**安徽大学集成电路学院**

**《算法设计与分析》第一次报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 算法设计与分析 | | | 日期 |  |
| 专业 | 集成电路设计与集成系统 | 学号 | WB2224186 | 姓名 | 邵宇鹏 |
| 【实验目的】  学习插入排序和二分归并排序的思想，并使用matlab语言进行实现 | | | | | |
| 【实验原理（预习）】  插入排序算法的原理是将数组分为已排序部分和未排序部分。开始时,已排序部分只有一个元素(通常是数组的第一个元素)。取未排序部分的第一个元素,与已排序部分的元素从后向前比较。找到该元素应该插入的位置,将它插入该位置。这需要移动已排序部分的元素来腾出空间。重复步骤2和3,直到未排序部分为空。  插入排序的时间复杂度是O(n^2)。它适用于小规模数据的排序,因为它的原理简单,且在数据基本有序时表现很好。但对于大规模数据,它的性能就会显著下降。  二分归并排序算法的设计思想是，将被排序的数组分成相等的两个子数组，然后使用同样的算法对两个子数组分别排序，最后将两个排好序的子数组归并成一个数组。  归并排序的时间复杂度为O(n log n),它是一种非常高效的排序算法。但它需要额外的空间来存储临时结果,所以对于空间受限的场景可能不太适用。 | | | | | |
| 【实验内容与记录（题号、操作步骤、数据记录与处理、附图编号、代码等）】  matlab代码   |  | | --- | | InsertSort.m | | A =input('请输入一个数组：');  n = length(A);  for j=2:n      x=A(j);      i=j-1;      while i>0 && x<A(i)          A(i+1)=A(i);          i=i-1;          A(i+1)=x;      end  end |  |  | | --- | | Mergesort.m | | function A=Mergesort(A,p,r)      if p<r          q=floor((p+r)/2);          A=Mergesort(A,p,q);          A=Mergesort(A,q+1,r);          A=Merge(A,p,q,r);      end  end |      |  | | --- | | Merge.m | | function A=Merge(A,p,q,r)      x=q-p+1;      y=r-q;      B=zeros(1,x+1);      C=zeros(1,y+1);      for i=1:x          B(i)=A(p+i-1);      end      for j=1:y          C(j)=A(q+j);      end      B(x+1)=inf;      C(y+1)=inf;      i=1;      j=1;      for k=p:r          if B(i)<=C(j)              A(k)=B(i);              i=i+1;          else              A(k)=C(j);              j=j+1;          end      end  end |   在编写好排序算法的代码后，我们需要验证算法编写的是否正确，我重新编写一个m文件如下   |  | | --- | | input.m | | A = [5, 2, 4, 7, 1, 3, 2, 6];  A = Mergesort(A, 1, length(A)) |     得到了正确的结果 | | | | | |
| 【小结与讨论】  通过本次实验,我对插入排序和二分归并排序这两种经典的排序算法有了更深入的了解。  **插入排序**是一种简单直观的排序算法,它通过构建有序序列,对于未排序数据,在已排序序列中从后向前扫描,找到相应位置并插入。它的时间复杂度为,在数据量较小或基本有序的情况下效率较高。但是对于大规模无序数据,插入排序的效率会大大降低。  **二分归并排序**则是一种基于分治策略的高效排序算法。它通过将待排序序列递归地分为越来越小的子序列,直到每个子序列只有一个元素,然后再通过合并有序子序列的方式实现最终排序。这种算法的时间复杂度为,在大规模无序数据情况下效率更高。但它需要额外的空间来存储临时结果,空间复杂度较高。  通过对这两种算法的对比分析,我认为在实际应用中应该根据具体情况选择合适的排序算法。对于数据量较小或基本有序的情况,插入排序可能是更好的选择;而对于大规模无序数据,二分归并排序通常能提供更好的性能。此外,还可以考虑其他高效排序算法,如快速排序、堆排序等,根据不同应用场景选择最优的方案。  总的来说,本次实验加深了我对经典排序算法的理解,为今后的算法设计与分析工作打下了良好的基础。 | | | | | |