项目说明文档

数据结构课程设计

——8种排序算法的比较案例

作 者 姓 名： 李子涵

学 号： 1851892

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目录

**[1 算法设计 3](#_Toc22378)**

[1.1 冒泡排序核心代码 3](#_Toc25758)

[1.2 选择排序核心代码 3](#_Toc18954)

[1.3 直接插入排序核心代码 4](#_Toc14115)

[1.4 希尔排序核心代码 4](#_Toc2673)

[1.5 快速排序核心代码 4](#_Toc10410)

[1.6 快速排序核心代码 5](#_Toc3521)

[1.7 归并排序核心代码 6](#_Toc18084)

[1.8 基数排序核心代码 6](#_Toc11575)

**[2 运行结果 8](#_Toc10425)**

[2.1 100个随机数排序比较 8](#_Toc16305)

[2.2 1000个随机数排序比较 9](#_Toc22724)

[2.3 10000个随机数排序比较 11](#_Toc17479)

[2.4 100000个随机数排序比较 13](#_Toc11134)

**[3 算法分析 15](#_Toc20169)**

[3.1 时间开销分析 15](#_Toc6020)

[3.2 储存开销分析 17](#_Toc11562)

[3.3 优缺点总结 17](#_Toc17832)

# 

# 1 算法设计

* 1. 冒泡排序核心代码

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

for(int j=n-1;j>=i;j--)

if (r[j - i] > r[j])

{

int t = r[j - 1];

r[j - 1] = r[j];

r[j] = t;

}

* 1. 选择排序核心代码

int k = i;

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

if (r[j] < r[k])

k = j;

if (k != i)

{

int t = r[k];

r[k] = r[i];

r[i] = t;

}

* 1. 直接插入排序核心代码

int\* rp = new int[n];

for(int i=1;i<=n-1;i++)

if (r[i] < r[i - 1])

{

t = r[i];

int j = i - 1;

do

{

r[j + 1] = r[j];

count++;

j--;

} while (j >= 0 && t < r[j]);

r[j + 1] = t;

count++;

}

delete[] rp;

* 1. 希尔排序核心代码
  2. 快速排序核心代码

int partition(int r[],const int low,const int high,int &rc)

{

int pivotpos = low, pivot = r[low];

for(int i=low+1;i<=high;i++)

if (r[i] < pivot)

{

pivotpos++;

if (pivotpos != i)

{

int t = r[pivotpos];

r[pivotpos] = r[i];

r[i] = r[pivotpos];

rc++;

}

}

r[low] = r[pivotpos];

r[pivotpos] = pivot;

rc++;

return pivotpos;

}

void quicksort(int r[], const int left, const int right, int &rc)

{

if (left < right)

{

int pivotpos = partition(r, left, right, rc);

quicksort(r, left, pivotpos - 1, rc);

quicksort(r, pivotpos + 1, right, rc);

}

}

* 1. 快速排序核心代码

void siftdown(int r[], const int start, const int m, int& rc)

{

int i = start, j = 2 \* i + 1;

int t = r[i];

while (j <= m)

{

if (j < m && r[i] < r[j + 1])

j++;

if (t >= r[j])

break;

else

{

r[i] = r[j];

i = j;

j = 2 \* j + 1;

rc++;

}

}

r[i] = t;

}

int sort6(int r[], int n)//堆排序

{

int count = 0;

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

siftdown(r, i, n - 1, count);

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

{

int t = r[i];

r[i] = r[0];

r[0] = t;

count++;

siftdown(r, 0, i - 1, count);

}

return count;

}

* 1. 归并排序核心代码

void merge(int r[], int r1[],const int left, const int mid, const int right, int& rc)

{

for (int k = left; k < right; k++)

{

r1[k] = r[k];

rc++;

}

int s1 = left, s2 = mid + 1, t = left;

while (s1 <= mid && s2 <= right)

{

if (r1[s1] <= r[s2])

r[t++] = r1[s1++];

else

r[t++] = r1[s2++];

rc++;

}

while (s1 < mid)

{

r[t++] = r1[s1++];

rc++;

}

while (s2 < right)

{

r[t++] = r1[s2++];

rc++;

}

}

void mergesort(int r[],int r1[],int left, int right,int &rc)

{

if (left >= right)

return;

int mid=(right+left)/2;

mergesort(r, r1,left, mid,rc);

mergesort(r, r1,mid + 1, right,rc);

merge(r, r1,left, mid, right, rc);

}

* 1. 基数排序核心代码

for (i = 1; i <= d; i++)

{

for (j = 0; j < 10; j++)

count[j] = 0;

for (j = 0; j < n; j++)

{

k = (data[j] / radix) % 10;

count[k]++;

}

for (j = 1; j < 10; j++)

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

for (j = n - 1; j >= 0; j--)

{

k = (data[j] / radix) % 10;

tmp[count[k] - 1] = data[j];

count[k]--;

}

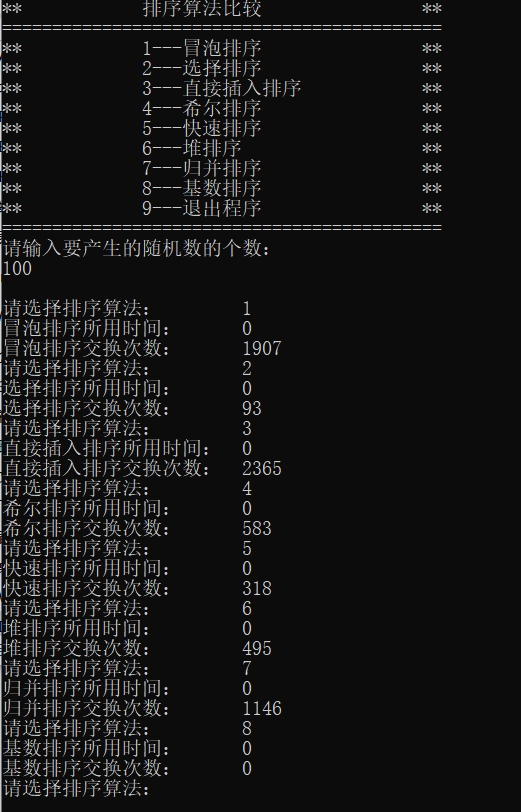
for (j = 0; j < n; j++)

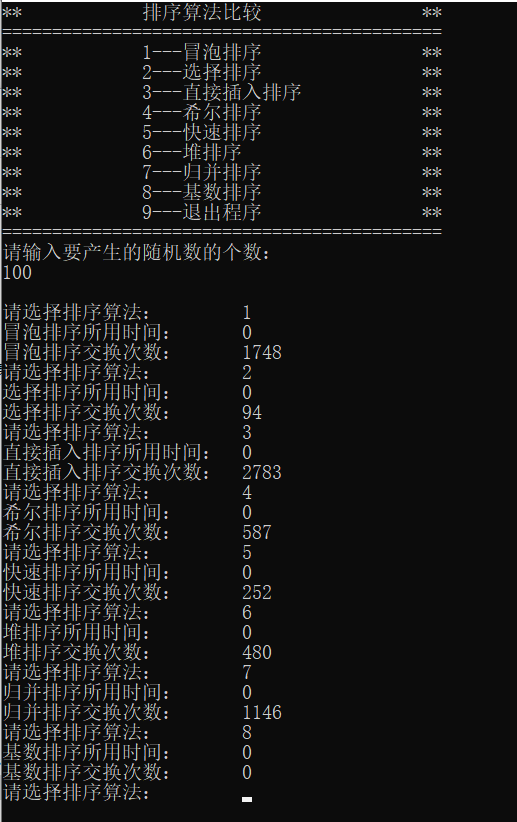
data[j] = tmp[j];

radix = radix \* 10;

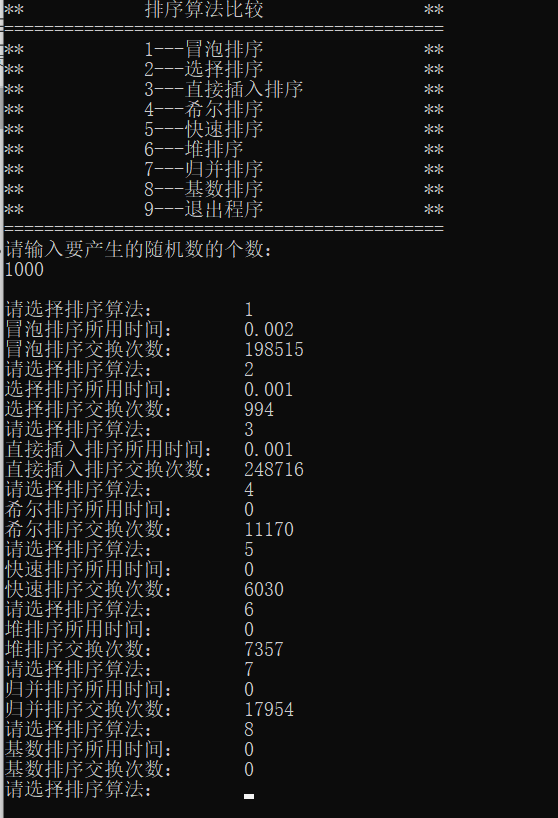
# 2 运行结果

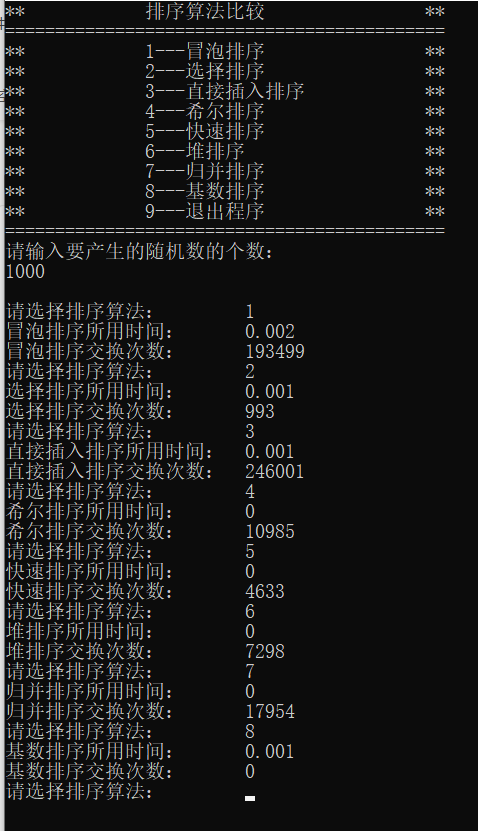
2.1 100个随机数排序比较



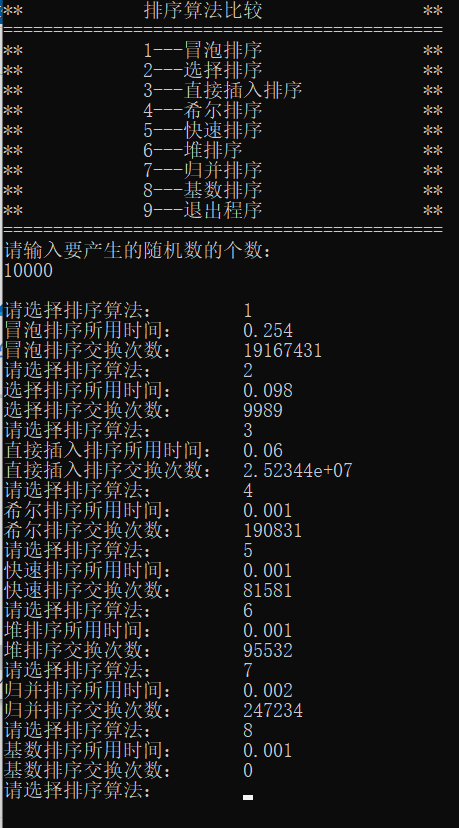


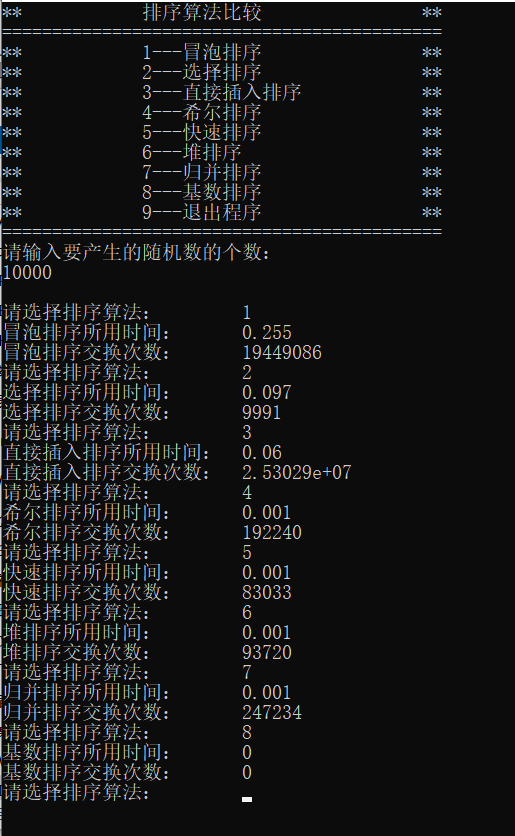
2.2 1000个随机数排序比较



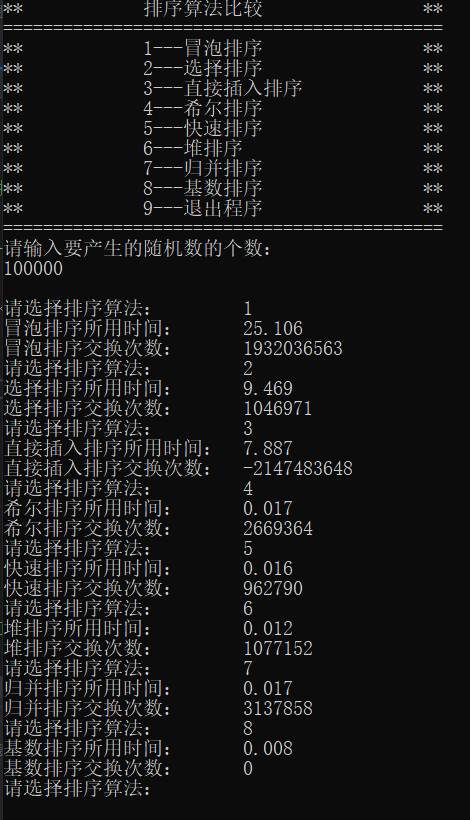


2.3 10000个随机数排序比较





2.4 100000个随机数排序比较



# 3 算法分析

3.1 时间开销分析

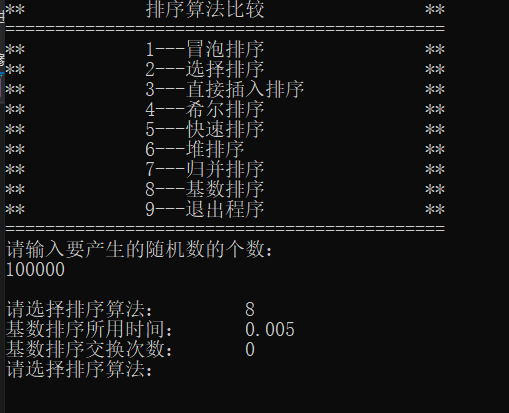
由实验结果可得，在排序数据量不太大的时候（n=100），八种排序算法并不能体现出时间开销上的差异，时间开销都可以忽略不计。当n=1000时，前三种算法和基数排序算法首先体现出时间开销。n=10000时，八种算法的时间开销基本呈现递减趋势，前三种算法的时间开销尤其明显，但相差仍不算太远。当n=100000时，前三种算法的时间开销显著增大，尤其冒泡排序的时间开销相当惊人，此时基数排序算法时间开销最少。

直观分析，直接插进入排序、冒泡排序和直接选择排序适合于规模比较小的元素序列，考虑到三种排序方式还有优化的空间，在n<100000时可以考虑使用三种基本算法。

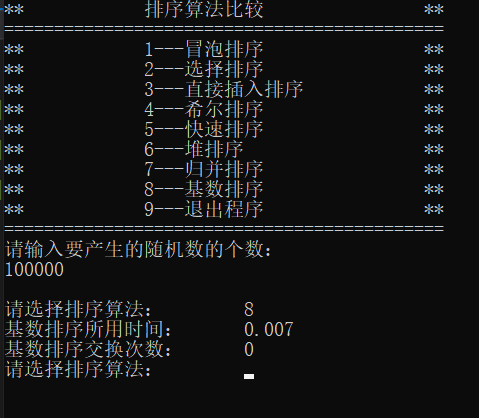
快速排序、希尔排序、堆排序和归并排序在各种情况下都体现出优良的时间开销性能。但是快速排序、希尔排序和堆排序的不稳定性以及归并排序的空间开销大特性，在排序元素数量较少的情况下，应当优先考虑前三种基本排序算法。在排序元素数量较大的时候，如n>100000时，应当考虑这四种排序算法来减少时间开销。

基数排序的时间开销受数据量的影响较前几种算法较小，在n=1000时基数排序就体现出了时间开销 ，但在n=100000时基数排序成为了时间开销最小的排序算法。这也跟本算法测试程序中使用的随机数大小为100000以内有关，数字的位数越大，基数排序的时间开销就会越显著。因此，针对基数排序的这种特性，进一步进行以下实验。

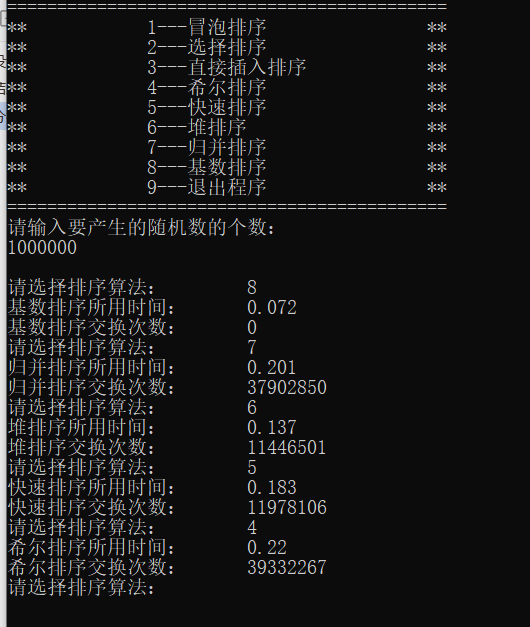
当随机数最大为10000：



当随机数最大为100000：



当随机数最大为1000000：



由以上测验可得出，在随机数的位数不太大（max<100000）时，基数排序在大量数据（n>1000000）的时间开销上明显小于其他算法。因此，在排序元素位数不大而总数很多时，可以考虑基数排序。

3.2 储存开销分析

直接插入排序、冒泡排序和直接选择排序的储存开销都仅有一个辅助元素。

快速排序在最坏的情况下空间复杂度会增加到O（n）。

堆排序基本不需要额外的储存空间。

归并排序的空间复杂度为O（n）。

希尔排序基本不需要额外内存。

基数排序需要的计数器额外内存空间较大。

综合以上分析，对储存空间要求较大的排序算法是快速排序、归并排序和基数排序。

3.3 优缺点总结

对于直接插入排序、直接选择排序和冒泡排序三种基本算法来说，优点是实现简单、空间开销小；缺点是在排序元素数量较大时时间开销明显增大。此外，直接插入排序和冒泡排序具有稳定的优点，但直接选择排序不稳定。

快速排序空间开销较大，但在对较多元素进行排序时时间开销很小。快速排序是不稳定的算法。

堆排序几乎不占用额外空间，并且在排序数量很多时有很强的时间优势，但堆排序同样是不稳定的算法。

归并排序也需要占用较大的储存空间，但它除了在对较多元素进行排序时时间消耗很少，还有稳定的优点。

希尔排序的时间复杂度较低，并且不需要额外的储存空间，它适用于中等规模的排序。希尔排序是不稳定的排序算法。

基数排序比较特殊，它要求排序码不能过多，否则会严重影响时间和空间开销。但是在排序码不多并且元素总数很大的情况下，使用基数排序可以得到很高效的排序结果。

后五种算法的实现较前三种来说较为困难，但总体来说具有了在数据量庞大的情况下高效排序的能力。

不同的算法适应于不同的情况，不能只看重时间开销而忽视算法的本质。比如，在总元素数量很少的情况下，基本排序算法反而具有更大的优势，编程人员可以只进行简单的实现而把更多精力放在更主要的地方。甚至在改进算法的过程中，将基本排序算法加入如快速排序的算法中用于数量较少的分治排序，可以优化算法性能。

可以适用于所有情况的高性能排序算法是不存在的，但通过对各种排序算法的巧妙选择和运用，可以得到特定情况下最优的算法。