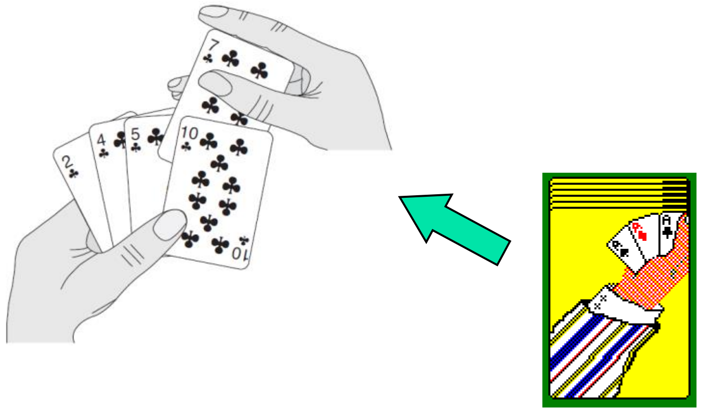
2 Framework for analyzing algorithms

2.3 Designing algorithms

Many ways to design algorithms（模拟、分治、剪枝、计数、回溯、贪心、...）

Sorting problem

- Bubble sort: bubbling
- Insertion sort: incremental approach （增量靠近）
- Merge sort: divide-and conquer（特定位置，分而治之）
- Quick sort: location（精准定位，分而治之）



……

- The worst-case running time of divide-and-conquer algorithms is much less than that of insertion sort.（基于分治策略的排序算法比插入排序好）

2.3.1 The divide-and-conquer approach

Divide-and-conquer approach（分而治之，各个击破）

Recursive : call themselves recursively one or more times to deal with closely related subproblems.

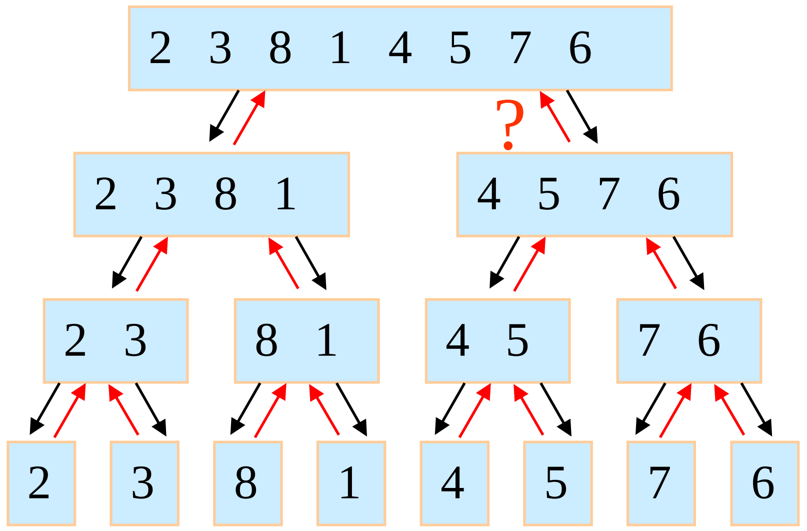
- **Divide** the problem into a number of subproblems.
- **Conquer** the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner. （递归求解子问题，若子问题规模小，则可直接求解）
- **Combine** the sloutions to the subproblems into the solution for the original problem.

2.3.1 The divide-and-conquer approach

Merge sort algorithm

- **Divide:** divide the *n*-element sequence to be sorted into two subsequences of *n*/2 elements each. (Exercise: how about four subsequences, *n*/4 elements each?)
- **Conquer:** merge sort the two subsequences recursively. （用递归方式，进行子问题求解）
- **Combine:** merge the two sorted subsequences to produce the sorted answer.

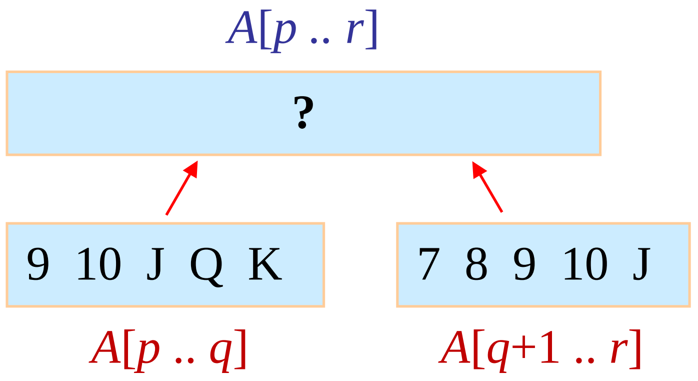
The recursion "bottoms out" when the sequence to be sorted has length 1, in which case there is no work to be done, since every sequence of length 1 is already in sorted order.
(子序列长度为1时，停止递归)

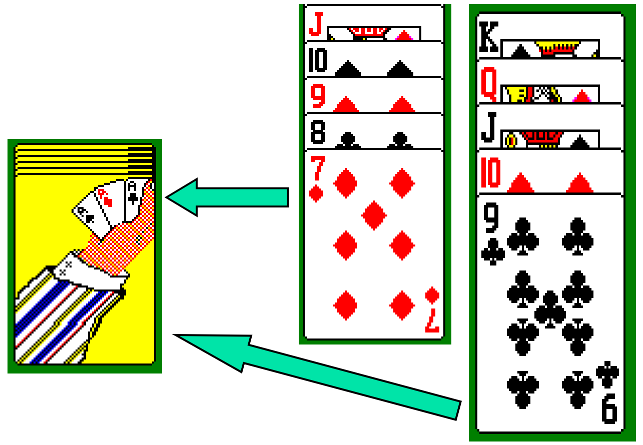


2.3.1 The divide-and-conquer approach

MERGE(A, p, q, r)**:** the key operation of the MERGE sort algorithm, the merging of two sorted sequences in the "combine" step.

- A is an array and p, q, and r are indices numbering elements of the array such that $p \leq q < r$.
- The procedure assumes the subarrays $A[p .. q]$ and $A[q+1 .. r]$ are in sorted order. The procedure merges them to form a single sorted subarray that replaces the current subarray $A[p .. r]$.
- Working step:

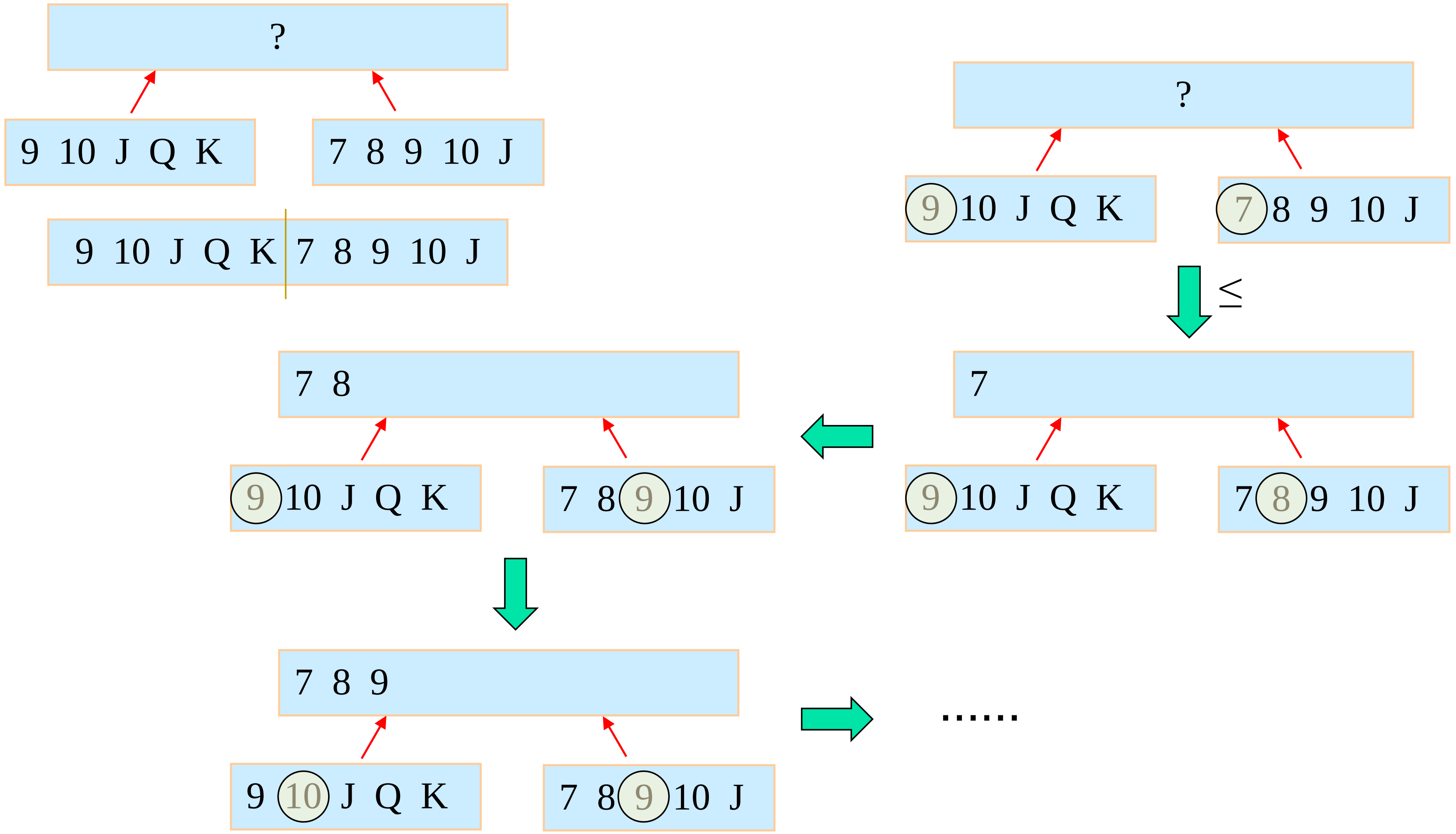




The procedure takes time $\Theta(n)$, $n = r\text{-}p\text{+}1$

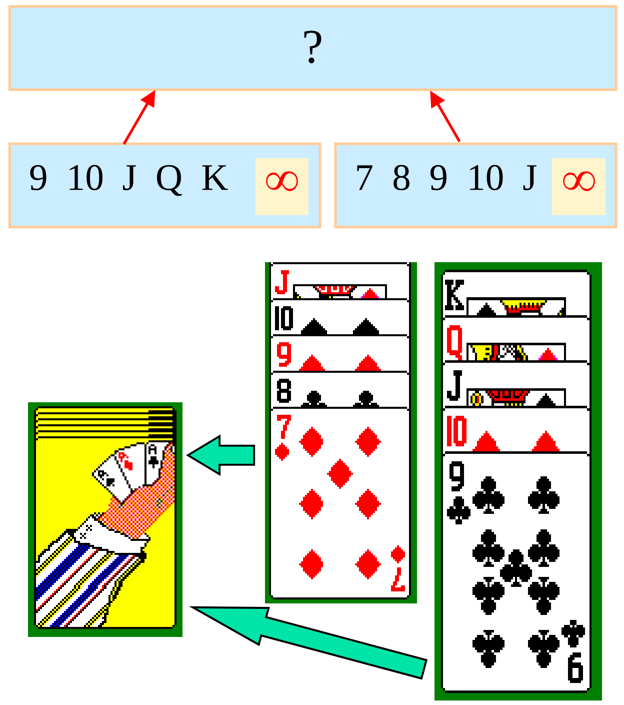
2.3.1 The divide-and-conquer approach

MERGE(A, p, q, r): $A[p\ ..\ q]$ and $A[q+1\ ..\ r]$ are in sorted order $\Rightarrow$ $A[p\ ..\ r]$



2.3.1 The divide-and-conquer approach

```
MERGE($A, p, q, r$)
    $n_1 \leftarrow q-p+1$
    $n_2 \leftarrow r-q$
    create arrays $L[1 .. n_1+1]$ and $R[1 .. n_2+1]$
    for $i \leftarrow 1$ to $n_1$
        $L[i] \leftarrow A[p+i-1]$
    for $j \leftarrow 1$ to $n_2$
        $R[j] \leftarrow A[q+j]$
    $L[n_1+1] \leftarrow \infty$
    $R[n_2+1] \leftarrow \infty$
    $i \leftarrow 1$
    $j \leftarrow 1$
    for $k \leftarrow p$ to $r$
        if $L[i] \leq R[j]$
            $A[k] \leftarrow L[i]$
            $i \leftarrow i+1$
        else $A[k] \leftarrow R[j]$
            $j \leftarrow j+1$
// $\infty$ : To avoid having to check whether either pile is empty in each
// basic step, a sentinel card is put on the bottom of each pile.
```

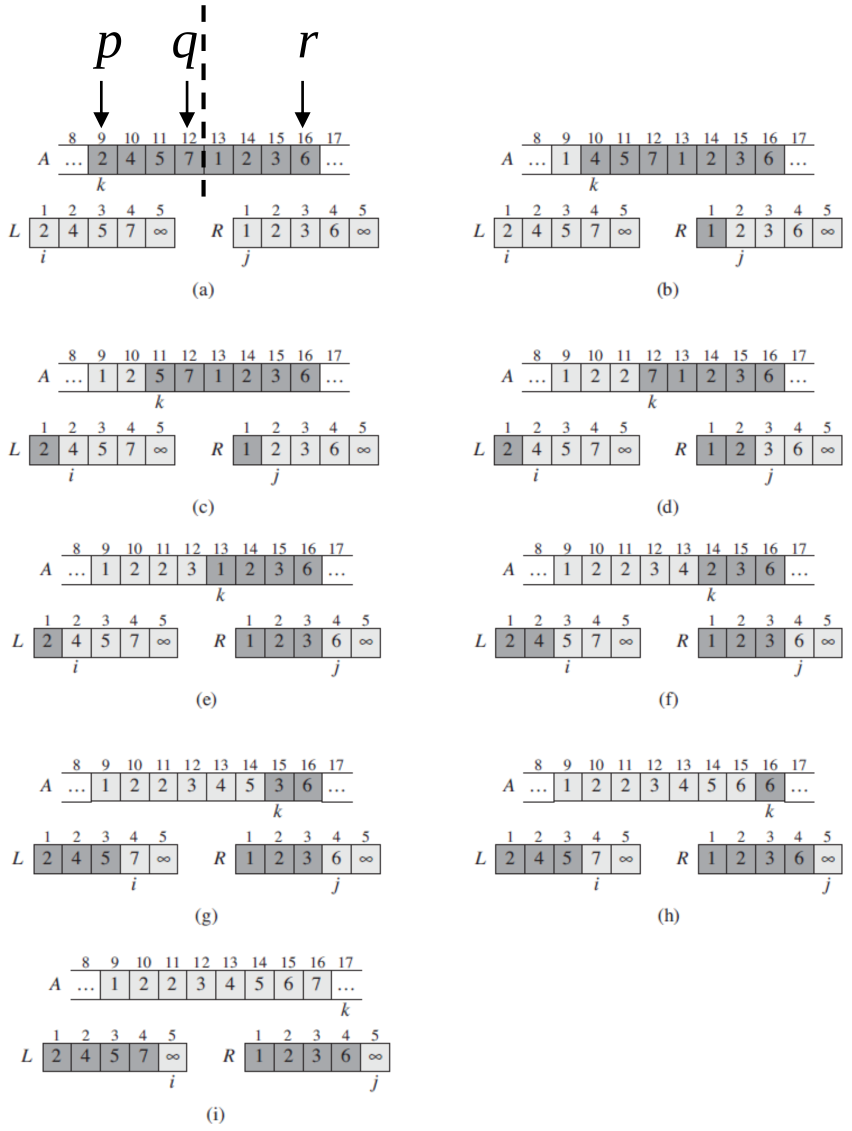


How does the subroutine work?

2.3.1 The divide-and-conquer approach

```
MERGE(A, p, q, r)
1     $n_1 \leftarrow q-p+1$
2     $n_2 \leftarrow r-q$
3     create arrays $L[1 .. n_1+1]$ and $R[1 .. n_2+1]$
4     for $i \leftarrow 1$ to $n_1$
5         $L[i] \leftarrow A[p+i-1]$
6     for $j \leftarrow 1$ to $n_2$
7         $R[j] \leftarrow A[q+j]$
8     $L[n_1+1] \leftarrow \infty$
9     $R[n_2+1] \leftarrow \infty$
10    $i \leftarrow 1$
11    $j \leftarrow 1$
12    for $k \leftarrow p$ to $r$
13        if $L[i] \leq R[j]$
14            $A[k] \leftarrow L[i]$
15            $i \leftarrow i+1$
16        else  $A[k] \leftarrow R[j]$
17            $j \leftarrow j+1$
```

Is the algorithm correct?



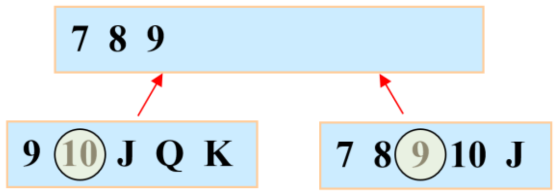
2.3.1 The divide-and-conquer approach -- correct

```
MERGE($A, p, q, r$)
1   $n_1 \leftarrow q-p+1$
2   $n_2 \leftarrow r-q$
3   create arrays $L[1 .. n_1+1]$ and $R[1 .. n_2+1]$
4   **for** $i \leftarrow 1$ **to** $n_1$
5       $L[i] \leftarrow A[p+i-1]$
6   **for** $j \leftarrow 1$ **to** $n_2$
7       $R[j] \leftarrow A[q+j]$
8   $L[n_1+1] \leftarrow \infty$
9   $R[n_2+1] \leftarrow \infty$
10  $i \leftarrow 1$
11  $j \leftarrow 1$
12  **for** $k \leftarrow p$ **to** $r$
13      **if** $L[i] \le R[j]$
14          $A[k] \leftarrow L[i]$
15          $i \leftarrow i+1$
16      **else** $A[k] \leftarrow R[j]$
17          $j \leftarrow j+1$
```

类似插入排序算法的证明，自己看。

properties of loop invariant for MERGE

- **Initialization**: prior to the first iteration of the loop, $k = p$, so $A[p .. k-1]$ is empty. $A[p .. k-1]$ contains 0 samllest elements of L and R, and since $i=j=1$, both $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.
- **Maintenance**: firstly suppose that $L[i] \le R[j]$. Because $A[p .. k-1]$ contains the $k-p$ smallest elements, after line 14 copies $L[i]$ into $A[k]$, the subarray $A[p .. k]$ will contain the $k-p+1$ smallest elements. If $L[i] > R[j]$, so do that.



- **Termination**: $k=r+1$, subarray $A[p .. k-1]$ is $A[p .. r]$, contains $r-p+1$ smallest elements of $L[1 .. n_1+1]$ and $R[1 .. n_2+1]$, in sorted order. All but the two largest (sentinels) have been copied back into A.

2.3.1 The divide-and-conquer approach -- Running time

```
MERGE($A$, $p$, $q$, $r$)                                      cost    times
1   $n_1 \leftarrow q-p+1$                                     $c$     $1$
2   $n_2 \leftarrow r-q$                                       $c$     $1$
3   create arrays $L[1 .. n_1+1]$ and $R[1 .. n_2+1]$          $c$     $1$
4   for $i \leftarrow 1$ to $n_1$                              $c$     $n_1+1$
5       $L[i] \leftarrow A[p+i-1]$                             $c$     $n_1$
6   for $j \leftarrow 1$ to $n_2$                              $c$     $n_2+1$
7       $R[j] \leftarrow A[q+j]$                               $c$     $n_2$
8   $L[n_1+1] \leftarrow \infty$                               $c$     $1$
9   $R[n_2+1] \leftarrow \infty$                               $c$     $1$
10  $i \leftarrow 1$                                           $c$     $1$
11  $j \leftarrow 1$                                           $c$     $1$
12  for $k \leftarrow p$ to $r$                                $c$     $r-p+2$
13      if $L[i] \le R[j]$                                     $c$     $r-p+1$
14          $A[k] \leftarrow L[i]$                             $c$     $x$
15          $i \leftarrow i+1$                                 $c$     $x$
16      else $A[k] \leftarrow R[j]$                           $c$     $r-p+1-x$
17          $j \leftarrow j+1$                                 $c$     $r-p+1-x$
```

$$r - p + 1$$
$$= n_1 + n_2 = n$$

$$1 \le x \le n_1$$

$$\Theta(n_1+n_2) = \Theta(n)$$

2.3.1 The divide-and-conquer approach -- Running time

```
MERGE(A, p, q, r)
1   $n_1 \leftarrow q-p+1$
2   $n_2 \leftarrow r-q$
3   create arrays $L[1 .. n_1+1]$ and $R[1 .. n_2+1]$
4   for $i \leftarrow 1$ to $n_1$
5       $L[i] \leftarrow A[p+i-1]$
6   for $j \leftarrow 1$ to $n_2$
7       $R[j] \leftarrow A[q+j]$
8   $L[n_1+1] \leftarrow \infty$
9   $R[n_2+1] \leftarrow \infty$
10  $i \leftarrow 1$
11  $j \leftarrow 1$
12  for $k \leftarrow p$ to $r$      // 设 $n = r-p$，算法执行 $n$ 次（每次循环，循环体语句执行常数次）
13      if $L[i] \leq R[j]$    // 这是基本操作，比较 $n$ 次
14          $A[k] \leftarrow L[i]$
15          $i \leftarrow i+1$
16      else $A[k] \leftarrow R[j]$
17          $j \leftarrow j+1$
```

以后分析复杂度时，只关心基本操作即可。

哪个（些）是基本操作?

$\Theta(n)$

2.3.1 The divide-and-conquer approach

- MERGE procedure as a subroutine.
- MERGE-SORT(A, p, r) sorts the elements in the subarray $A[p \mathinner{..} r]$.
- If $p \geq r$, the subarray has at most one element, already sorted, base case.
- Otherwise, the divide step computes an index q that partitions $A[p \mathinner{..} r]$ into two subarrays: $A[p \mathinner{..} q]$, containing $\lfloor n/2 \rfloor$ elements, and $A[q+1 \mathinner{..} r]$, containing $\lceil n/2 \rceil$ elements.

```
MERGE-SORT($A$, $p$, $r$)
if $p < r$
    $q \leftarrow \lfloor (p+r)/2 \rfloor$
    MERGE-SORT($A$, $p$, $q$)
    MERGE-SORT($A$, $q+1$, $r$)
    MERGE($A$, $p$, $q$, $r$)
```

```
MERGE($A$, $p$, $q$, $r$)
$n_1 \leftarrow q-p+1$
$n_2 \leftarrow r-q$
create arrays $L[1 .. n_1+1]$ and $R[1 .. n_2+1]$
for $i \leftarrow 1$ to $n_1$
    $L[i] \leftarrow A[p+i-1]$
for $j \leftarrow 1$ to $n_2$
    $R[j] \leftarrow A[q+j]$
$L[n_1+1] \leftarrow \infty$
$R[n_2+1] \leftarrow \infty$
$i \leftarrow 1$
$j \leftarrow 1$
for $k \leftarrow p$ to $r$
    if $L[i] \leq R[j]$
        $A[k] \leftarrow L[i]$
        $i \leftarrow i+1$
    else $A[k] \leftarrow R[j]$
        $j \leftarrow j+1$
```

```
MERGE-SORT($A, p, r$)
1  if $p < r$
2      $q \leftarrow \lfloor (p+r)/2 \rfloor$
3      MERGE-SORT($A, p, q$)
4      MERGE-SORT($A, q+1, r$)
5      MERGE($A, p, q, r$)
```

```
MERGE($A, p, q, r$)
1      $n_1 \leftarrow q-p+1$
2      $n_2 \leftarrow r-q$
3      create arrays $L[1 .. n_1+1]$ and $R[1 .. n_2+1]$
4      for $i \leftarrow 1$ to $n_1$
5          $L[i] \leftarrow A[p+i-1]$
6      for $j \leftarrow 1$ to $n_2$
7          $R[j] \leftarrow A[q+j]$
8      $L[n_1+1] \leftarrow \infty$
9      $R[n_2+1] \leftarrow \infty$
10     $i \leftarrow 1$
11     $j \leftarrow 1$
12     for $k \leftarrow p$ to $r$
13         if $L[i] \le R[j]$
14                 $A[k] \leftarrow L[i]$
15                 $i \leftarrow i+1$
16         else  $A[k] \leftarrow R[j]$
17                 $j \leftarrow j+1$
```

在使用的时候:

忽略递归过程!

函数重在接口!

递归重在调用!

c2-2-mergesort.c

```
MERGE-SORT($A$, 1, 4)
1   if 1 < 4
2       $q \leftarrow \lfloor (1+4)/2 \rfloor = 2$
3       MERGE-SORT($A$, 1, 2)
            1   if 1 < 2
            2       $q \leftarrow \lfloor (1+2)/2 \rfloor = 1$
            3       MERGE-SORT($A$, 1, 1)
                    1   if 1 < 1
            4       MERGE-SORT($A$, 2, 2)
                    1   if 2 < 2
            5       MERGE($A$, 1, 1, 2)
4       MERGE-SORT($A$, 3, 4)
            1   if 3 < 4
            2       $q \leftarrow \lfloor (3+4)/2 \rfloor = 3$
            3       MERGE-SORT($A$, 3, 3)
                    1   if 3 < 3
            4       MERGE-SORT($A$, 4, 4)
                    1   if 4 < 4
            5       MERGE($A$, 3, 3, 4)
5       MERGE($A$, 1, 2, 4)
```

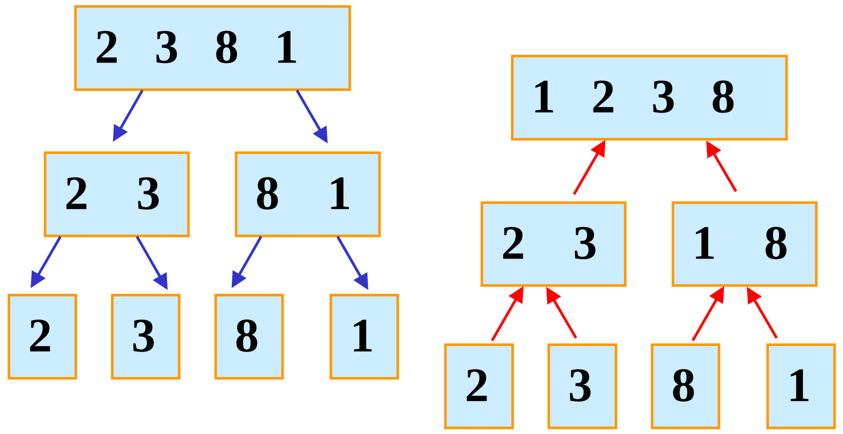
有了伪代码，程序就好写了！

```
MERGE-SORT($A$, $p$, $r$)
1  if $p < r$
2      $q \leftarrow \lfloor (p+r)/2 \rfloor$
3      MERGE-SORT($A$, $p$, $q$)
4      MERGE-SORT($A$, $q+1$, $r$)
5      MERGE($A$, $p$, $q$, $r$)
```

```
void merge_sort(int Lt, int Rt)
{
    int mid;
    if(Lt < Rt)
    {
        mid = (Lt+Rt)/2;
        merge_sort(Lt, mid);
        merge_sort(mid+1, Rt);
        merge(Lt, mid, Rt);
    }
}
```

2.3.1 The divide-and-conquer approach

```
MERGE-SORT($A, p, r$)
1    if $p < r$
2         $q \leftarrow \lfloor (p+r)/2 \rfloor$
3         MERGE-SORT($A, p, q$)
4         MERGE-SORT($A, q+1, r$)
5         MERGE($A, p, q, r$)
```



```
MERGE-SORT($A$, 1, 4)
1    if $1 < 4$
2         $q \leftarrow \lfloor (1+4)/2 \rfloor = 2$
3         MERGE-SORT($A$, 1, 2)
              1    if $1 < 2$
              2         $q \leftarrow \lfloor (1+2)/2 \rfloor = 1$
              3         MERGE-SORT($A$, 1, 1)
                            1    if $1 < 1$
              4         MERGE-SORT($A$, 2, 2)
                            1    if $2 < 2$
              5         MERGE($A$, 1, 1, 2)
4         MERGE-SORT($A$, 3, 4)
              1    if $3 < 4$
              2         $q \leftarrow \lfloor (3+4)/2 \rfloor = 3$
              3         MERGE-SORT($A$, 3, 3)
                            1    if $3 < 3$
              4         MERGE-SORT($A$, 4, 4)
                            1    if $4 < 4$
              5         MERGE($A$, 3, 3, 4)
5         MERGE($A$, 1, 2, 4)
```

2.3.2 Analyzing divide-and-conquer algorithms

When an algorithm contains a recursive call to itself, its running time can often be described by a **recurrence equation** or **recurrence**, which described the overall running time on a problem of size *n* in terms of the running time on smaller inputs. **（递归函数的计算时间可表示为递归方程）**

We get the recurrence

$$T(n)=\begin{cases}\Theta(1) & \text{if } n\le c\\ aT(n/b)+D(n)+C(n) & \text{otherwise}\end{cases}$$

- $D(n)$ is the time of dividing the problem into subproblems.
- $C(n)$ is the time of combining the solutions to the subproblems into the solution to the original problem.

```
MERGE-SORT(A, p, r)
1  if p < r
2      q ← ⌊(p+r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q+1, r)
5      MERGE(A, p, q, r)
```

2.3.2 Analyzing divide-and-conquer algorithms

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

Assume that the original problem size is a power of 2, the recurrence analysis is simplified. Each divide step then yields two subsequences of size exactly *n*/2 (the assumption does not affect the order of growth of the solution to the recurrence).

（假设输入n是2的幂次方，每次分解，子问题输入数减半，则问题分析可以被简化。这种简化不影响问题的一般性分析。）

```
MERGE-SORT($A, p, r$)
1   if $p < r$
2       $q \leftarrow \lfloor (p+r)/2 \rfloor$
3       MERGE-SORT($A, p, q$)
4       MERGE-SORT($A, q+1, r$)
5       MERGE($A, p, q, r$)
```

2.3.2 Analyzing divide-and-conquer algorithms

How to set up the recurrence for $T(n)$, the worst-case running time of merge sort on n numbers?

- If one element $n = 1$, takes constant time.
- When $n > 1$,
  - **Divide**: One step, just computes the mid of the subarray, Thus, $D(n) = \Theta(1)$.
  - **Conquer**: Recursively solve two subprolems, each of size $n/2$, which contributes $2T(n/2)$ to the running time.
  - **Combine**: MERGE procedure shows $C(n) = \Theta(n)$.

```
MERGE-SORT(A, p, r)                          T(n)
1  if p < r
2        q ← ⌊(p+r)/2⌋                       D(n) = 1
3        MERGE-SORT(A, p, q)                 T(n/2)
4        MERGE-SORT(A, q+1, r)               T(n/2)
5        MERGE(A, p, q, r)                   C(n) = n
```

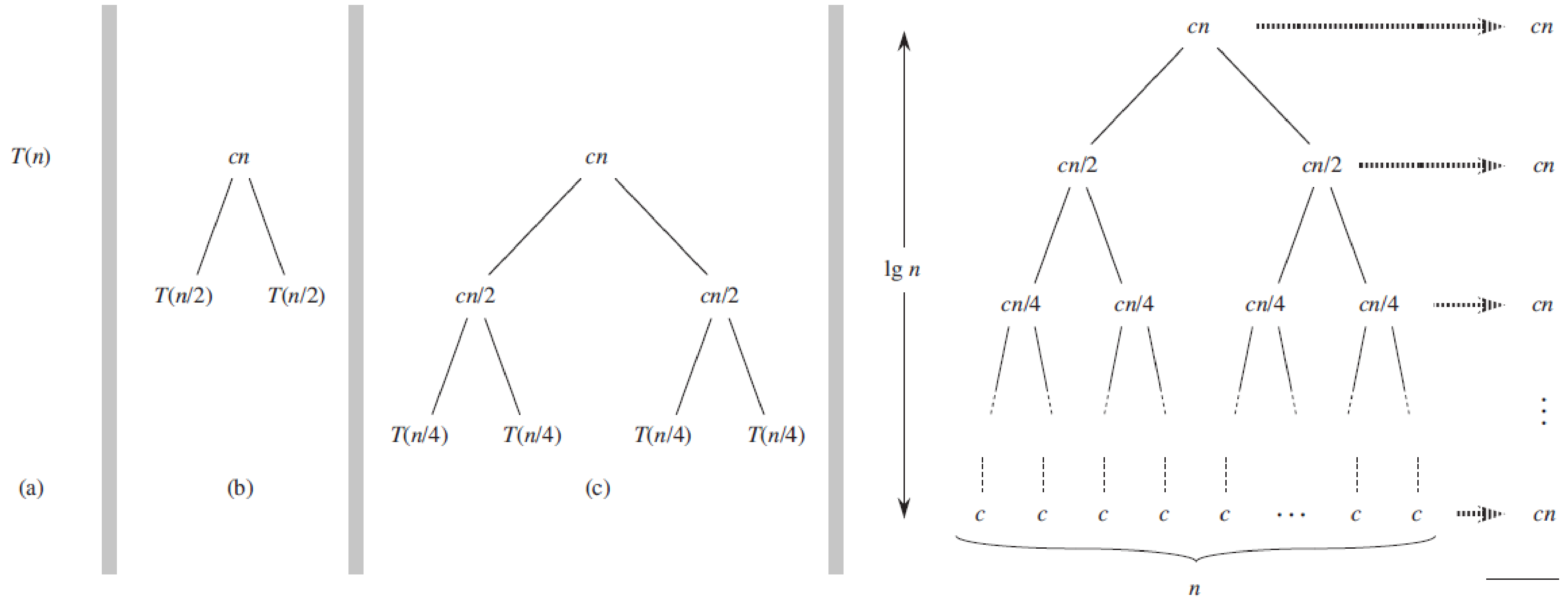
$$T(n)=\begin{cases}\Theta(1) & \text{if } n\le c\\ aT(n/b)+D(n)+C(n) & \text{otherwise}\end{cases} \Rightarrow T(n)=\begin{cases}\Theta(1) & \text{if } n=1\\ 2T(n/2)+\Theta(n) & \text{if } n>1\end{cases}$$

2.3.2 Analyzing divide-and-conquer algorithms

Rewrite the recurrence equation

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

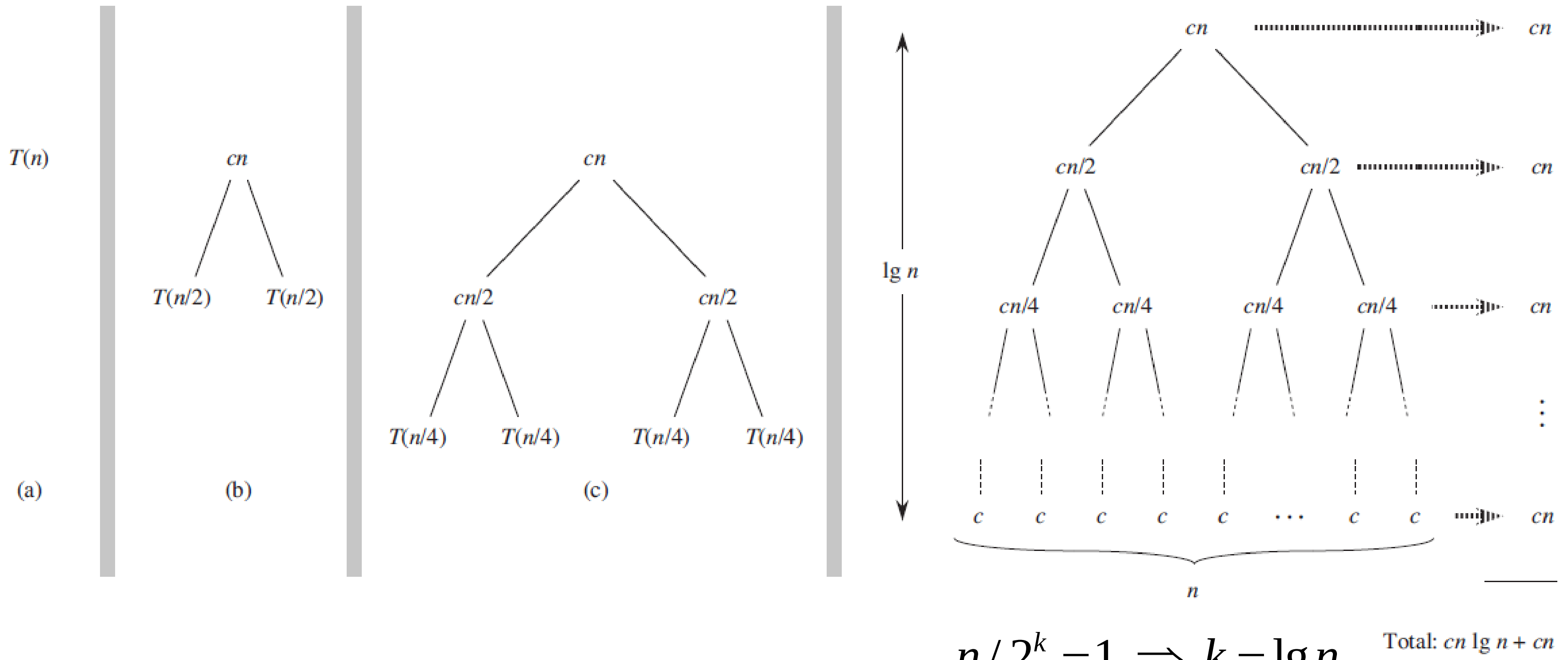
solve the recurrence equation.



$$n/2^k=1 \Rightarrow k=\lg n$$

Total: $cn\lg n+cn$

2.3.2 Analyzing divide-and-conquer algorithms



$$n/2^k = 1 \Rightarrow k = \lg n$$

Add up the costs of all the levels.

There are lg$n$+1 levels, each costing cn, for a total cost $cn(\lg n+1) = cn\lg n + cn = \Theta(n\lg n)$, obviously, it outperforms insertion sort, whose running time is $\Theta(n^2)$, in the worst case, for large enough inputs. （显然，归并排序优于插入排序，因为，$n\lg n < n^2$ ）

Exercises and problems

Exercises 分析求 Fibonacci序列 $F(n) = F(n\text{-}1) + F(n\text{-}2)$ 如下4种算法的时间复杂度。

```
F(n)
    if(n <= 2)
        return 1;
    else
        return F(n-1)+F(n-2);
```

```
F(n)
    f1 = 1;
    f2 = 1;
    for(i = 3; i <= n; i++)
    {
            f = f1 + f2;
            f1 = f2;
            f2 = f;
    }
```

[F(n), F(n-1)]

$$= [F(n\text{-}1), F(n\text{-}2)] * \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n\text{-}2}$$

$$F(n) = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \rightarrow \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2} * \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2}$$

Exercises and problems

Exercises

- 所有课后习题
- Horner’s rule for evaluating a polynomial.
- 补充：自己编程实现几种常用的排序算法（插入、冒泡、选择、归并、快排、等等），并比较分析其运算时间。输入分别为正序、随机序、逆序的n个数，n的规模达到10^5
- 写一篇关于排序的实现、对比、分析、应用相关的小论文
- 做一个快排动画的小软件

Problems

- 所有课后Problems