限于运行时间和比较次数等，同时需要考虑用户交互等功能。

* + 1. 异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

1. **项目设计**

2.1.数据结构设计

基于项目分析，在本项目中选择使用数组形式实现各个排序算法，主要基于以下几个考虑：

(1)性能因素：数组是一种性能高效的数据结构，对于排序算法来说，数组的随机访问速度非常快，这有助于提高排序算法的性能；

(2)通用性：数组是一种通用的数据结构，几乎可以用于任何数据类型的排序，只需要根据需要进行模板化或者修改比较和交换操作；

(3)理解性：使用数组利于从底层深入理解数据结构和排序算法的原理知识。使用数组需要注意数组的内存分配和释放，并且避免出现内存泄漏和数组访问越界等问题。

2.2.泛型化设计

程序使用泛型化设计实现各个排序算法。泛型化是指算法可以处理不同类型的数据，而不只是特定类型的数据。其具体实现如下：

(1)使用模板template

(2)<template T>是一个模板参数，表示排序函数使用一个名为 T 的类型数据来表示数组中的元素类型。

(3)T arr[]是一个类型为 T 的数组

(4)使用泛型参数 T 可以针对不同类型的数据进行排序

2.3.结构体设计

typedef void (\*SortFunction)(int\*, int)：SortFunction结构体用于指向不同的排序函数，这些函数的参数为一个整数数组和一个整数

struct SortOption {

SortFunction sortFunc;

const char \*name;

};

SortOption结构体用于存储排序函数及其名称，每个SortOption对象都包含一个SortFunction函数指针和一个字符串。

SortOption sortOptions[] = {

{bubbleSort, "冒泡排序"},

{selectionSort, "选择排序"},

{insertionSort, "插入排序"},

{shellSort, "希尔排序"},

{quickSort, "快速排序"},

{heapSort, "堆 排 序"},

{mergeSort, "归并排序"},

{radixSort, "基数排序"}

};

SortOption结构体数组包含了多个SortOption对象，每个对象代表一个排序算法，程序通过调用这些函数进行不同类型的排序。

2.4.项目主体架构设计

项目主体架构设计为：

(1)初始化随机数生成器；

(2)列出所有可选的排序算法；

(3)获取用户希望生成的随机数数量并创建一个整数数组并填充随机数；

(4)通过循环结构允许用户选择不同的排序算法进行比较；

(5)根据用户的选择执行相应的排序算法，并显示排序的时间和比较次数。

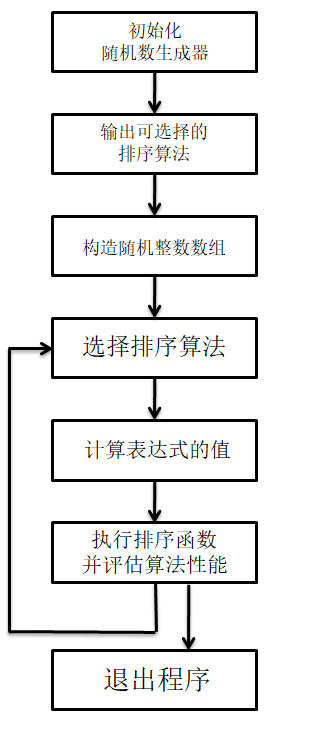


图 2.3 项目主体架构设计流程图

**3.项目功能实现**

3.1.项目主体架构的实现

项目主体架构实现思路为：

(1)调用srand函数初始化随机数生成器；

(2)打印系统进入提示，并打印所有可选的排序算法；

(3)调用inputInteger 函数获取用户希望生成的随机数数量，创建一个整数数组arr并填充随机数，随机数生成成功后打印给出提示信息；

(4)通过循环结构调用makeChoice函数允许用户选择不同的排序算法进行比较。如果用户选择选项0，则退出程序；

(5)根据用户的选择调用ChoiceSort函数，执行相应的排序算法，并显示排序的时间和比较次数。函数内部调用所选的排序函数对数组进行排序，实现了不同算法函数的调用一致性，并记录及显示排序时间和比较次数。

3.2.冒泡排序算法的实现

冒泡排序算法的实现思路为：

(1)从数组的第一个元素开始，比较相邻的元素；

(2)如果第一个比第二个大（小），就交换它们两个；

(3)对每一对相邻元素做同样的工作，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（或最小）的数；

(4)针对所有的元素重复以上的步骤，除了最后一个；

(5)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。

冒泡排序性能分析：

(1)时间复杂度分析：①最佳情况：T(n)=O(n)，当输入的数据已经是正序时，只需要进行一轮比较。②最差情况：T(n)=O(n²)，当输入的数据是反序时，需要进行n\*(n-1)/2次比较和交换。③平均情况：T(n)=O(n²)，平均情况下，比较和交换次数大致为n2/2。

(2)空间复杂度分析：空间复杂度为O(1)，除了输入数组外，冒泡排序只需要少量的临时存储空间

3.3.选择排序算法的实现

选择排序算法的实现思路为：

(1)首先在未排序序列中找到最小（或最大）元素，存放到排序序列的起始位置；

(2)然后，再从剩余未排序元素中继续寻找最小（或最大）元素，然后放到已排序序列的末尾；

(3)重复第二步，直到所有元素均排序完毕。

选择排序性能分析：

（1）时间复杂度分析：①最佳情况：T(n)=O(n2)，即使输入的数据是正序，选择排序仍然需要进行n\*(n-1)/2次比较；最坏情况：T(n)=O(n2)，输入数据是反序时，选择排序仍需进行n\*(n-1)/2次比较；平均情况：T(n)=O(n2)，选择排序的比较次数大致相同。

（2）空间复杂度分析：空间复杂度为O(1)，除了输入数组外，选择排序只需要少量的临时存储空间。

3.4.插入排序算法的实现

插入排序算法的实现思路为：

(1)从第一个元素开始，该元素可以认为已经被排序；

(2)取出下一个元素，在已经排序的元素序列中从后向前扫描；

(3)如果该元素（已排序）大于新元素，将该元素移到下一位置；

(4)重复步骤3，直到找到已排序的元素小于或等于新元素的位置；

(5)将新元素插入到该位置后；

(6)重复步骤2至5。

插入排序性能分析：

（1）时间复杂度分析：最佳情况：T(n)=O(n)，当输入数据为正序时；最坏情况：T(n)=O(n2)，当输入数据为反序时；平均情况：T(n)=O(n2)

（2）空间复杂度分析：空间复杂度为O(1)，除了输入数组外，插入排序只需要少量的临时存储空间。

3.5.希尔排序算法的实现

希尔排序算法的实现思路为：

(1)选择一个合适的增量序列。一开始选一个大的增量，逐渐减少增量，最后增量为1；

(2)根据当前的增量，将待排序列分割成若干个子序列，所有距离为增量的元素组成一个子序列；

(3)对每个子序列应用直接插入排序；

(4)减少增量，重复步骤2和3，直到增量为1，执行最后一次直接插入排序后排序完成。

希尔排序性能分析：

（1）时间复杂度分析：最佳情况：T(n)=O(nlogn)，着取决于增量序列；最坏情况：T(n)=O(n2)，这也取决于增量序列；平均情况：一般被认为是T(n)=O(nlog2n)或T(n)=O(n3/2)。

（2）空间复杂度分析：空间复杂度为O(1)，除了输入数组外，希尔排序只需要少量的临时存储空间。

3.6.快速排序算法的实现

快速排序算法的实现思路为：

(1)从数组中挑出一个元素，称为“基准”（pivot）；

(2)重新排序数组，所有比基准小的元素摆放在基准前面，所有比基准大的元素摆放在基准后面。在这个分割结束之后，该基准就处于数组的中间位置。这个称为分区操作；

(3)递归地把小于基准值元素的子数组和大于基准值元素的子数组排序。

快速排序性能分析：

（1）时间复杂度分析：最佳情况：T(n)=O(nlogn)，当每次分区操作都能均匀地划分数组时；最坏情况：T(n)=O(n2)，当每次分区操作只排除一个元素时；平均情况：T(n)=O(nlogn)。

（2）空间复杂度分析：空间复杂度为O(logn)，主要是递归调用的栈空间。

3.7.堆排序算法的实现

堆排序算法的实现思路为：

(1)将输入的数据数组构造成一个最大堆（或最小堆）；

(2)由于堆的最大（或最小）元素总是位于根节点，可以通过将其与堆的最后一个元素交换并减少堆的大小，从而将最大（或最小）元素移至数组末尾；

(3)重复这个过程，每次从堆中移除最大（或最小）元素，并减小堆的大小，直到堆的大小为1。

堆排序性能分析：

（1）时间复杂度分析：所有情况下都是T(n)=O(nlogn)。

（2）空间复杂度分析：空间复杂度为O(1)，除了输入数组外，堆排序只需要少量的临时存储空间。

3.8.归并排序算法的实现

归并排序算法的实现思路为：

(1)将数组分成两半，然后对每半分别进行归并排序；

(2)将排序后的两部分合并成一个完整的排序数组；

(3)合并时，从两个数组的起始位置开始比较，选择较小的元素放入到结果数组中，直到其中一个数组结束；

(4)将另一个数组中剩余的元素复制到结果数组中。

归并排序性能分析：

（1）时间复杂度分析：所有情况下都是T(n)=O(nlogn)。

（2）空间复杂度分析：空间复杂度为O(n)，归并排序需要与原数组相同数量地空间来存储排序后的合并结果。

3.9.基数排序算法的实现

基数排序算法的实现思路为：

(1)找到数组中最大数，并找出最大数的位数；

(2)从最低位开始，对数组中的每个元素按照当前位的数值进行排序；

(3)使用稳定的排序算法（如计数排序）来排序每一位；

(4)重复上述过程，直到最高位。

基数排序性能分析：

（1）时间复杂度分析：T(n)=O(nk)，其中n是需要排序的元素的数量，k是要排序元素的最大位数。

（2）空间复杂度分析：空间复杂度为O(n+k)。

3.10.算法性能评测功能的实现

算法性能评测功能的函数名为choiceSort，算法性能评测功能实现的思路为：

(1)初始化比较次数计数器；

(2)输出排序算法名称；

(3)进行排序数组的内存分配和复制；

(4)使用high\_resolution\_clock测量排序函数的执行时间；

(5)调用排序函数；

(6)输出该排序算法的排序时间和比较次数；

(7)释放动态分配的copyArr数组来避免内存泄漏。

3.11.异常处理功能的实现

在进行动态内存申请时，程序使用new(std::nothrow)来尝试分配内存。 new(std::nothrow)在分配内存失败时不会引发异常，而是返回一个空指针 （NULL或nullptr），代码检查指针是否为空指针，如果为空指针，意味着内存 分配失败，这时程序将执行以下操作：

(1)向标准错误流std::cerr输出一条错误消息"Error: Memory allocation failed."，指出内存分配失败；

(2)调用exit函数，返回错误码MEMORY\_ALLOCATION\_ERROR（通过宏定义方式定义为-1），用于指示内存分配错误，并导致程序退出。

**4.项目测试**

4.1.输入生成随机数个数功能测试

分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理。当输入合法时，程序继续运行。

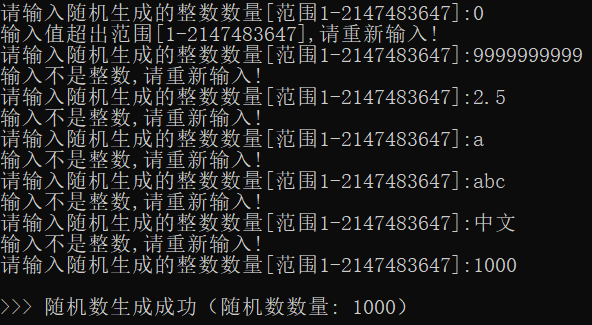


图 4.1 输入生成随机数个数功能测试

4.2. 排序算法功能正确性测试

修改代码使执行算法性能评测后输出排序后数组。输入要生成随机数的个数， 可以验证各个排序算法的正确性，均正确实现了数据的升序排列。

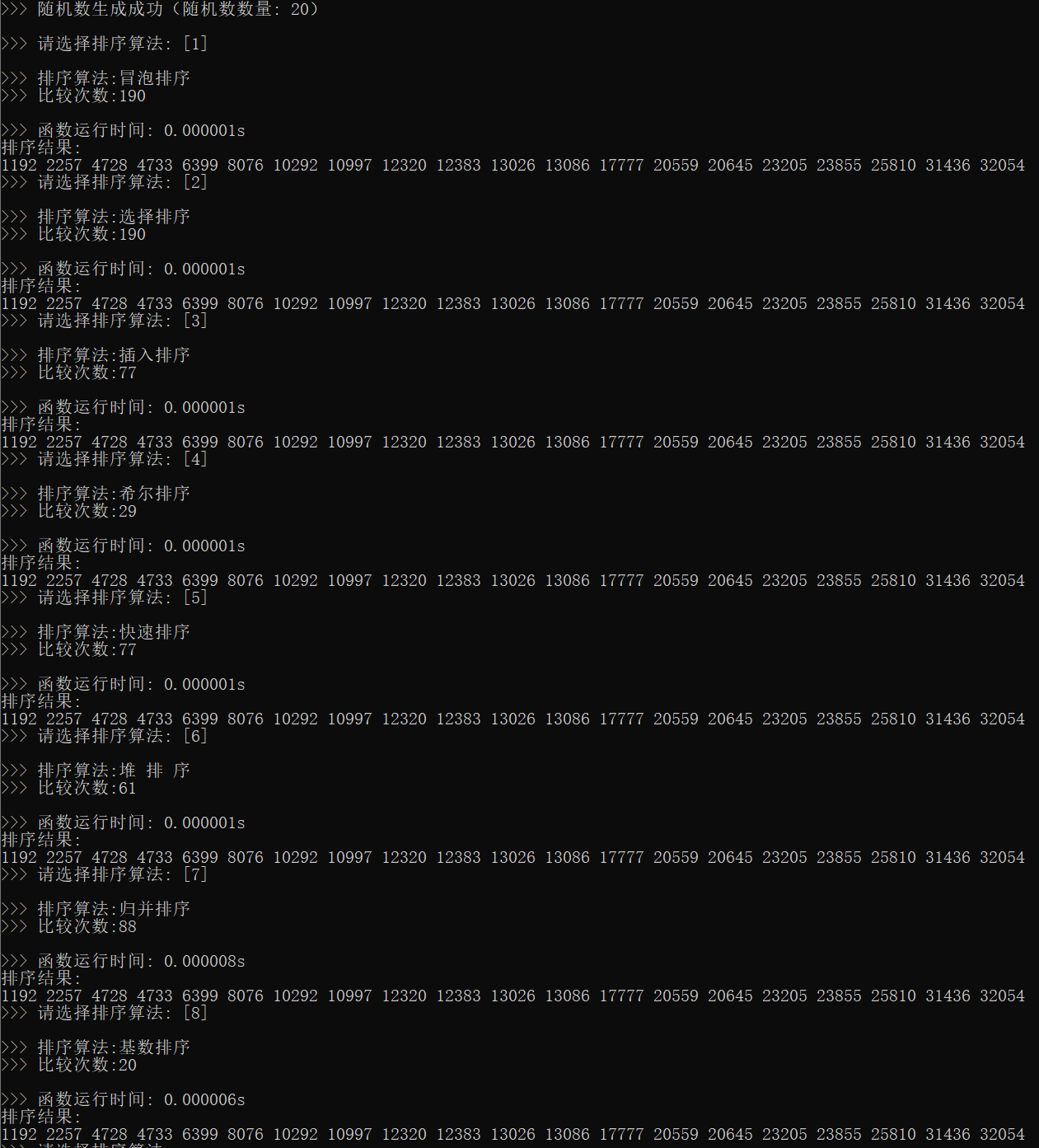


图 4.2 排序算法功能正确性测试