计算机导论

中国科学技术大学
University of Science and Technology of China

归并排序

- 给定一个数列，将其排序
- 排序问题的分治法求解
  - 将待排序数列一分为二，每个子数列分别排序
  - 将完成排序的两个子数列合并

```
function mergesort(a[1...n])
Input:  An array of numbers a[1...n]
Output:  A sorted version of this array

if n > 1:
  return merge(mergesort(a[1...⌊n/2⌋]), mergesort(a[⌊n/2⌋+1...n]))
else:
  return a
```

  - 问题：如何合并？merge()的算法

- 考虑合并两个排序的数组 $x[1...k]$和 $y[1...l]$，结果存入 $z[1...k+l]$ 中

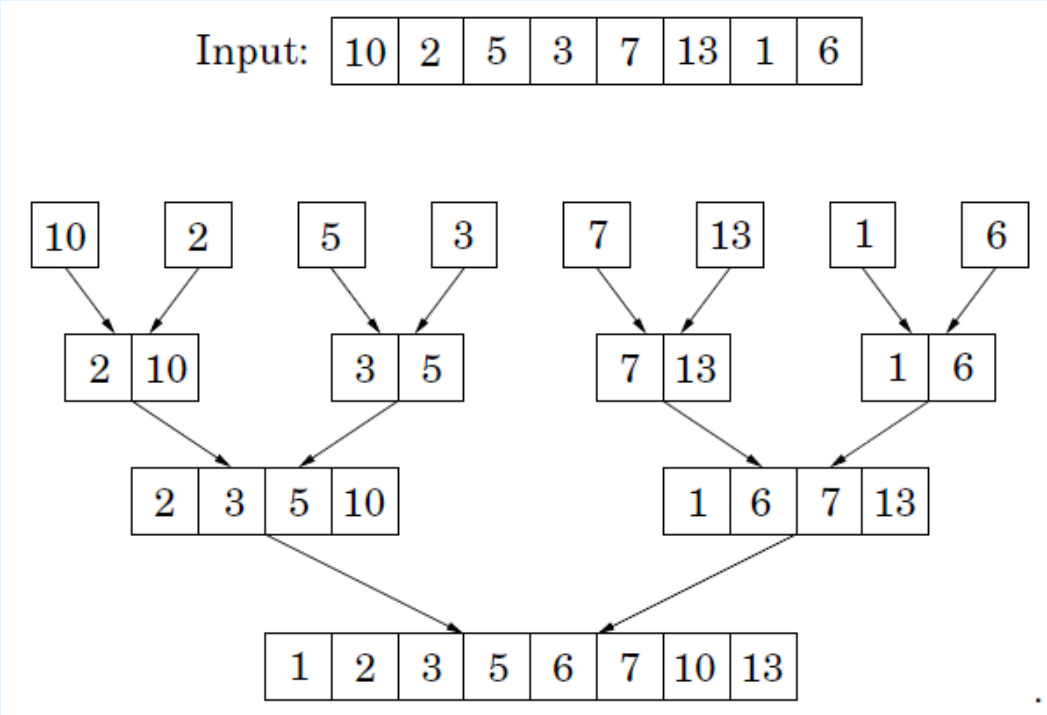
```
function merge(x[1...k], y[1...l])
if k = 0:   return y[1...l]
if l = 0:   return x[1...k]
if x[1] ≤ y[1]:
  return x[1] ∘ merge(x[2...k], y[1...l])
else:
  return y[1] ∘ merge(x[1...k], y[2...l])
```

“∘”表示元素与数列连接

- 这个算法的执行时间是线性的，即O(*k+l*)
- mergesort算法的执行时间满足 $T(n) = 2T(n/2) + O(n)$

  算法执行时间为 $O(n \log n)$

- 一个mergesort算法执行实例



- 观察算法，发现所有改变元素位置的操作都在合并中完成。两个元素合并为一个二元数组，然后二元数列合并为四元数数组，…。
- 算法可以迭代实现

- 1945年，冯·诺依曼提出归并排序算法

- O(nlogn),空间复杂度O(n),稳定的排序

冯·诺依曼（John von Neumann）（1903-1957）

- 另一种迭代形式的算法描述

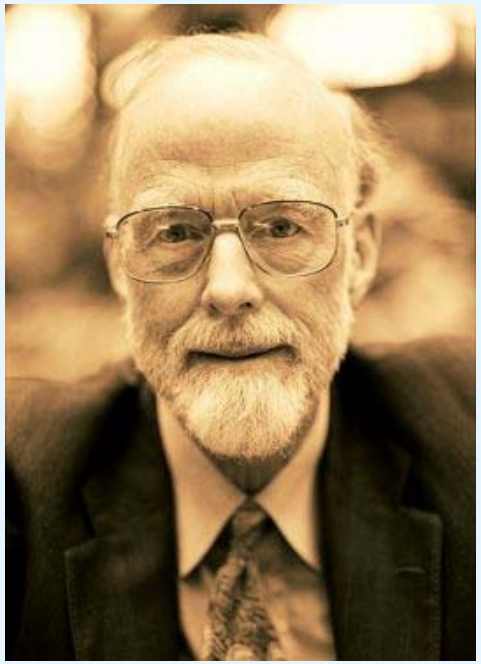
```
function iterative-mergesort(a[1...n])
Input:  elements a_1, a_2, ..., a_n to be sorted

Q = [ ] (empty queue)
for i = 1 to n:
    inject(Q, [a_i])
while |Q| > 1:
    inject(Q, merge(eject(Q), eject(Q)))
return eject(Q)
```

  - inject是将一个元素添加到队列Q的尾部，而eject返回并移出Q头部的一个元素
  - 不断从队列头部取出两个数组合并，将合并的结果放到队尾

快速排序

- 1960年，由Tony Hoare提出

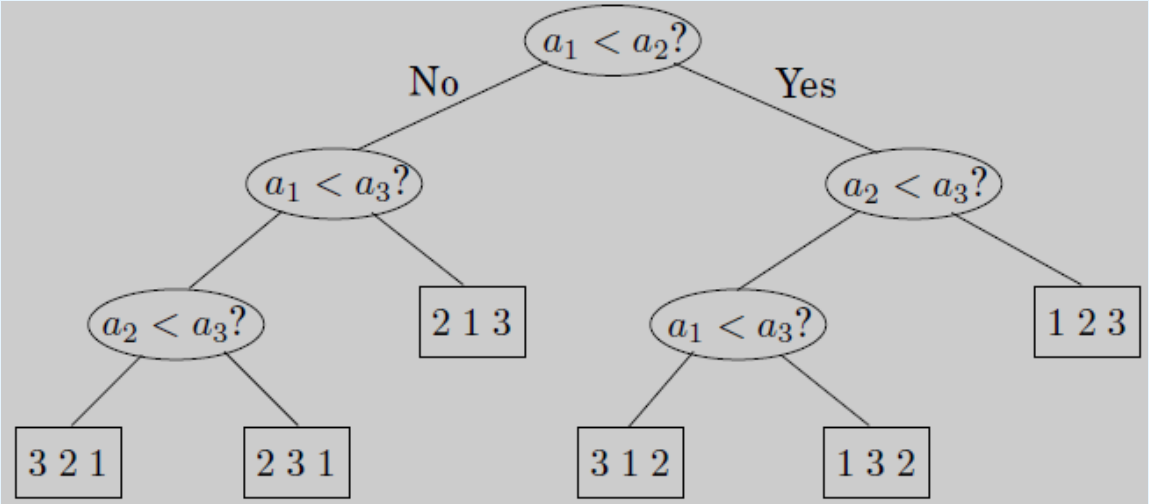
- 2009年3月他发表了题为“Null引用：代价十亿美元的错误”的演讲，回忆自己1965年设计第一个全面的类型系统时，未能抵御住诱惑，加入了Null引用，仅仅是因为实现起来非常容易。它后来成为许多程序设计语言的标准特性，导致了数不清的错误、漏洞和系统崩溃，可能在之后40年中造成了十亿美元的损失。他在同月出版Communications of the ACM中表示，如何证明程序的正确性仍然是计算机科学中有待解决的重大课题。



Tony Hoare或者C. A. R. Hoare (1934—)

最坏的情况

- 排序算法可用树来表示。例如对数组a_1, a_2, a_3排序。
  - 首先比较a_1和a_2，如果a_1较大，则比较a_1与a_3，否则比较a_2与a_3。树结构表示如下



- 最终的排序结果总是位于树的叶子节点，而从根到叶子的最长路径上节点的个数就是最坏情况下必需的比较次数
  - 此例中为3
- n个元素排序对应的树有n!个叶子节点（n!种排列），由于树是二叉树，则树的高度$d=\log_2 n!$。由于

$$n! \approx \sqrt{\pi\left(2n+\frac{1}{3}\right)} \cdot n^n \cdot e^{-n}$$

则$\log_2 n! \approx n\log_2 n$。所以任何排序算法至少比较$\Omega(n\log n)$次。

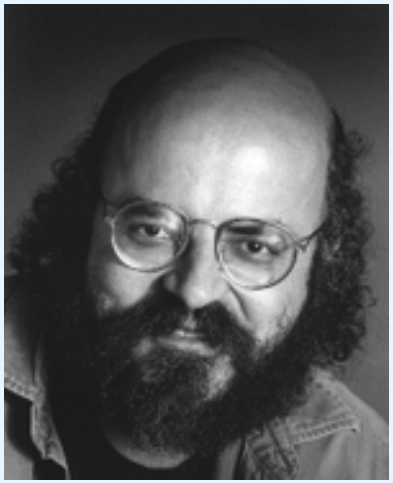
- 换言之，归并排序是最优的排序算法。

算法研究的两个方向

- 优化
  - 寻找更好的算法
  - 设计技巧
  - 一个新的算法（上界）
- 可能性
  - 说明难以得到更好的算法
  - 证明技巧
  - 对问题的更好认识（下界）

❖ Gates, William H. and Christos H. Papadimitriou.
Bounds for sorting by prefix reversal.
Discrete Mathematics 27 (1979), 47--57.

Harvard University(1973)
Microsoft (1975)



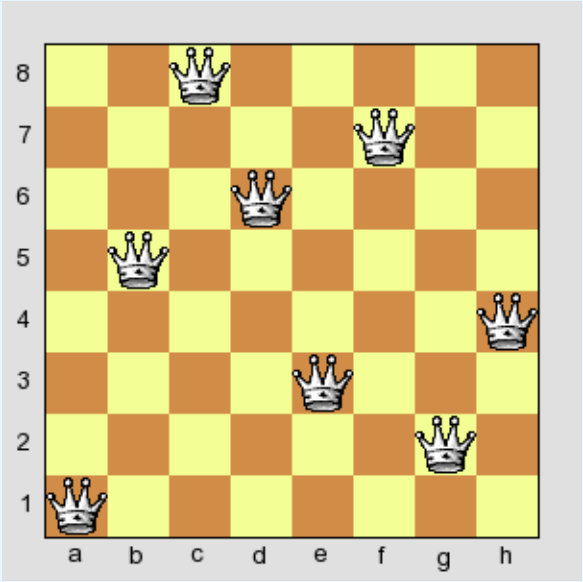
Princeton University
(MS 1974 and PhD 1976)

上界与下界

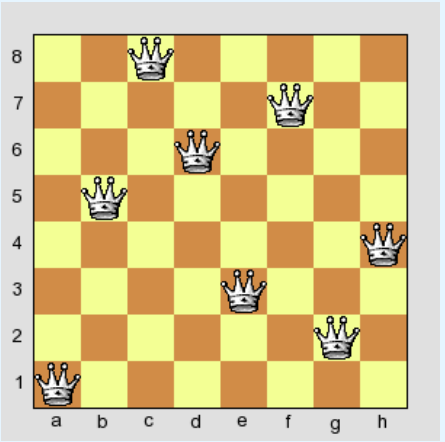
- 问题描述： 仅使用前缀翻转（prefix reversal）操作对n个大小不同的序列排序。
- 前缀翻转： 将包含首个元素的子序列进行翻转
- 结果：
  - 给出算法，证明至多(5n+5)/3 次操作可以排序完成
  - 给出例子，证明17n/16次操作无法完成排序
- 改进：
  - 1995年，新的下界结果

问题：8皇后问题

- 一个古老而著名的问题，是回溯算法的典型例题。
- 在8×8格的国际象棋上摆放八个皇后，使其不能互相攻击，即任意两个皇后不能处于同一行、同一列或同一斜线，问有多少种摆法。



问题：8皇后问题

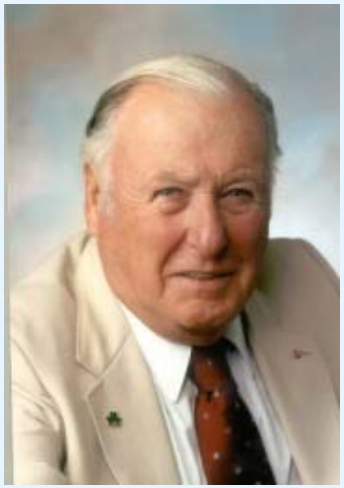


- 1850年，高斯认为有76种方案。
- 1854年在柏林的象棋杂志上不同作者发表了40种不同的解。
- 之后有人用图论的方法解出92种结果。

Carolus Fridericus Gauss
(1777—1855)

快速傅里叶变换（FFT）

- Cooley, J.W., and J.W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, Vol. 19, Apr. 1965



James William Cooley
(1926—2016)

John Wilder Tukey
(1915—2000)

Carolus Fridericus Gauss
(1777—1855)

- 1805年，高斯独立提出了FFT

编程实现

- 理解问题的内在复杂度
- 思考如何分解问题
- 使用已有的高效算法（程序）

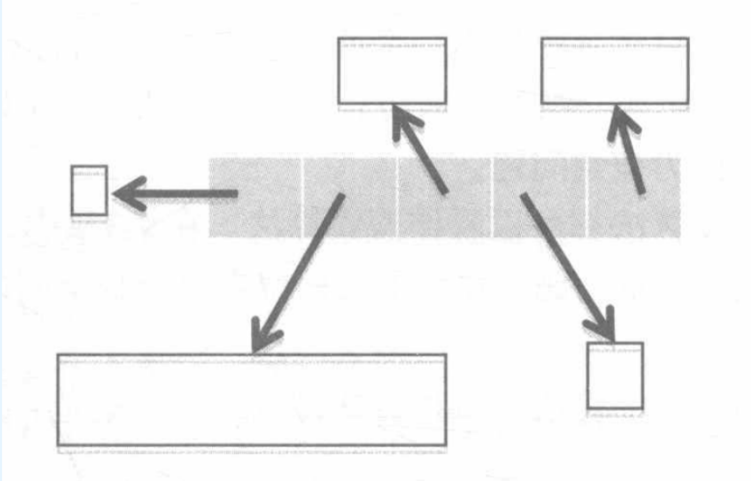
```
def search(L, e):
    """假设L是列表
       如果e是L中的元素，则返回True，否则返回False"""
```

线性查找

```
for i in range(len(L)):
    if L[i] == e:
        return True
return False
```

线性查找

- 简单情况：列表中元素是整数（固定字节数）
- 一般情况：列表中元素任意



“计算中的任何问题都可以通过添加另一个间接层解决。”

二分查找

- 二分查找是著名的分治法算法
- 在一个排序的序列z[0,1,...,n-1]中查找元素k（比如在拼音排序的名册中找某个名字）。
  - 比较k和元素z[n/2]，根据结果
    - k>z[n/2]，对序列z[0,1,...,n/2-1]使用二分法，
    - k<z[n/2]，对序列z[n/2,...,n]使用二分法
    - k=z[n/2]，找到 $T(n) = T\left(\lceil n/2 \rceil\right) + O(1)$

- 执行时间

$$T(n) = O(\log n)$$

二分查找和利用假设（信息）

```
def search(L, e):
    """假设L是列表，其中元素按升序排列。
            ascending order.
       如果e是L中的元素，则返回True，否则返回False"""
    for i in range(len(L)):
        if L[i] == e:
            return True
        if L[i] > e:
            return False
    return False
```

二分查找和利用假设（信息）

```
def search(L, e):
    """假设L是列表，其中元素按升序排列。
       ascending order.
      如果e是L中的元素，则返回True，否则返回False"""

    def bSearch(L, e, low, high):
        #Decrements high - low
        if high == low:
            return L[low] == e
        mid = (low + high)//2
        if L[mid] == e:
            return True
        elif L[mid] > e:
            if low == mid: #nothing left to search
                return False
            else:
                return bSearch(L, e, low, mid - 1)
        else:
            return bSearch(L, e, mid + 1, high)

    if len(L) == 0:
        return False
    else:
        return bSearch(L, e, 0, len(L) - 1)
```

排序算法

- L.sort()
- sorted(L)

- 选择排序
- 归并排序

C语言里的函数指针

可以用一个原型匹配的函数的名字给一个函数指针赋值。要通过函数指针调用它所指向的函数，写法为：

函数指针名(实参表)；

下面的程序说明了函数指针的用法

```
#include <stdio.h>
void PrintMin(int a,int b)
{
    if( a<b )
        printf("%d",a);
    else
        printf("%d",b);
}
int main(){
    void (* pf)(int ,int);
    int x = 4, y = 5;
    pf = PrintMin;   pf(x,y);
    return 0;
}
```

上面的程序输出结果是：

4

函数指针应用：快速排序库函数qsort

```
void qsort(void *base, int nelem, unsigned int width,
    int ( * pfCompare)( const void *, const void *));
```

base是待排序数组的起始地址，
nelem是待排序数组的元素个数，
width是待排序数组的每个元素的大小（以字节为单位）
pfCompare是一个函数指针，它指向一个“比较函数”。该比较函数应是返回值为int,有两个参数为const void * 的函数

```
int 函数名(const void * elem1, const void * elem2);
```

快速排序库函数qsort

排序就是一个不断比较并交换位置的过程。qsort如何在连元素的类型是什么都不知道的情况下，比较两个元素并判断哪个应该在前呢？答案是，qsort函数在执行期间，会通过pfCompare指针调用“比较函数”，调用时将要比较的两个元素的地址传给“比较函数”，然后根据“比较函数”返回值判断两个元素哪个更应该排在前面。

这个“比较函数”不是C/C++的库函数，而是由使用qsort的程序员编写的。在调用qsort时，将“比较函数”的名字作为实参传递给pfCompare。程序员当然清楚该按什么规则决定哪个元素应该在前，哪个元素应该在后，这个规则就体现在“比较函数”中。

快速排序库函数qsort

qsort函数的用法规定，“比较函数”的原型应是：

```
int 函数名(const void * elem1, const void * elem2);
```

该函数的两个参数，elem1和elem2，指向待比较的两个元素。也就是说， * elem1和 * elem2就是待比较的两个元素。该函数必须具有以下行为：

1）如果 * elem1应该排在 * elem2前面，则函数返回值是负整数（任何负整数都行）。

2）如果 * elem1和* elem2哪个排在前面都行，那么函数返回0

3）如果 * elem1应该排在 * elem2后面，则函数返回值是正整数（任何正整数都行）。

下面的程序，功能是调用qsort库函数，将一个unsigned int数组按照个位数从小到大进行排序。比如8，23，15三个数，按个位数从小到大排序，就应该是23，15，8

```
#include <stdio.h>
#include <stdlib.h>
int MyCompare( const void * elem1,
               const void * elem2 )
{
    unsigned int * p1, * p2;
    p1 = (unsigned int *) elem1;
    p2 = (unsigned int *) elem2;
    return  (* p1 % 10)  - (* p2 % 10 );
}
```

```
#define NUM 5
int main()
{
    unsigned int an[NUM] = { 8,123,11,10,4 };
    qsort( an,NUM,sizeof(unsigned int),
         MyCompare);
    for( int i = 0;i < NUM; i ++ )
         printf("%d ",an[i]);
    return 0;
}
```

上面程序的输出结果是：

10 11 123 4 8

思考题: 如果要将an数组从大到小排序，
那么MyCompare函数该如何编写？

例题：单词排序

输入若干行单词（不含空格），请按字典序排序输出。
大小写有区别。单词一共不超过100行，每个单词不超过20字符

Sample input:

What
man
Tell
About
back

Sample output:

About
Tell
What
back
man

例题：单词排序

用什么保存多个单词？

答：用二维字符数组

char Word[100][30];
则表达式 Word[i] 的类型就是 char *
Word[i] 就是数组中的一行，就是一个字符串
Word[i][0] 就是Word[i] 这个字符串的头一个字符

二维字符数组也能初始化，例如：

```
char week[7][10]={"Saturday", "Sunday", "Monday",
   "Tuesday", "Wednesday", "Thursday", "Friday"};
```

例题： 单词排序

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char Word[100][30];
int MyCompare( const void * e1, const void * e2 )
{
        return strcmp( (char * ) e1, (char * ) e2 );
}
int main()
{
        int i, n = 0; //单词个数
        while(scanf("%s",Word[n]) != EOF && Word[n][0])
                n ++;
        qsort(Word, n,sizeof(Word[0]),MyCompare);
        for( i = 0; i < n; i ++ )
                printf("%s\n",Word[i]);
        return 0;
}
```

为了对付有可能最后一行读入的是空行

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char Word[100][30];
int MyCompare( const void * e1, const void * e2 )
{
    return strcmp( (char * ) e1, (char * ) e2 );
}
int main()
{
    int i, n = 0; //单词个数
    while(scanf("%s",Word[n]) != EOF && Word[n][0])
        n ++;
    qsort(Word, n,sizeof(Word[0]),MyCompare);
    for( i = 0; i < n; i ++ )
        printf("%s\n", Word[i]);
    return 0;
}

```

感谢关注~！

中国科学技术大学
University of Science and Technology of China