目录

湖南工商大学 算法导论 课程教案

授课题目（教学章、节或主题） **课时安排: 2学时**

第一讲：算法基础知识 **授课时间: 第一周周一第1、2节**

教学内容（包括基本内容、重点、难点）：

基本内容： （1）常用的算法：插入排序、归并排序。

（2）算法分析：最坏情况；平均情况分析；最好时间分析。

（3）算法设计：归并排序算法设计。

教学重点、难点：重点为算法时间复杂度分析、归并排序算法；难点为归并排序算法时间分析。

教学媒体的选择：本章使用大数据分析软件Jupyter教学，Jupyter集课件、Python程序运行、HTML网页制作、Pdf文档生成、Latex文档编译于一身，是算法导论课程教学的最佳选择。

板书设计：一大打黑板分为上下两块，第一块基本定义，推导证明以及例子放在第二块。第一块整个课堂不擦洗，以便学生随时看到算法流程图以及基本算法理论等内容。

课程过程设计： （1）讲解基本算法理论； （2）举例说明； （3）程序设计与编译； （4）对本课堂进行总 结、讨论； （5）布置作业与实验报告

第一讲 算法基础知识

1.1 算法

算法: 任何一个良定义的计算过程

$$\text{input} \rightarrow \text{算法} \rightarrow \text{output}$$

例如: 排序问题

$$\text{input} \rightarrow \text{n 个数序列} \rightarrow a_1, a_2, \cdots a_n$$

$$\text{output} \rightarrow \text{找一个重排} \rightarrow 1, 2, 3, \cdots n$$

计算步骤（算法）： 如何达到上述关系?

不可解问题: AlanTuring,Oncomputablenumbers,with anapplication to the Entscheidungs problem,（论可计算数及其在判定问题中的应用）Proceedings of the London Math. Society, Series 2,42 (1936),pp230-265.

在这篇划时代的论文中，图灵提出了图灵机的概念，给出了停机问题的定义并且证明了它是不可解问题.

图灵停机问题的描述： 不存在一个程序（或算法），它能够计算任何程序在给定输入上是否会结束（停机）.

1. 停机问题就是判断任意一个程序是否会在有限的时间之内结束运行的问题.
2. 如果这个问题可以在有限的时间之内解决，那么就可以有一个程序判断其本身是否会停机.
3. 在 程序停止之前，没有办法判断它会不会停止.所以这是一个不可解的问题.

2.1 插入排序

输入: 序列 A 的 n 个元素 $A[1], A[2], \ldots \ldots A[n]$

输出: 序列 A 的从小到大的排列

插入排序算法: 从后往前枚举已有序部分来确定插入位置.

例 1: 有一个序列 $A = \{5, 2, 4, 6, 3, 1\}$

第一步：当前已有序$\{5\}$，需要把 $A[2]=2$ 插入一有序部分中。

5 2 4 6 1 3

显然插入位置为1号位，插入后变为 $\{2，5\}$

2 5 4 6 1 3

第二步：当前有序部分为 $\{2，5\}$，需要把 $A[3]=4$ 插入已有序列。为2号位，插入后变为 $\{2，4，5\}$

2 4 5 6 1 3

第三步：当前有序部分为 $\{2，4，5\}$，需要把 $A[4]=6$ 插入已有序列。为4号位，插入后变为 $\{2，4，5，6\}$

2 4 5 6 1 3

第四步：当前有序部分为 $\{2，4，5，6\}$，需要把$A[5]=1$ 插入已有序列。为1号位，插入后变为 $\{1，2，4，5，6\}$

1 2 4 5 6 3

第五步：当前有序部分为 $\{1，2，4，5，6\}$，需要把$A[6]=3$ 插入已有序列。为3号位，插入后变为 $\{1，2，3，4，5，6\}$

1 2 3 4 5 6

下面的Python代码给出了具体实现：

In [11]:

```python
#插入排序python程序1

array = [5,2,4,6,3,1]

 # insert_sort
for i in range(1, len(array)):
    if array[i - 1] > array[i]:
        temp = array[i]      # 当前需要排序的元素
        index = i            # 用来记录排序元素需要插入的位置
        while index > 0 and array[index - 1] > temp:
            array[index] = array[index - 1]      # 把已经排序好的元素后移一位，
            index -= 1
        array[index] = temp # 把需要排序的元素，插入到指定位置

 # print sort result.
print(array)

```

[1, 2, 3, 4, 5, 6]

2.2 分析算法

目的: 估算算法所需的资源（时间、空间、带宽）

计算模型: 单处理机 RAM

涉及知识: 时间分析
1.算法耗费时间：输入的大小；实例的构成
2.运行时间
3.最坏时间分析：对任何输入的最坏运行时间
4.平均时间分析：输入等概率；或随机化

插入排序的运行时间 $T[n]$:

$$T[n] = C_1 * n + C_2 * (n-1) + C_3 * (n-1) + C_4 * \sum_{i=2}^{n} t_i$$

$$+C_5 * \sum_{i=2}^{n} (t_i - 1) + C_6 * \sum_{i=2}^{n} (t_i - 1) + C_7 * (n-1)$$

注： 1. C_i 表示第 i 行的每次执行需要时间；
2. t_i 表示对那个值i执行while循环测试的次数;
3. 最佳情况：输入数组已排好序，从而 $t_i = 1$;
4. 最坏情况：输入数组反向排序，从而 $t_i = i$。

插入排序的运行时间 $T[n]$:

1. 最佳情况：输入数组已排好序，从而 $t_i = 1$

$$T[n] = C_1 * n + C_2 * (n-1) + C_3 * (n-1) + C_4 * (n-1) + C_7 * (n-1)$$
$$= a * n + b$$

2. 最坏情况：输入数组反向排序，从而 $t_i = i$

$$T[n] = C_1 * n + C_2 * (n-1) + C_3 * (n-1) + C_4 * \left(\frac{n(n+1)}{2} - 1\right)$$
$$+C_5 * \frac{n(n-1)}{2} + C_6 * \frac{n(n-1)}{2} + C_7 * (n-1)$$
$$= a * n^2 + b * n + c$$

例 2 求解 n 的阶乘 $n! = 1 * 2 * \cdots * n$.

解： 递归式 $F(n) = F(n-1) * n$
递归边界 $F(0) = 1$.

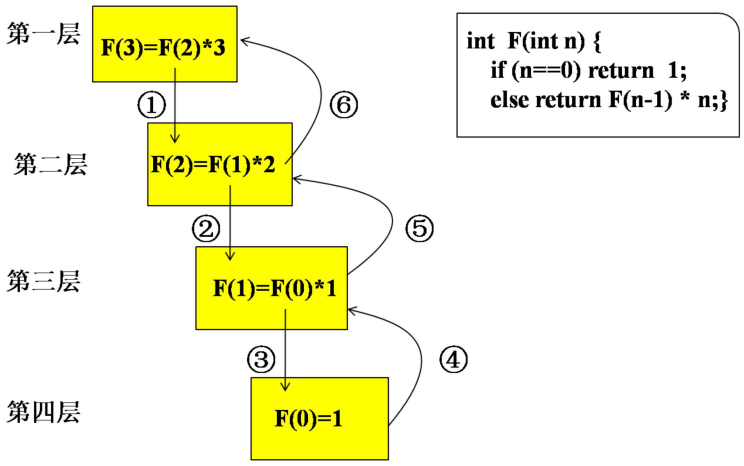


图2.1 递归求解3的阶乘的过程示意图

In [ ]:

```c
#C 语言程序
#include <stdio.h>
Int  F(int n) {
      if (n==0) return 1;           //当到达递归边界，返回 F(0) ==1
      else  return  F(n-1) * n;     //没有达到递归边界 ，继续递归
}
int  main ( ) {
      int  n;
      scanf(“%d”, & n);
      printf (“%d\n”, F(n));
      return  0;}
```

In [1]:

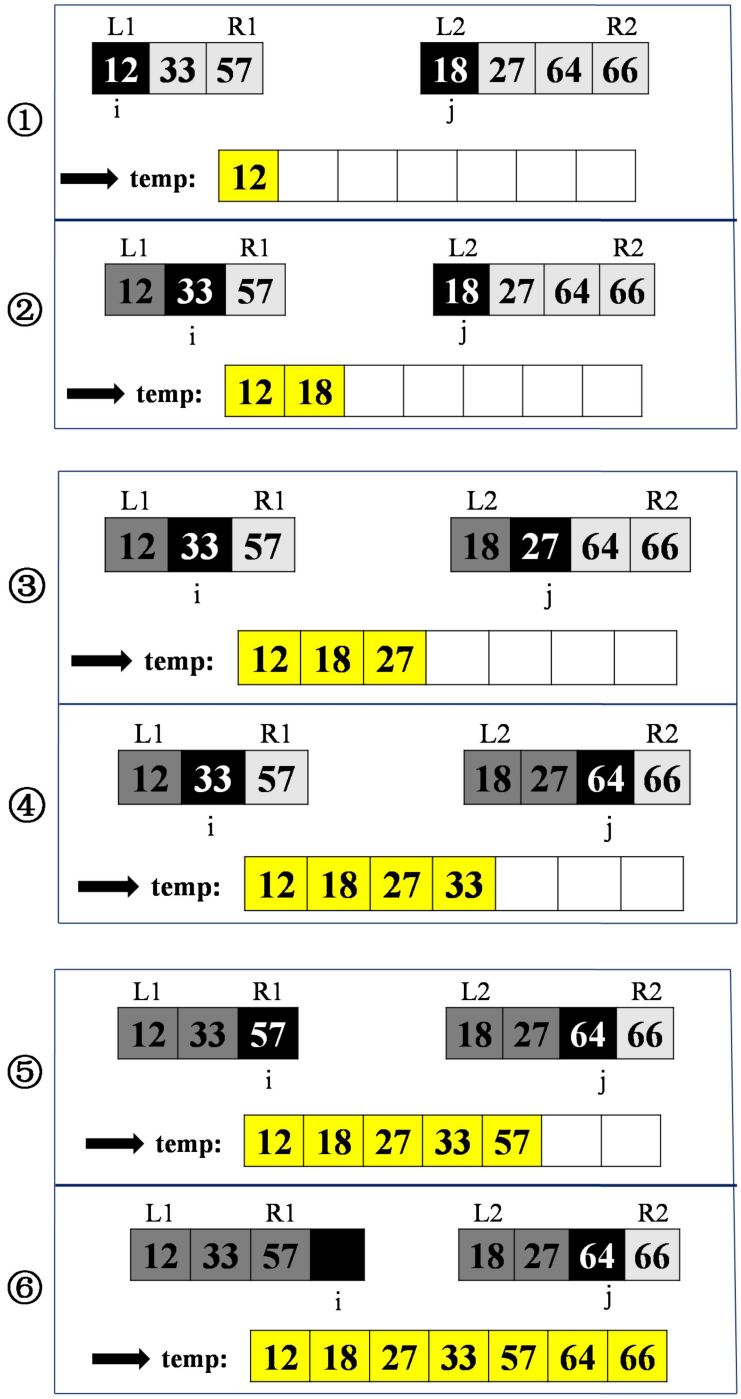
```python
#阶乘的Python程序
import math      #下面使用math库，要先导入数学库
n = int(input())
#print(math.factorial(n)) #math库里有factorial(n)函数可直接求解n的阶乘，可用于
fact = 1
for i in range(1,n+1):    #range()默认从0开始，这里range(1,n+1)表示从1 开始取
    fact = fact * i
print(fact)
```

```
10
3628800
```

3.1 归并排序算法

归并排序（MERGE-SORT）是利用归并的思想实现的排序方法，该算法采用经典的分治（divide-and-conquer）策略（分治法将问题分(divide)成一些小的问题然后递归求解，而治(conquer)的阶段则将分的阶段得到的各答案"修补"在一起，即分而治之)。归并排序是稳定排序，它也是一种十分高效的排序。

例 3 将 $\{12, 33, 57\}$ 与 $\{18, 27, 64, 66\}$合并



In [ ]:

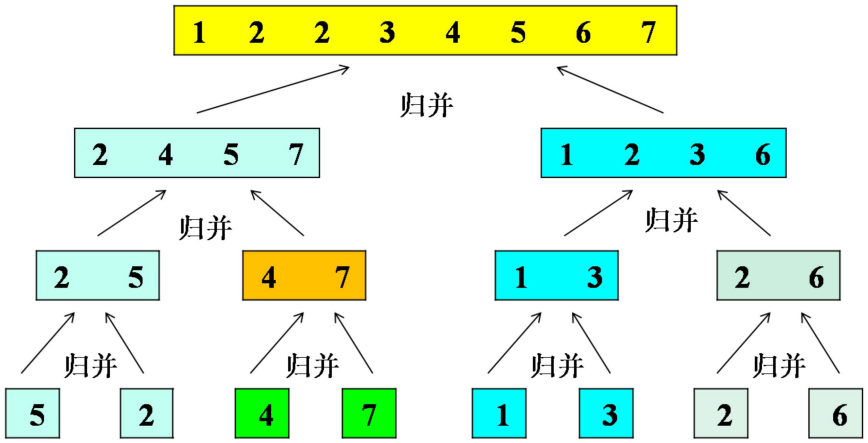
```
#归并 C 语言程序
归并排序的递归：
void  mergeSort (int  A[ ],  int  left,  int  right )  {
   if ( left < right )  {
        int  mid = ( left + right )/2;                    //取[ left,  right ] 中
     mergeSort (A,  left,  mid); //递归，左子区间归并排序
     mergeSort (A,  mid+1,  right);  //右子区间归并排序
     merge (A,  left,  mid,  mid + 1,  right);//左右区间合并
       }}
```

In [25]:

```python
#归并排序python程序
# 记录归并排序
def MergeSort(lists):
    if len(lists) <= 1:
        return lists
    middle = len(lists)//2
    left = MergeSort(lists[:middle])
    right = MergeSort(lists[middle:])
    return merge(left, right)


def merge(a, b):
    c = []
    h = j = 0
    while j < len(a) and h < len(b):
        if a[j] < b[h]:
            c.append(a[j])
            j += 1
        else:
            c.append(b[h])
            h += 1
    if j == len(a):
        for i in b[h:]:
            c.append(i)
    else:
        for i in a[j:]:
            c.append(i)

    return c

if __name__ == '__main__':
    a = [5,2,4,7,1,3,2,6]
    print(MergeSort(a))
```

```
[1, 2, 2, 3, 4, 5, 6, 7]
```

例 4 归并排序 $A = \{5,2,4,7,1,3,2,6\}$



归并排序算法的分析：

1. 分解：计算子数组的中间位置，运行时间

$$D(n) = \Theta(1)$$

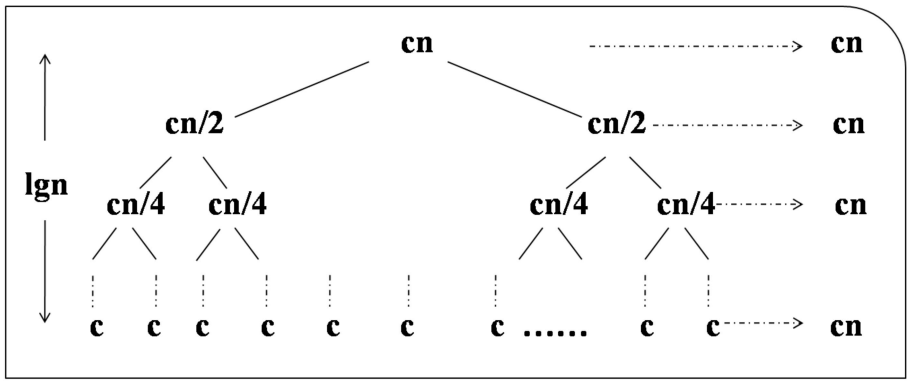
2. 归并：递归求解两个规模为n/2的子问题,将贡献 $2T(n/2)$ 的运行时间；

3. 合并：合并为一个n个元素的过程merge函数, 将贡献 $\Theta(n)$ 的运行时间。

归并排序的最坏运行时间 $T(n)$ 的递归式：

$$T(n) = \begin{cases} \Theta(1) & \text{若 } n = 1 \\ 2T(n/2) + \Theta(n) & \text{若 } n > 1 \end{cases}$$

由主定理，可得 $T(n) = \Theta(nlgn)$,



课后练习

1. 写出归并排序的Python代码或 C语言代码。

2. 用归并排序代码实现对数组 $A = 3, 41, 52, 26, 38, 57, 9, 49$ 进行排序。

3. 使用图示，说明上面数组A归并排序的操作过程。

讨论、思考题、作业： 第22页 练习 2.3.1, 2.3.3

参考资料（含参考书、文献等）：算法笔记. 胡凡、曾磊，机械工业出版社，2016.

授课类型（请打√）：理论课□ 讨论课□ 实验课□ 练习课□ 其他□

教学过程设计（请打√）：复习□ 授新课□ 安排讨论□ 布置作业□

教学方式（请打√）：讲授□ 讨论□ 示教□ 指导□ 其他□

教学资源（请打√）：多媒体□ 模型□ 实物□ 挂图□ 音像□ 其他□

填表说明：1、每项页面大小可自行添减；2、教学内容与讨论、思考题、作业部分可合二为一。