数据结构课程设计

项目说明文档

**8种排序算法的比较案例**



同济大学

Tongji University

姓名： 林觉凯

学号： 2253744

指导老师： 张颖

学院专业： 软件学院 软件工程

**目录**

**1.项目分析-------------------------------------------------------------------------3**

**1.1 项目背景分析-------------------------------------------------------------------------3**

**1.2项目功能分析--------------------------------------------------------------------------3**

**1.2.1项目功能要求**-------------------------------------------------------------------------------**3**

**1.2.2项目输入要求-------------------------------------------------------------------------------3**

**1.2.3项目输出要求-------------------------------------------------------------------------------3**

**1.2.4项目示例-------------------------------------------------------------------------------------3**

**2.项目实现-------------------------------------------------------------------------4**

**2.1 冒泡排序-------------------------------------------------------------------------------4**

**2.1.1冒泡排序逻辑-------------------------------------------------------------------------------4**

**2.1.2冒泡排序代码-------------------------------------------------------------------------------4**

**2.2 选择排序-------------------------------------------------------------------------------4**

**2.2.1选择排序逻辑-------------------------------------------------------------------------------4**

**2.2.2选择排序代码-------------------------------------------------------------------------------5**

**2.3 直接插入排序-------------------------------------------------------------------------5**

**2.3.1直接插入排序逻辑-------------------------------------------------------------------------5**

**2.3.2直接插入排序代码-------------------------------------------------------------------------5**

**2.4 希尔排序-------------------------------------------------------------------------------6**

**2.4.1希尔排序逻辑-------------------------------------------------------------------------------6**

**2.4.2希尔排序代码-------------------------------------------------------------------------------6**

**2.5 快速排序-------------------------------------------------------------------------------7**

**2.5.1快速排序逻辑-------------------------------------------------------------------------------7**

**2.5.2快速排序代码-------------------------------------------------------------------------------7**

**2.6 堆排序----------------------------------------------------------------------------------8**

**2.6.1堆排序逻辑----------------------------------------------------------------------------------8**

**2.6.2堆排序代码----------------------------------------------------------------------------------8**

**2.7 归并排序-------------------------------------------------------------------------------9**

**2.7.1归并排序逻辑-------------------------------------------------------------------------------9**

**2.7.2归并排序代码-------------------------------------------------------------------------------9**

**2.8 基数排序------------------------------------------------------------------------------10**

**2.8.1基数排序逻辑-------------------------------------------------------------------------------10**

**2.8.2基数排序代码-------------------------------------------------------------------------------10**

**3.项目测试------------------------------------------------------------------------12**

**3.1 项目功能测试------------------------------------------------------------------------12**

**3.2 项目代码健壮性测试---------------------------------------------------------------12**

**3.2.1输入随机数个数的代码健壮性测试----------------------------------------------------12**

**3.2.2输入排序算法选项的代码健壮性测试-------------------------------------------------13**

**4.项目心得与体会---------------------------------------------------------------13**

**1.项目分析**

**1.1 项目背景分析**

随机函数产生10000个随机数，用冒泡排序，选择排序，直接插入排序，希尔排序，快速排序，堆排序，归并排序和基数排序八种排序方法排序，并统计每种排序所花费的排序时间和比较次数（或者交换次数）。其中，随机数的个数由用户定义，系统产生随机数，并且显示他们的比较次数（或者交换次数）。

**1.2 项目功能分析**

**1.2.1项目功能要求**

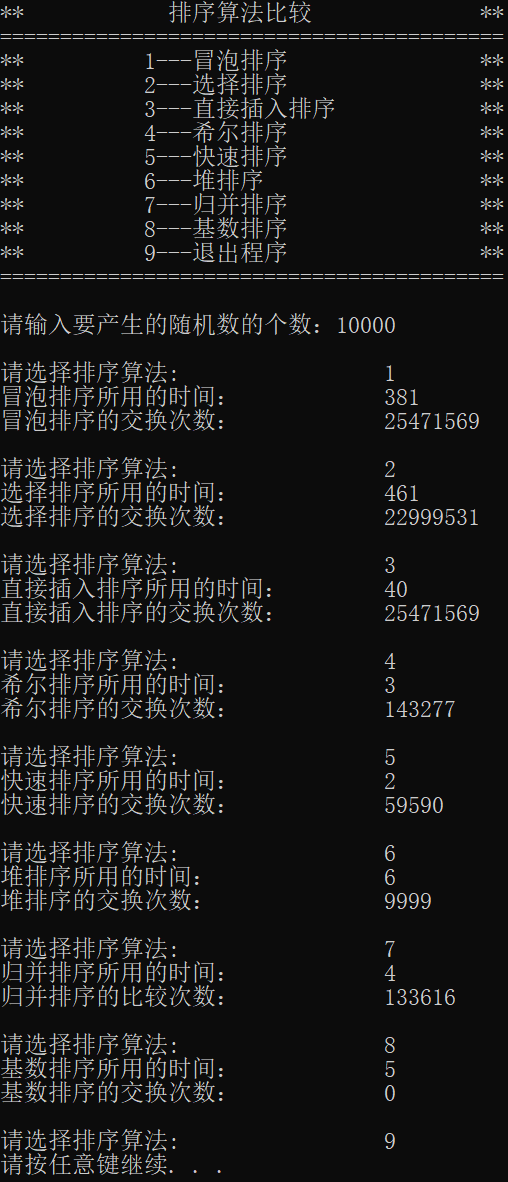
本项目是对八种排序算法的比较案例，项目功能要求我们可以随机产生用户输入的随机数的个数，并且可以按照用户的指定排序方法计算该方法下的排序时间和比较次数（或者交换次数）并进行输出。

**1.2.2项目输入要求**

用户输入所要排序的数字个数，每次用户输入选择的排序方法。

**1.2.3项目输出要求**

输出按照用户的指定排序方法计算该方法下的排序时间和比较次数（或者交换次数）。在输入不正确的时候输出提示信息。

**1.2.4项目示例**

**2.项目实现**

**2.1 冒泡排序**

**2.1.1 冒泡排序逻辑**

冒泡排序通过比较相邻的两个元素，如果它们的顺序不对则交换它们，直到整个数组排序完成。首先，从数组的第一个元素开始，将其与下一个元素比较。如果当前元素大于下一个元素，则交换它们的位置。接下来，继续比较下一个元素和它后面的元素，重复上述步骤，直到遍历到数组的倒数第二个元素。重复以上步骤，直到整个数组排序完成。冒泡排序的时间复杂度为。

**2.1.2 冒泡排序代码**

void Bubble\_sort(int\* array,int number)

{

for (int i = 0; i < number; i++)

{

for (int j = 0; j < number - i - 1; j++)

{

if (array[j] > array[j + 1])

{

swap(array[j], array[j + 1]);

Bubble\_count++;

}

}

}

}

**2.2 选择排序**

**2.2.1 选择排序逻辑**

选择排序每次从未排序的部分中选择最小（或最大）的元素，然后将其放到已排序部分的末尾。首先，找到未排序部分中的最小（或最大）元素。将找到的最小（或最大）元素与未排序部分的第一个元素交换位置，使其成为已排序部分的最后一个元素。接下来，将未排序部分缩小一个元素，即将已排序部分的末尾作为新的未排序部分的起始位置。重复以上步骤，直到未排序部分为空。选择排序的时间复杂度为，其中*n*为数组的长度。

**2.2.2 选择排序代码**

void Select\_sort(int\* array, int number)

{

for (int i = 0; i < number - 1; i++)

{

int Min\_index = i;

for (int j = i; j < number; j++)

{

if (array[Min\_index] > array[j])

Min\_index = j;

Select\_count++;

}

if (Min\_index != i)

swap(array[i], array[Min\_index]);

}

}

**2.3 直接插入排序**

**2.3.1 直接插入排序逻辑**

直接插入排序将待排序的元素按照顺序一个个插入到已排序部分的合适位置。首先，将数组的第一个元素看作已排序部分，其余元素为未排序部分。遍历未排序部分的元素，将每个元素插入到已排序部分的合适位置。对于每个未排序元素，从已排序部分的末尾开始比较，如果当前已排序元素大于待插入元素，则将当前元素后移一位，直到找到合适的位置。将待插入元素插入到找到的合适位置。重复以上步骤，直到未排序部分为空。直接插入排序的时间复杂度为，其中*n*为数组的长度。然而，在某些特定情况下，如当待排序数组已经基本有序时，直接插入排序的时间复杂度可以降低到O(*n*)。

**2.3.2 直接插入排序代码**

void Insert\_sort(int\* array, int number)

{

for (int i = 1; i < number; i++)

{

int temp = array[i];

int j = i - 1;

while (j >= 0 && array[j] > temp)

{

array[j + 1] = array[j];

Insert\_count++;

j--;

}

array[j + 1] = temp;

}

}

**2.4 希尔排序**

**2.4.1 希尔排序逻辑**

希尔排序是插入排序算法的改进版，它通过将待排序的元素分成多个子序列，分别进行插入排序，然后逐步缩小增量，直到增量为1时完成最后一次插入排序。

首先，选择一个增量序列，通常为*n*/2，*n*/4，*n*/8，...，1，其中*n*为数组的长度。根据增量将待排序元素分成多个子序列，每个子序列使用插入排序进行排序。逐步缩小增量，重复上述步骤，直到增量为1时进行最后一次插入排序。大量实验统计得出，希尔排序的渐近时间复杂度大约是，但相对于一般的插入排序，希尔排序具有更好的性能。

**2.4.2 希尔排序代码**

void Shell\_sort(int\* array, int number)

{

int gap = number / 2;

while (gap)

{

for (int i = gap; i < number; i++)

{

int temp = array[i];

int j = i - gap;

while (j >= 0 && array[j] > temp)

{

array[j + gap] = array[j];

Shell\_count++;

j -= gap;

}

array[j + gap] = temp;

}

gap /= 2;

}

}

**2.5 快速排序**

**2.5.1 快速排序逻辑**

快速排序是一种高效的排序算法，它使用了分治的思想。快速排序的核心思想是通过一趟排序将待排序的元素分割成独立的两部分，其中一部分的元素都小于另一部分的元素，然后再分别对这两部分进行排序。选择一个基准元素（通常为待排序数组的第一个或最后一个元素）。将数组划分为两个子数组，小于基准元素的放在左边，大于基准元素的放在右边。在划分过程中，可以使用两个指针分别从左右两端向中间扫描，将小于基准元素的交换到左边，大于基准元素的交换到右边。对左右两个子数组分别重复上述步骤，直到子数组的长度为1或0，即完成排序。快速排序的时间复杂度为。

**2.5.2 快速排序代码**

void Quick\_sort(int\* array, int left, int right)

{

if (left < right)

{

int i = left, j = right, x = array[left];

while (i < j)

{

while (i < j && array[j] >= x)

j--;

if (i < j)

{

array[i++] = array[j];

Quick\_count++;

}

while (i < j && array[i] < x)

i++;

if (i < j)

{

array[j--] = array[i];

Quick\_count++;

}

}

array[i] = x;

Quick\_count++;

Quick\_sort(array, left, i - 1);

Quick\_sort(array, i + 1, right);

}

}

**2.6 堆排序**

**2.6.1 堆排序逻辑**

堆排序是一种基于堆数据结构的排序算法，它利用堆的性质进行排序。堆是一个完全二叉树，并且具有以下性质：对于任意节点i，其父节点的值大于等于（或小于等于）其左右子节点的值。将待排序的数组构建成一个大顶堆（或小顶堆），即满足堆的性质。将堆顶元素（最大值或最小值）与堆的最后一个元素交换位置，然后将堆的大小减1，即将最大值（或最小值）放到已排序的部分。重新构建堆，将剩余的元素调整为大顶堆（或小顶堆）。重复上述步骤，直到堆的大小为1，即完成排序。堆排序的时间复杂度为。

**2.6.2 堆排序代码**

void Adjust(int\* array, int length, int index)

{

int left = 2 \* index + 1;

int right = 2 \* index + 2;

int max\_idx = index;

if (left < length && array[left] > array[max\_idx])

max\_idx = left;

if (right < length && array[right] > array[max\_idx])

max\_idx = right;

if (max\_idx != index)

{

swap(array[max\_idx], array[index]);

Adjust(array, length, max\_idx);

}

}

void Heap\_sort(int\* array, int number)

{

for (int i = number / 2 - 1; i >= 0; i--)

Adjust(array, number, i);

for (int i = number - 1; i >= 1; i--)

{

swap(array[0], array[i]);

Heap\_count++;

Adjust(array, i, 0);

}

}

**2.7 归并排序**

**2.7.1 归并排序逻辑**

归并排序采用了分治的思想，归并排序的核心思想是将待排序的数组不断地划分为更小的子数组，然后将这些子数组进行合并，直到最终得到有序的数组。

将待排序的数组不断地划分成更小的子数组，直到每个子数组的长度为1。将相邻的两个子数组进行合并，合并过程中将两个子数组按照从小到大的顺序依次取出元素，放入一个辅助数组中。重复上述合并步骤，直到所有子数组都合并成一个有序的数组。归并排序的时间复杂度为。

**2.7.2 归并排序代码**

void Merge(int\* array, int start, int mid, int end)

{

int length = end - start + 1;

int\* temp\_array = new int[length];

for (int index = 0; index < length; index++)

temp\_array[index] = array[start + index];

int i = 0;

int j = mid - start + 1;

for (int k = start; k <= end; k++)

{

if (i == mid - start + 1)

{

array[k] = temp\_array[j++];

Merge\_count++;

continue;

}

if (j == end - start + 1)

{

array[k] = temp\_array[i++];

Merge\_count++;

continue;

}

if (temp\_array[i] > temp\_array[j] && temp\_array[1])

{

array[k] = temp\_array[j++];

Merge\_count++;

}

else

{

array[k] = temp\_array[i++];

Merge\_count++;

}

}

delete[] temp\_array;

}

void Merge\_sort(int\* array, int start, int end)

{

if (start >= end)

return;

int mid = (start + end) / 2;

Merge\_sort(array, start, mid);

Merge\_sort(array, mid + 1, end);

Merge(array, start, mid, end);

}

**2.8 基数排序**

**2.8.1 基数排序逻辑**

基数排序根据元素的每个位上的值进行排序。基数排序的核心思想是将待排序的元素按照个位、十位、百位等位数进行排序，直到最高位，最终得到有序的数组。首先，确定待排序数组中最大值的位数，记为*d*。从个位开始，对待排序数组按照当前位的值进行计数排序。将排序结果覆盖原数组，得到按照当前位排序后的数组。重复上述步骤，直到对最高位进行排序，最终得到有序的数组。

基数排序的时间复杂度为*O*(*d*\*(*n*+*r*))，其中*d*是最大值的位数，*n*是数组的长度，r是基数。在每一次计数排序的过程中，需要遍历待排序的数组一次，所以计数排序的时间复杂度为*O*(*n*)，而进行*d*次计数排序。

**2.8.2 基数排序代码**

int Get\_maxbit(int\* array, int number)

{

int Max\_num = array[0];

for (int i = 0; i < number; i++)

{

if (array[i] > Max\_num)

Max\_num = array[i];

}

int bit = 0;

while (Max\_num > 0)

{

Max\_num /= 10;

bit++;

}

return bit;

}

int Get\_index\_number(int number, int index)

{

int i = 1;

while (index > 1)

{

i \*= 10;

index--;

}

return number / i % 10;

}

void Radix\_sort(int\* array, int number)

{

int maxbit = Get\_maxbit(array, number);

int\* temp\_array = new int[number];

int count[10] = { 0 };

for (int d = 1; d <= maxbit; d++)

{

for (int i = 0; i < 10; i++)

count[i] = 0;

for (int i = 0; i < number; i++)

count[Get\_index\_number(array[i], d)]++;

for (int i = 1; i < 10; i++)

count[i] += count[i - 1];

for (int i = number - 1; i >= 0; i--)

{

int k = Get\_index\_number(array[i], d);

temp\_array[count[k] - 1] = array[i];

count[k]--;

}

for (int j = 0; j < number; j++)

array[j] = temp\_array[j];

}

delete[] temp\_array;

}

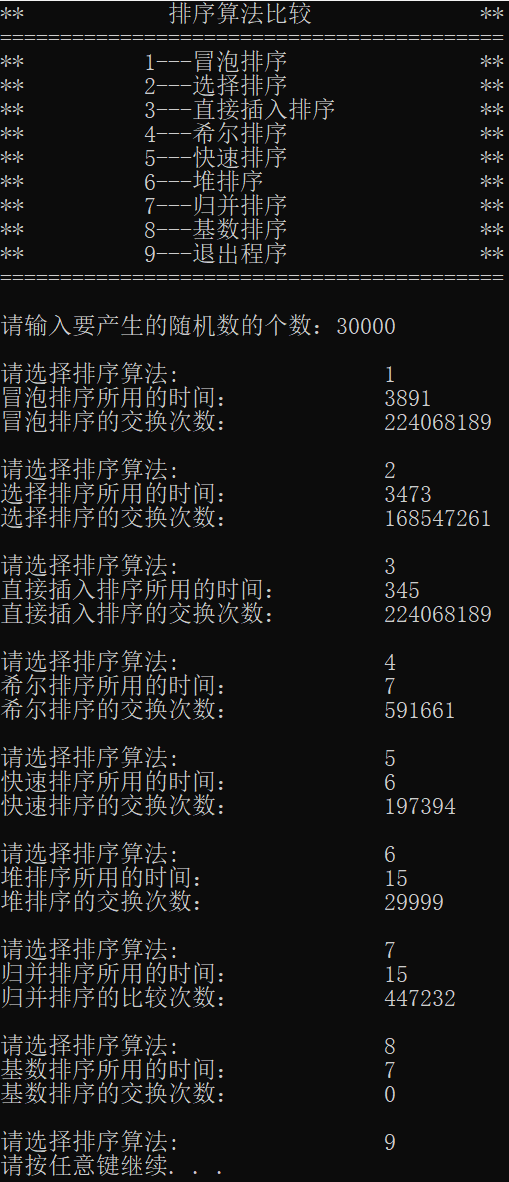
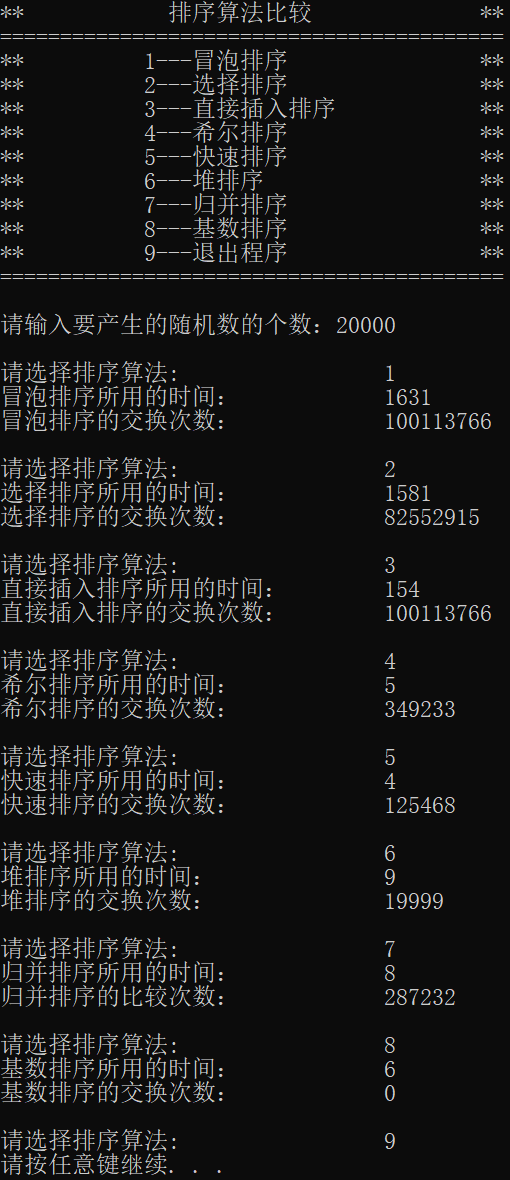
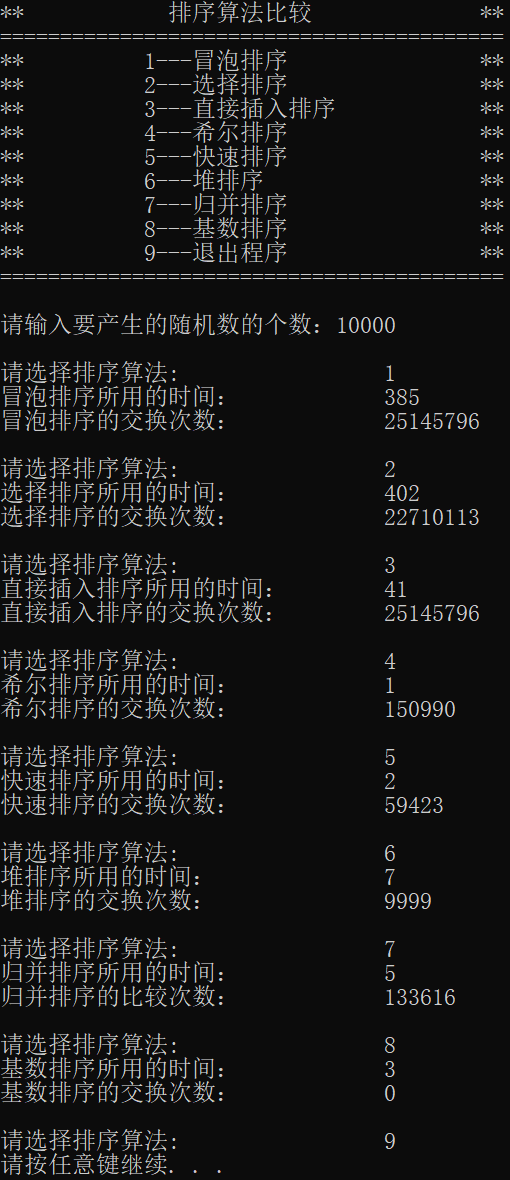
**3.项目测试**

**3.1 项目功能测试**

**测试用例：**生成的随机数个数分别为10000、20000和30000

**预期结果：**输出相应排序方法的时间和交换或比较次数

**实验结果：**

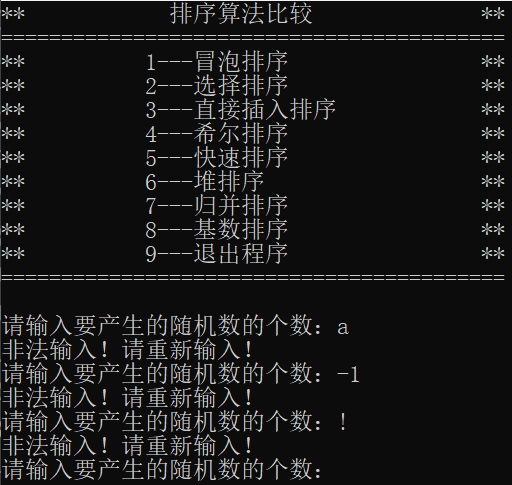
****

**3.2 代码健壮性测试**

**3.2.1 输入随机数个数的代码健壮性测试**

**测试用例：**在输入生成随机数个数的时候输入非法数据

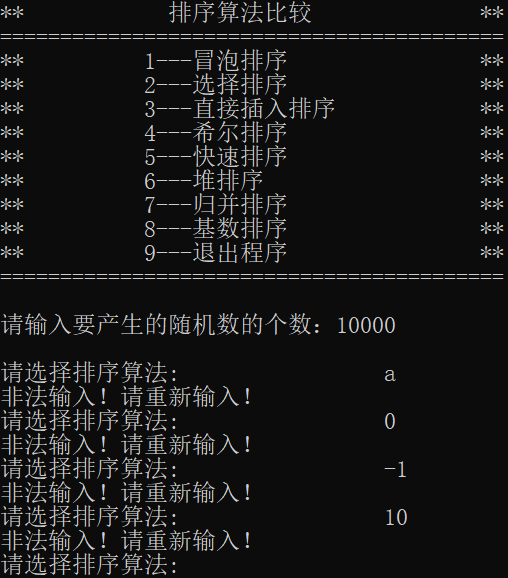
**预期结果：**给出相应提示程序运行不崩溃

**实验结果：**

**3.2.2 输入排序算法选项的代码健壮性测试**

**测试用例：**在输入排序算法选项的时候输入非法数据

**预期结果：**给出相应提示程序运行不崩溃

**实验结果：**

**4.项目心得与体会**

通过这次项目的编写，首先我对八种排序算法的时间复杂度有了更深一步的理解，理解了每一种排序方法的时间复杂度的大致计算过程；其次，本次项目练习了我对函数分解分块书写的能力，此次项目由十几个小函数构成为一个大项目；最主要的是，通过这次项目，我对八大排序算法的原理更加了解，之前可能只是只有听说过，直接调用过，但是没有实际地去编写相应的程序。在编写程序的过程中，对于堆排序、归并排序和基数排序的编写比较困难，需要在之后的学习和练习中不断加深对其原理的理解。在写完这次项目之后，以后有需要用到排序算法的时候，我可以有选择地挑出八种基本方法的某一种直接调用。