第八讲 排序算法

— 常用排序算法
— C++ 实现

潘建瑜@MATH.ECNU

排序算法

排序 (Sorting)：将一串数据依照指定方式进行排列。

常用排序方式：数值顺序，字典顺序。

算法评价重要指标

时间复杂度（最差、平均）

设有 n 个数据，一般来说，好的排序算法性能是 $O(n \log n)$，差的性能是 $O(n^2)$，而理想的性能是 $O(n)$。

空间复杂度

算法在运行过程中临时占用存储空间的大小。

稳定排序算法：相同的数据，排序后仍维持原有相对次序。

常见排序算法

算法	平均时间复杂度		算法	平均时间复杂度
选择排序	$O(n^2)$		归并排序	$O(n \log n)$
插入排序	$O(n^2)$		堆排序	$O(n \log n)$
希尔排序	$O(n \log^2 n)$		图书馆排序	$O(n \log n)$
冒泡排序	$O(n^2)$		基数排序	$O(n \cdot k)$
快速排序	$O(n \log n)$		桶排序	$O(n + k)$
			计数排序	$O(n + k)$
			鸽巢排序	$O(n + D)$
…… …… …… ……				

目录页

Contents

本讲假定是对数据进行从小到大排序

1 选择排序

—— 也称 最小排序

基本思想

- 先找出最小值，将其与第一个位置的元素进行交换
- 对剩余的数据重复以上过程，直至排序结束

选择排序：示例

原始序列：	2	8	3	12	5	20	7	14	5	16
第1轮排序：	2	8	3	12	5	20	7	14	5	16
第2轮排序：	2	3	8	12	5	20	7	14	5	16
第3轮排序：	2	3	5	12	8	20	7	14	5	16
第4轮排序：	2	3	5	5	8	20	7	14	12	16
第5轮排序：	2	3	5	5	7	20	8	14	12	16
第6轮排序：	2	3	5	5	7	8	20	14	12	16
第7轮排序：	2	3	5	5	7	8	12	14	20	16
第8轮排序：	2	3	5	5	7	8	12	14	20	16
第9轮排序：	2	3	5	5	7	8	12	14	16	20

MATLAB 演示：sort_min.m

选择排序：C++程序

```
// 找出最小值所在的位置
int findmin(int *px, int n)
{   int idx=0, xmin=*px;
    for (int i=1; i<n; i++)
        if (*(px+i)<xmin)
        {  xmin=*(px+i); idx=i;  }
    return idx;
}
```

```
// 选择排序(最小排序)
void sort_min(int *px, int n)
{   int idx, t;
    for(int k=0; k<n; k++)
    {   idx=findmin(px+k,n-k);
        t=px[k]; px[k]=px[k+idx]; px[k+idx]=t; % 交换
    }
}
```

选择排序：C++程序

Example：sort_selection.cpp
Example：sort_selection100000.cpp

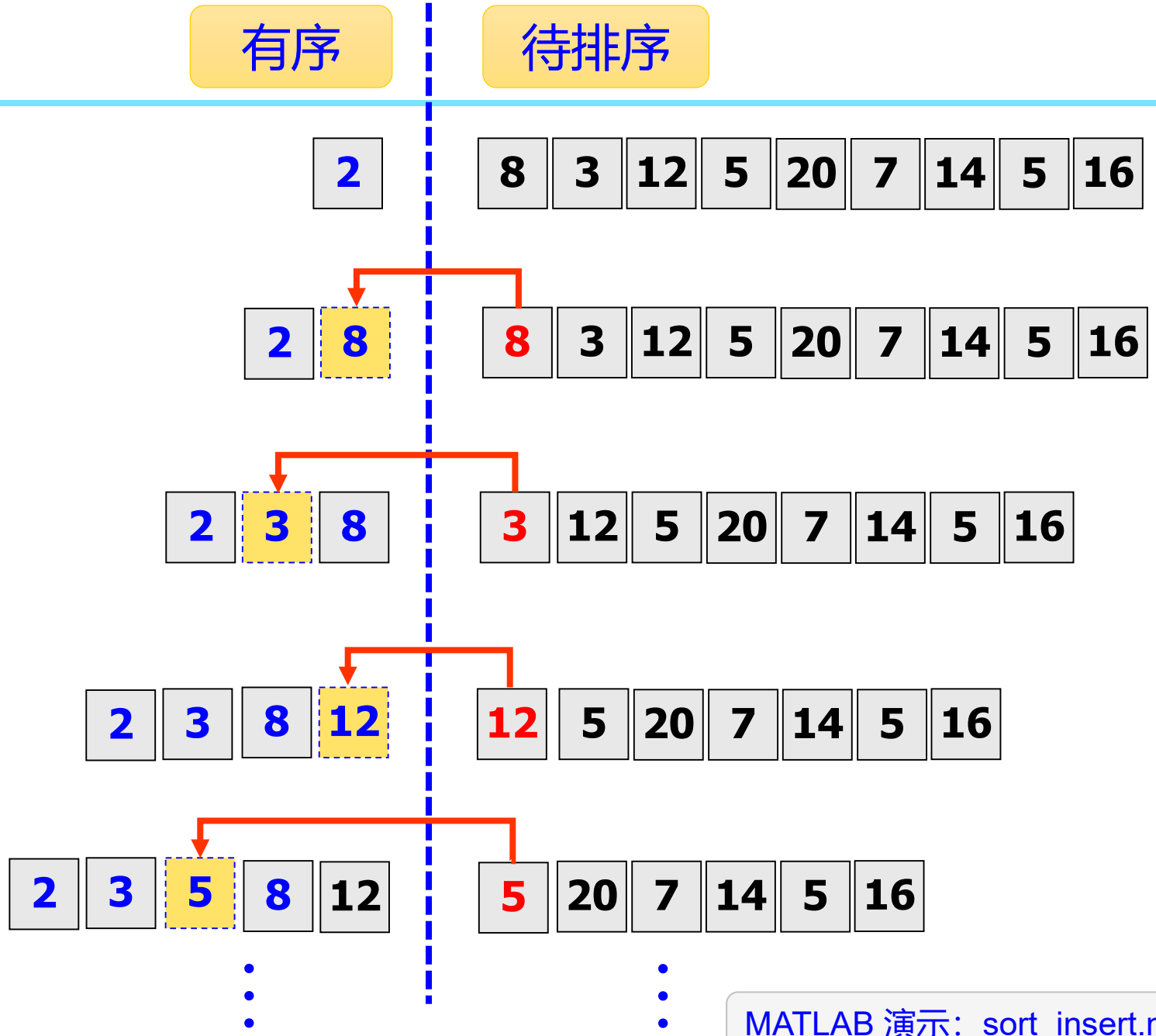
```
int main()
{
    int x[]={2, 8, 3, 12, 5, 20, 7, 14, 5, 16};
    int n, i;

    // 获取数据个数
    n = sizeof(x)/sizeof(x[0]);

    cout << "x=\n"; // 输出原始数据
    for(i=0;i<n;i++) cout << setw(3) << x[i];
    cout << endl;

    sort_min(x, n);  // 排序

    cout << "排序后：\n"; // 输出排序后结果
    for(i=0;i<n;i++) cout << setw(3) << x[i];
    return 0;
}
```

2 插入排序

基本思想

- 假设前面 k 个元素已经按顺序排好了，在排第 k+1 个元素时，将其插入到前面已排好的 k 个元素中，使得插入后得到的 k+1 个元素组成的序列仍按值有序。
- 然后采用同样的方法排第 k+2 个元素。
- 以此类推，直到排完序列的所有元素为止。

示例



MATLAB 演示：sort_insert.m

插入排序的实现

关键点：如何将第 $k+1$ 个元素插入到前面的有序序列中?

假定序列为 $x_1, x_2, \ldots, x_k, x_{k+1}, \cdots$

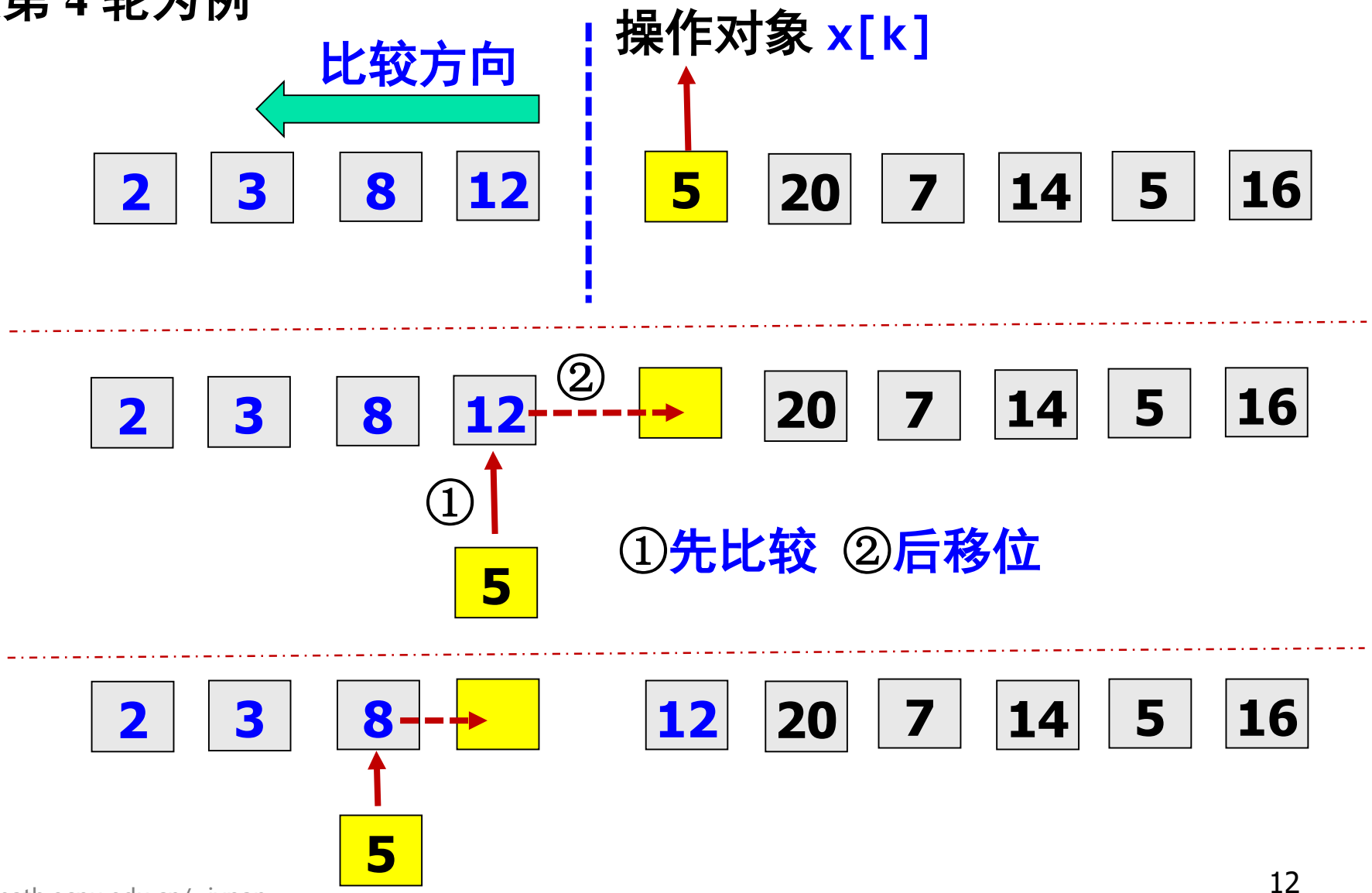
策略：将 x_{k+1} 依次与 $x_k, x_{k-1}, \ldots$ 进行比较，

直至遇见第一个不大于 x_{k+1} 的元素为止。

优化：可以将比较与移位同时进行。

插入排序的实现

以第 4 轮为例



插入排序：C++ 程序

```
// 插入排序（部分代码）
   ... ...
   for(k=1; k<n; k++)
   {
      key = x[k];
      for (i=k-1; x[i]>key && i>=0; i--)
      {
         x[i+1] = x[i];
      }
      x[i+1] = key;
   }
   ... ...
```

留作练习

```
void sort_insert(int * px, int n)
```

3 希尔排序

又称“缩小增量排序” Diminishing Increment Sort

由 D. Shell 于 1959 年提出，是对插入排序的改进

希尔排序基本过程

MATLAB 演示：sort_shell_comp.m

基本过程描述如下：

① 把序列按照某个增量（gap）分成几个子序列，对这几个子序列进行插入排序。

② 不断缩小增量，扩大每个子序列的元素数量，并对每个子序列进行插入排序。

③ 当增量为 1 时，子序列就是整个序列，而此时序列已经基本有序了，因此只需做少量的比较和移动就可以完成对整个序列的排序。

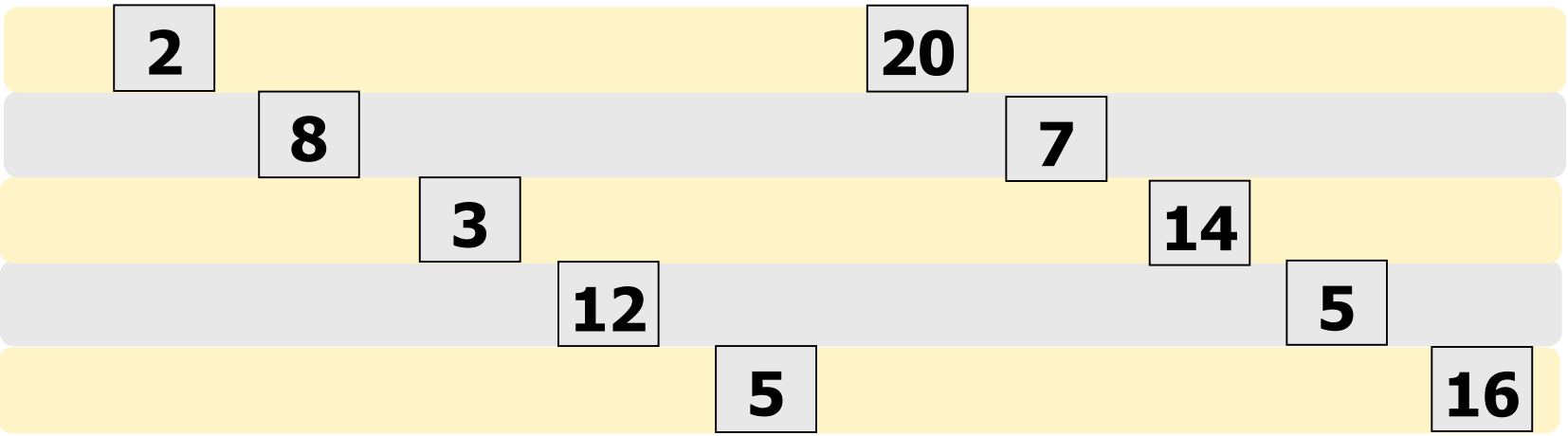
出发点：插入排序在元素基本有序的情况下，效率很高。

gap：初始值设为 n/2，然后不断减半。

希尔排序：示例

2	8	3	12	5	20	7	14	5	16

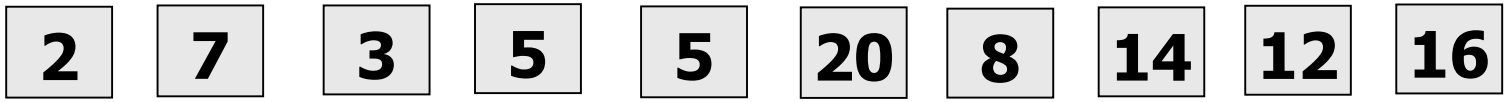
第一轮：gap=10/2=5



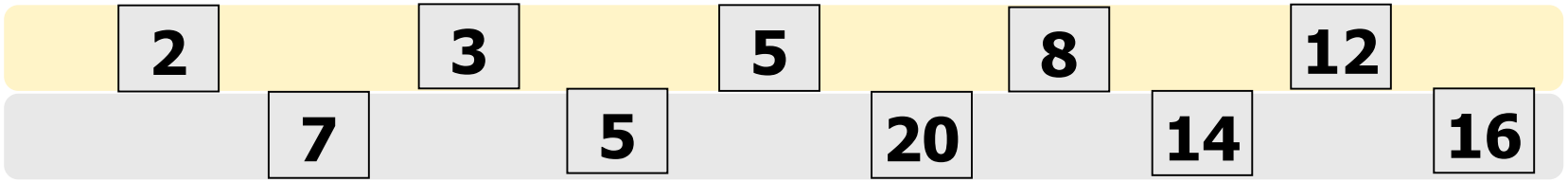
第一轮排序后

2	7	3	5	5	20	8	14	12	16

希尔排序：示例



第二轮：gap=gap/2=2



第二轮排序后

希尔排序：示例

2	5	3	7	5	14	8	16	12	20

第三轮：gap=gap/2=1

第三轮排序后

2	3	5	5	7	8	12	14	16	20

MATLAB 演示：sort_shell.m

留作练习

```
void sort_shell(int * px, int n)
```

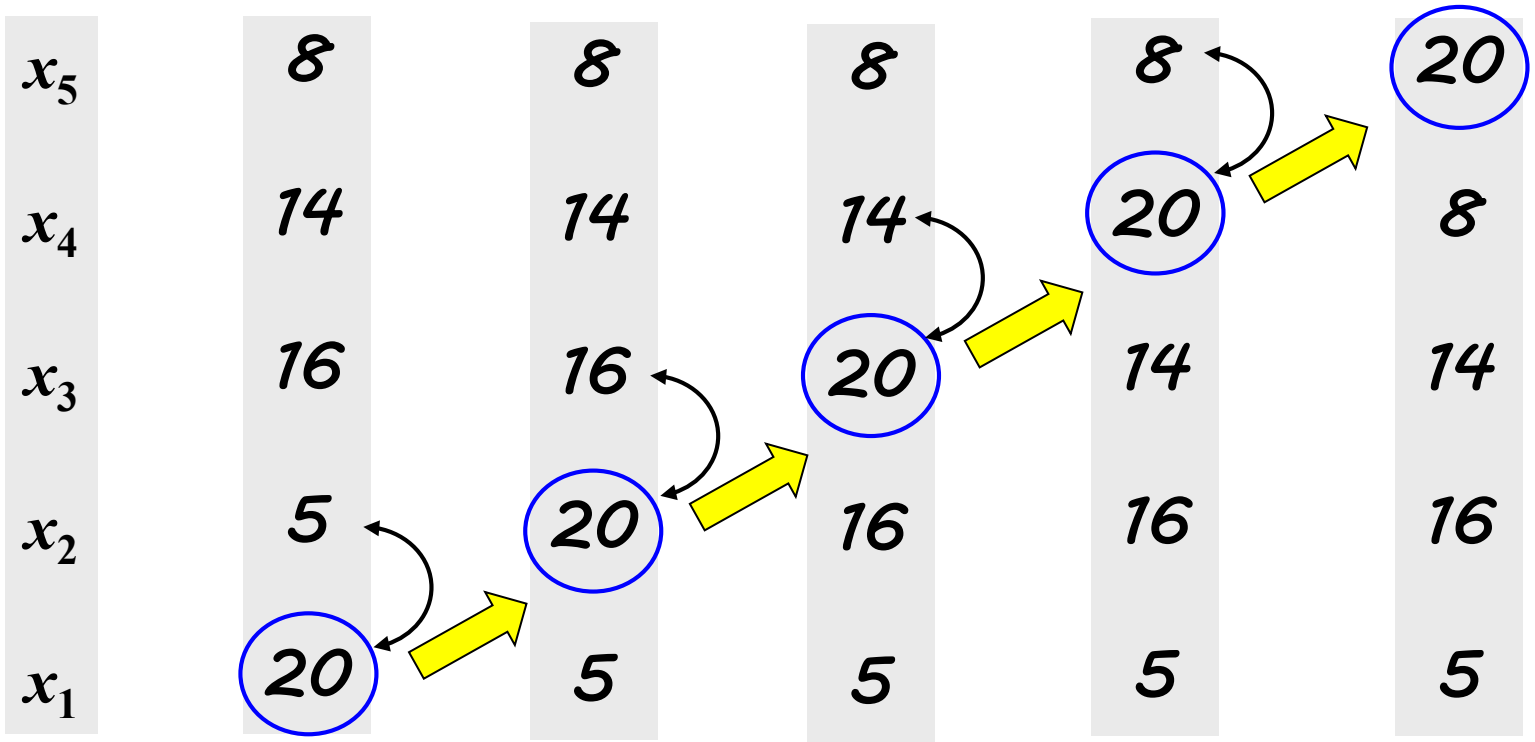
4 冒泡排序

基本过程

- 走访需要排序的序列，比较相邻的两个元素，如果他们的顺序错误就把他们交换过来。
- 不断重复上述过程，直到没有元素需要交换，排序结束

算法名字由来是因为越大的元素会经由交换慢慢“浮”到数列的顶端。

冒泡排序：示例



MATLAB 演示：sort_bubble.m

冒泡排序：过程描述

冒泡排序 执行过程

- 将第 1 个和第 2 个元素进行比较，如果前者大于后者，则交换两者的位置，否则位置不变；然后将第 2 个元素与第 3 个元素进行比较，如果前者大于后者，则交换两者的位置，否则位置不变；依此类推，直到最后两个元素比较完毕为止。这就是第一轮冒泡过程，这个过程结束后，最大的元素就“浮”到了最后一个位置上。
- 对前面 n-1 个元素进行第二轮冒泡排序，结束后，这 n-1 个元素中的最大值就被安放在了第 n-1个位置上。
- 对前面的 n-2个元素进行第三轮冒泡排序。
- 以此类推，当执行完第 n-1 轮冒泡过程后，排序结束。

冒泡排序的优化

简单优化

如果在某轮冒泡过程中没有发生元素交换，这说明整个序列已经排好序了，这时就不用再进行后面的冒泡过程，可以直接结束程序

进一步优化

假设有 100 个数组成的数组，仅前面10个无序，后面90个都已排好序且都大于前面10个数字，那么在第一轮冒泡过程后，最后发生交换的位置必定小于10，且这个位置之后的数据必定已经有序了，记录下这位置，第二轮遍历是只要到这个位置就可以了。

记录每轮遍历最后发生交换的位置，下次遍历只需到此位置为止

5 快速排序

—— 目前最常用的排序算法之一

快速排序采用的是**分而治之**思想：将原问题分解为若干个规模更小但结构与原问题相似的子问题，然后递归求解这些子问题，最后将这些子问题的解组合为原问题的解

快速排序

具体过程可以描述为：

随机选定其中一个元素作为基准数（pivot）（通常采用第一个元素），然后通过循环和比较运算，将原序列分割成两部分，使得新序列中在该基准数前面的元素都小于等于这个元素，而其后面的元素都大于等于这个元素。（这时基准数已经归位）

依此类推，再对这两个分割好的子序列进行上述过程，直到排序结束。（递归思想，分而治之）

快速排序：执行过程

原始序列：

6	1	3	7	5	9	2	4	8	10

第一步的具体实现方法：（假定基准数的原始位置是 $i_1=1$）

1) 先从原序列的最右边开始，往左找出第一个小于 6 的数，然后将该数与基准数交换位置，设基准数新位置为 i_2

2) 从 i_1 右边的位置开始，往右找出第一个大于 6 的数，然后将该数与基准数交换位置，设基准数新位置为 i_3

3) 从 i_2 左边的位置开始，往左找出第一个小于 6 的数，然后将该数与基准数交换位置，设基准数新位置为 i_4

4) 从 i_3 右边的位置开始，往右找出第一个大于 6 的数，然后将该数与基准数交换位置，设基准数新位置为 i_5

5) 不断重复以上过程，遍历整个序列

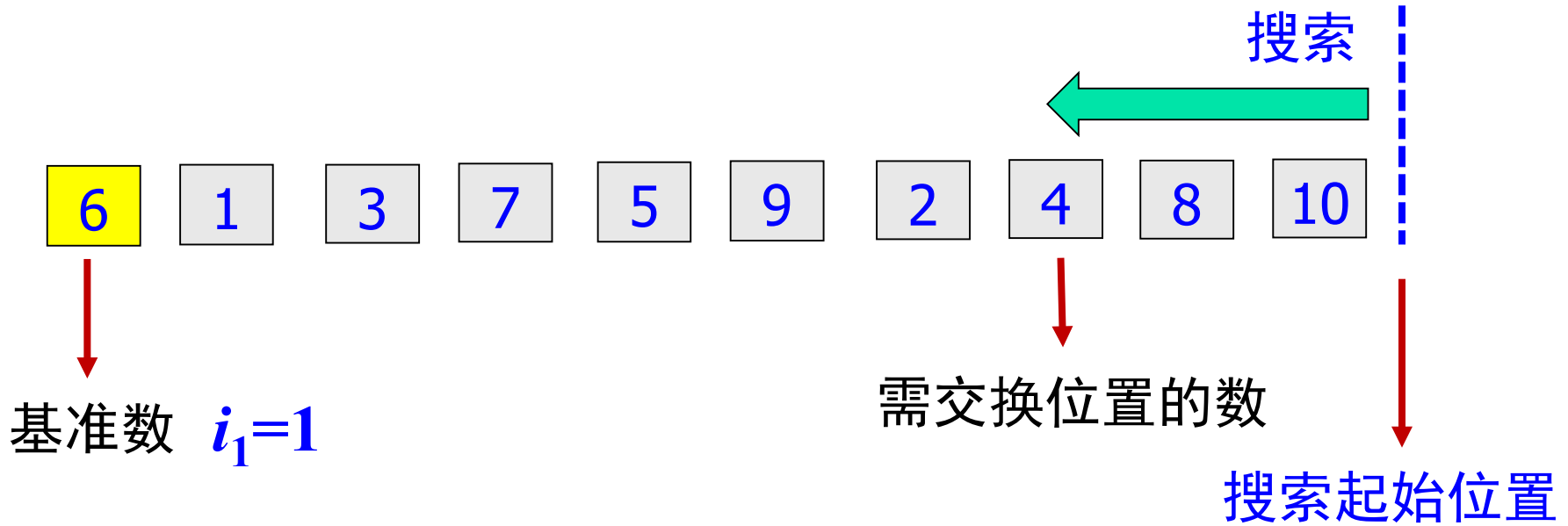
快速排序：示例

原始序列：

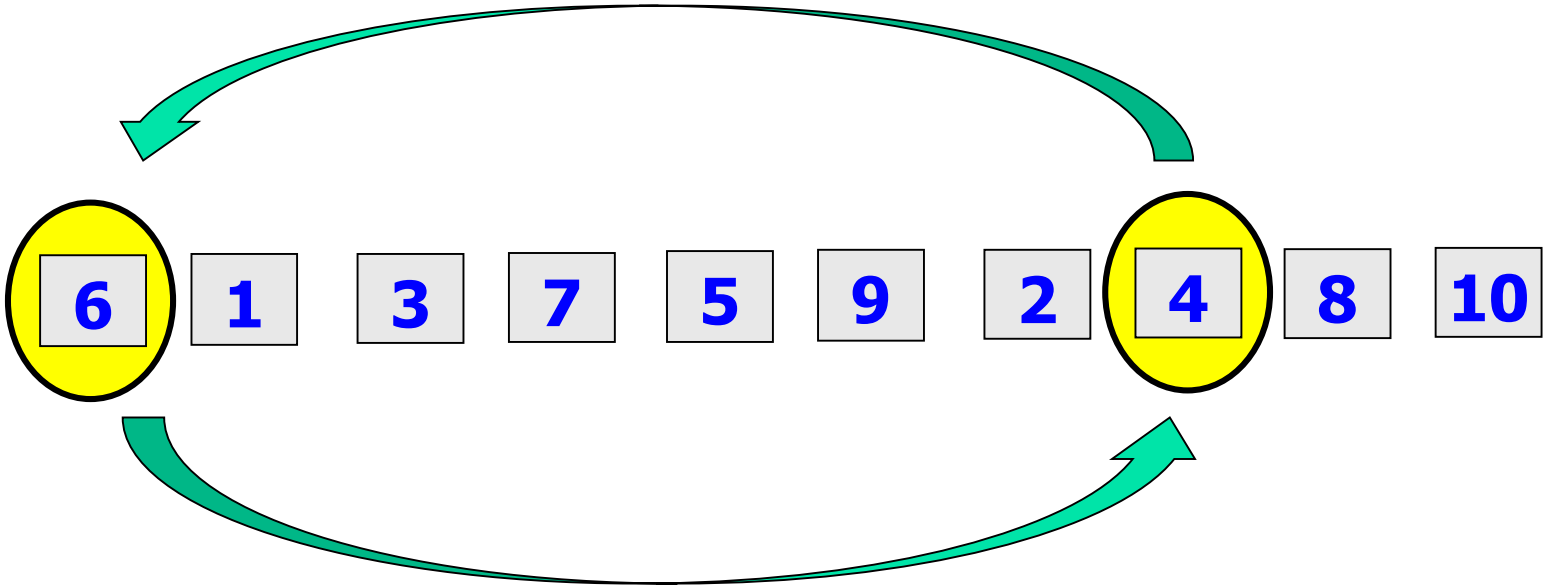
6	1	3	7	5	9	2	4	8	10

第一步：

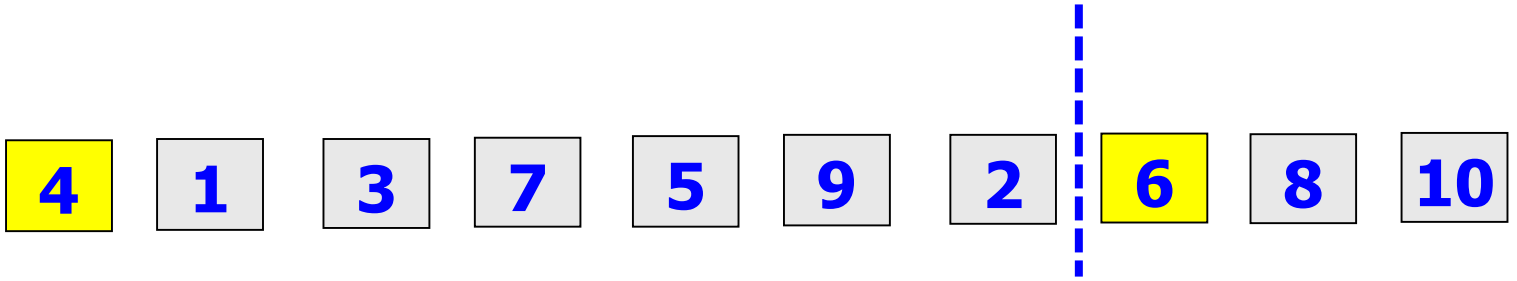
我们选第一个元素为基准数，即将 6 作为基准数。我们的目标是得到一个新序列，使得在这个新序列中，排在 6 前面的数字都小于 6，而排在 6 后面的数字都不小于 6。



举例：第一步

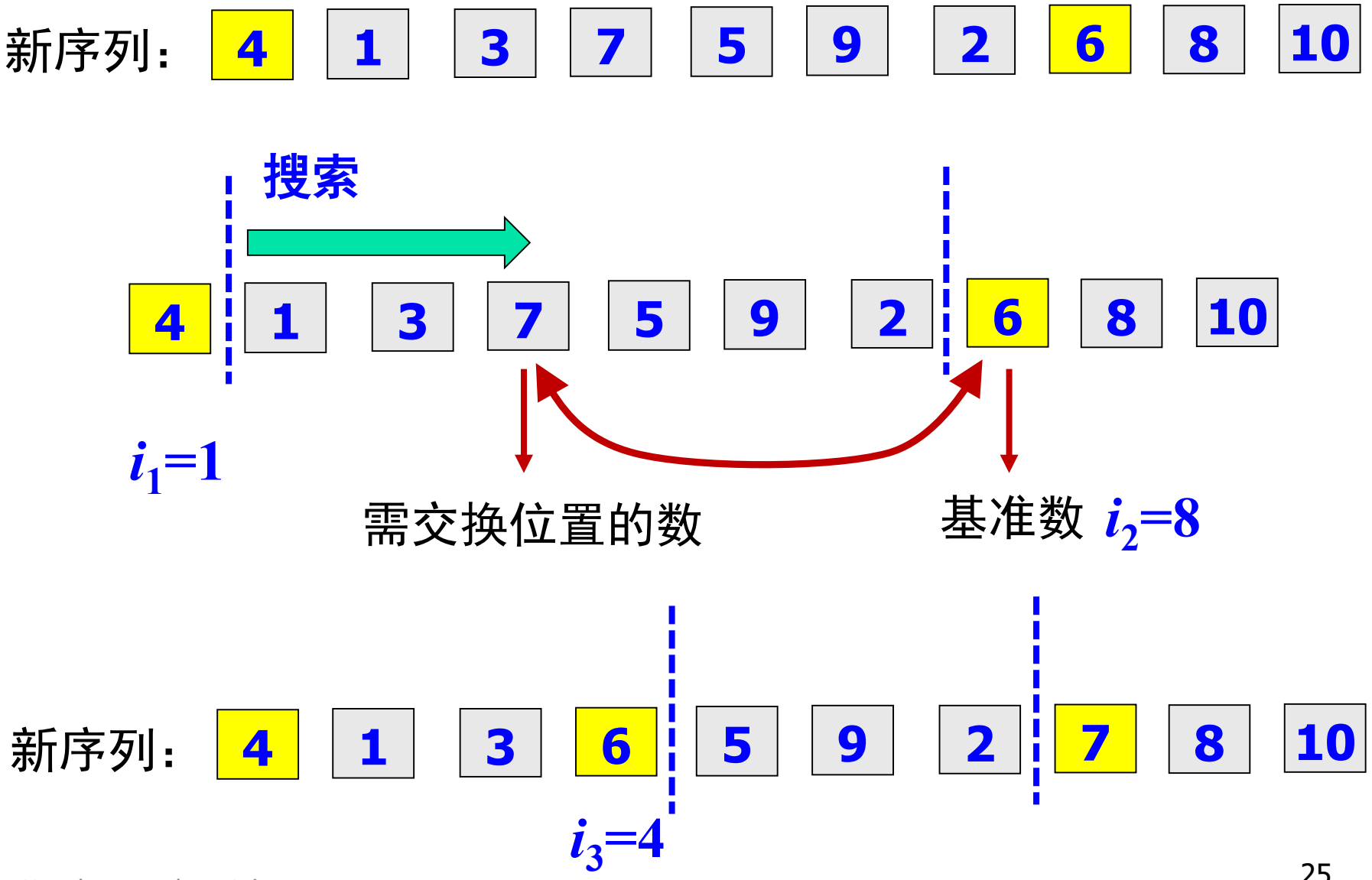


新序列：

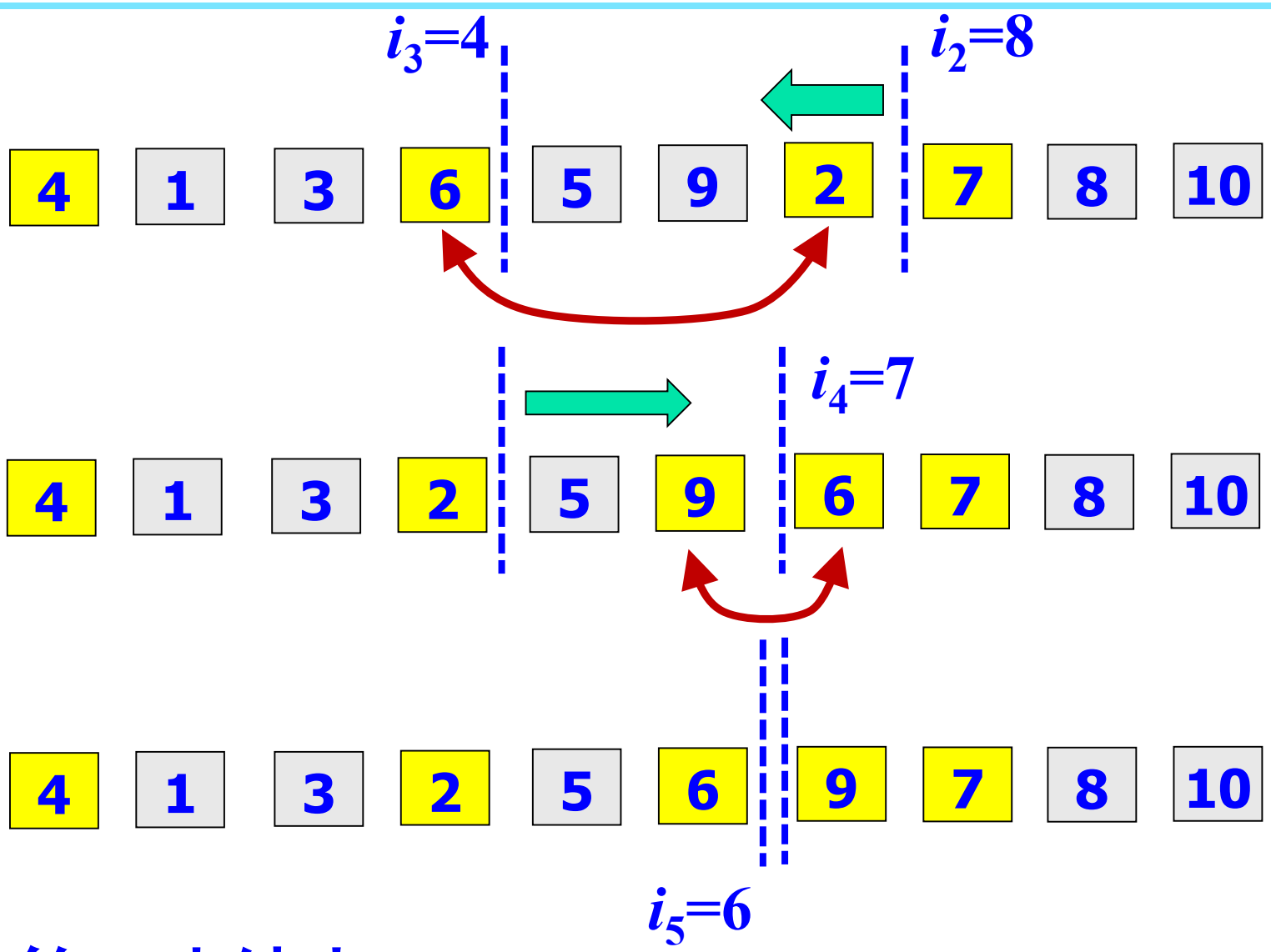


$i_1=1$ $i_2=8$

举例：第一步


新序列：
4 1 3 7 5 9 2 6 8 10
搜索
4 1 3 7 5 9 2 6 8 10
i_1=1
需交换位置的数
基准数 i_2=8
新序列：
4 1 3 6 5 9 2 7 8 10
i_3=4

举例：第一步

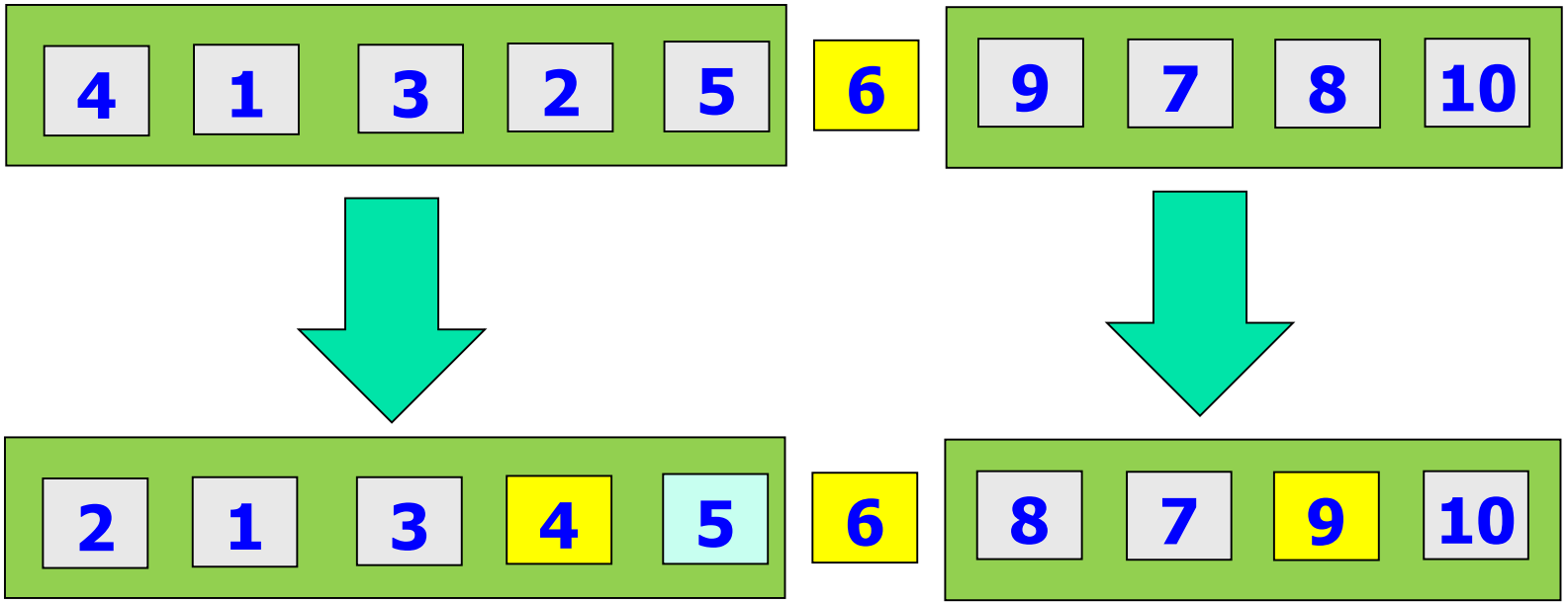


第一步结束！

举例：第二步

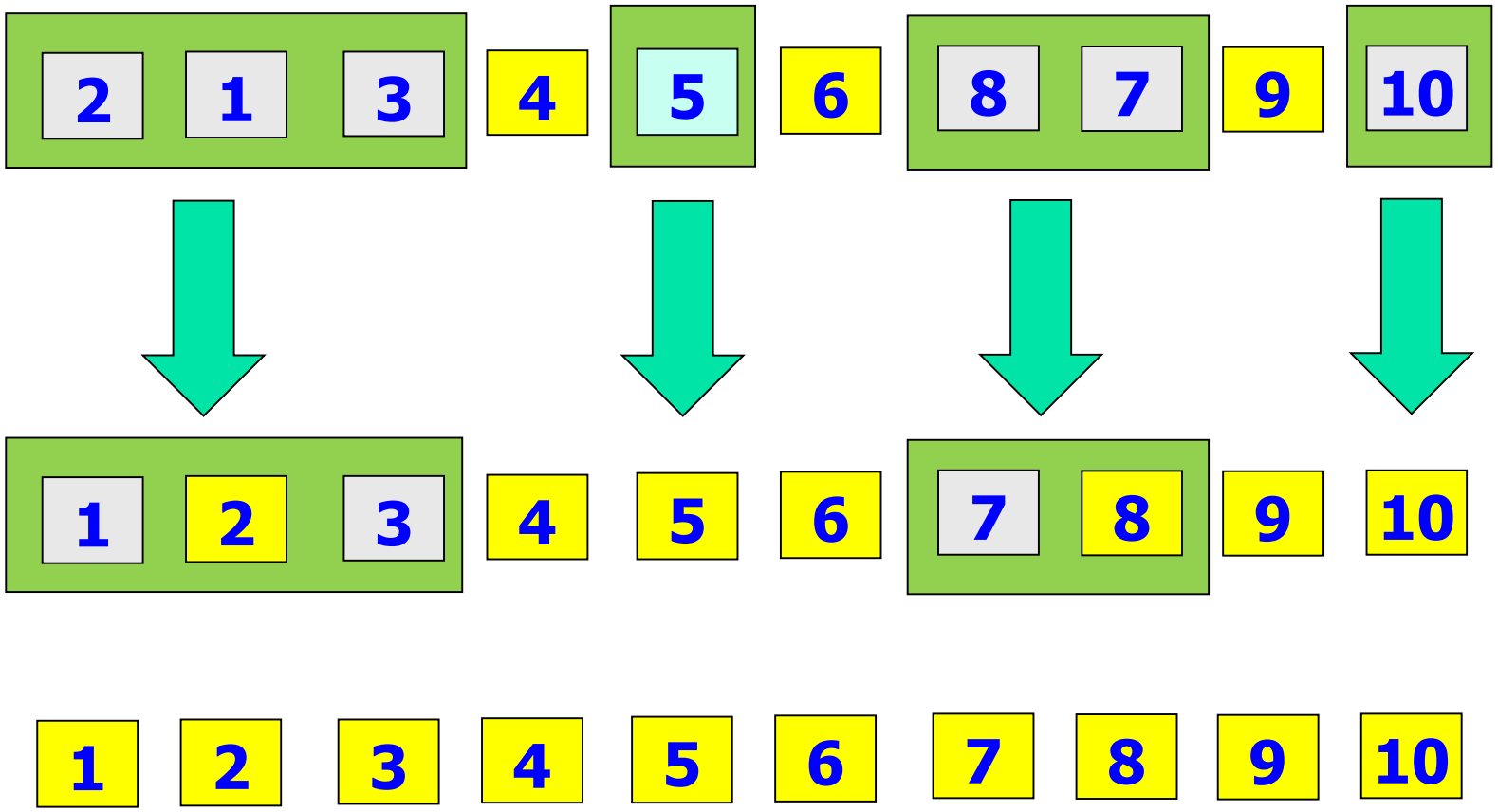
第二步：

对基准数所在位置前面的子序列和后面的子序列，分别重复第一步的过程。



不断重复：递归

MATLAB 演示：sort_quick.m



排序结束！

快速排序：C++ 程序

事实上，在快速排序每一步执行过程中，可以不用交换，而是直接覆盖即可。思考：如何实现？

快速排序还有很多改进版本，如随机选择基准数，区间内数据较少时直接用其它的方法排序以减小递归深度等等。有兴趣的同学可以深入研究。

留作练习

```
void sort_quick(int* px, int left, int right)
```

第八讲上机作业

1、编写函数，实现插入排序。在主函数中生成 15 个不超过 100 的随机正整数，存放在数组 *x* 中，然后调用排序函数对 *x* 进行排序，并输出排序前后的 *x*。（程序取名 **hw08_01.cpp**）

```
void sort_insert(int * px, int n);
```

2、将上题中的排序方法改为 **Shell** 排序。（程序取名 **hw08_02.cpp**）

```
void sort_shell(int * px, int n);
```

3、将上题中的排序方法改为优化后的冒泡排序。（程序取名 **hw08_03.cpp**）

```
void sort_bubble(int * px, int n);
```

4、将上题中的排序方法改为快速排序。（程序取名 **hw08_04.cpp**）

```
void sort_quick(int * px, int left, int right);
```

第八讲上机作业

5、折半插入排序

插入排序主要过程是依次将新元素插入到前面已排好序的序列中。在寻找插入点时，我们采用的是按顺序依次进行比较。为了减少比较次数，我们可以采用折半查找的方法：设待插入元素为 a[k]，待插入区域为 [low, high]，记 m=(low+high)/2，如果 a[k]<a[m]，则新的待插入区间为 [low, m-1]（即令 high=m-1），否则为 [m+1, high]（即令 low=m+1）。依此类推，直到 low<=high 不成立，此时 high+1 就是插入点。

编写函数，实现折半插入排序，并在主函数中以 ***x*** 为例，调用排序函数对 ***x*** 进行排序，并输出排序前后的 ***x***。（程序取名 **hw08_05.cpp**）

```
int find_location(int * px, int left, int right, int key);
void sort_insert_binary (int* px, int n);
```