Lecture2
Sorting

陆清怡

2022.10.21

991《数据结构与算法》考纲

8、排序

（1）排序的基本概念。

（2）插入排序、冒泡排序、快速排序、堆排序、归并排序、基数排序算法的原理、复杂度。

Sorting

- 排序的基本概念
- 插入排序
- 冒泡排序
- 归并排序
- 基数排序
- 快速排序
- 堆排序
- 真题回顾

排序的基本概念

- 算法的稳定性
  - 3，2，1，2　　1，2，2，3 稳定　　1，2，2，3不稳定

- 内部排序 vs 外部排序
  - 内部排序：排序期间元素全部存放在内存中，常用操作为比较和移动
  - 外部排序：元素无法全部同时存放在内存中

排序的基本概念

- Inversion

Given any list of n numbers, there are

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

pairs of numbers

For example, the list (1, 3, 5, 4, 2, 6) contains the following 15 pairs:

(1, 3)	(1, 5)	(1, 4)	(1, 2)	(1, 6)
	(3, 5)	(3, 4)	(3, 2)	(3, 6)
		(5, 4)	(5, 2)	(5, 6)
			(4, 2)	(4, 6)
				(2, 6)

For a random ordering, we would expect approximately half of all pairs are inversions:

$$\frac{1}{2}\binom{n}{2} = \frac{n(n-1)}{4} = \mathbf{O}(n^2)$$

Insertion Sort

Case	Run Time	Comments
Worst	$\Theta(n^2)$	Reverse sorted
Average	$O(d+n)$	Slow if $d = \omega(n)$
Best	$\Theta(n)$	Very few inversions: $d = O(n)$

For any unsorted list:

- Treat the first element as a sorted list of size 1

Then, given a sorted list of size $k-1$

- Insert the k^{th} item into the sorted list
- The sorted list is now of size k

Code for this would be:

```
for ( int j = k; j > 0; --j ) {
    if ( array[j - 1] > array[j] ) {
        std::swap( array[j - 1], array[j] );
    } else {
        // As soon as we don't need to swap, the (k + 1)st
        // is in the correct location
        break;
```

Insertion Sort

- 折半插入算法 O(n^2)
- 希尔排序

Insertion Sort

(1) (8 Points) Here is a sorting algorithm in the following.

```
Procedure Sort(A):
  for j = 2 to A.length:
      key = A[j]
      i = j - 1
      while i > 0 and A[i] > key:
          A[i+1] = A[i]
          i = i - 1
      A[i+1] = key
      // Mark
```

- (3 Points) Which sorting algorithm does it describe?
- (5 Points) Given a list as `[31, 4, 59, 26, 41, 58]`, we use the above procedure to sort it. Write down what will the list be like each time when the procedure meets the `Mark`.

Insertion Sort

- Insertion Sort.
- `[4, 31, 59, 26, 41, 58]`

  `[4, 31, 59, 26, 41, 58]`

  `[4, 26, 31, 59, 41, 58]`

  `[4, 26, 31, 41, 59, 58]`

  `[4, 26, 31, 41, 58, 59]`

  or if misunderstanding "`2 to A.length`",

  `[31, 4, 59, 26, 41, 58]`

  `[31, 4, 26, 59, 41, 58]`

  `[31, 4, 26, 41, 59, 58]`

  `[31, 4, 26, 41, 58, 59]`

Bubble Sort

Starting with the first item, assume that it is the largest

Compare it with the second item:

- If the first is larger, swap the two,
- Otherwise, assume that the second item is the largest

Continue up the array, either swapping or redefining the largest item

After one pass, the largest item must be the last in the list

Start at the front again:

- the second pass will bring the second largest element into the second last position

Repeat $n - 1$ times, after which, all entries will be in place

Bubble Sort

Case	Run Time	Comments
Worst	$\Theta(n^2)$	$\Theta(n^2)$ inversions
Average	$\Theta(n + d)$	Slow if $d = \omega(n)$
Best	$\Theta(n)$	$d = O(n)$ inversions

```
template <typename Type>
void bubble( Type *const array, int const n ) {
        for ( int i = n - 1; i > 0; --i ) {
                for ( int j = 0; j < i; ++j ) {
                        if ( array[j] > array[j + 1] ) {
                                std::swap( array[j], array[j + 1] );
                        }
                }
        }
}
```

Bubble Sort

The next few slides show some implementations of bubble sort together with a few improvements:

- reduce the number of swaps,
- halting if the list is sorted,
- limiting the range on which we must bubble
- alternating between bubbling up and sinking down

Bubble Sort

(1) (8 Points) Here is a sorting algorithm in the following.

```
Procedure Sort(A):
  for i = 1 to A.length - 1:
      for j = A.length downto i + 1:
          if A[j] < A[j - 1]
              key = A[j]
              A[j] = A[j - 1]
              A[j - 1] = key
      // Mark
```

- (3 Points) Which sorting algorithm does it describe?
- (5 Points) Given a list as `[31, 4, 59, 26, 41, 58]`, we use the above procedure to sort it. Write down what will the list be like each time when the procedure meets the `Mark`.

Bubble Sort

- Bubble Sort.
- `[4, 31, 26, 59, 41, 58]`

  `[4, 26, 31, 41, 59, 58]`

  `[4, 26, 31, 41, 58, 59]`

  `[4, 26, 31, 41, 58, 59]`

  `[4, 26, 31, 41, 58, 59]`

Merge Sort

The merge sort algorithm is defined recursively:

- If the list is of size 1, it is sorted—we are done;
- Otherwise:
  - Divide an unsorted list into two sub-lists,
  - Sort each sub-list recursively using merge sort, and
  - Merge the two sorted sub-lists into a single sorted list

This strategy is called *divide-and-conquer*

Merge Sort

We can then run the following loop:

```
#include <cassert>
//...
int i1 = 0, i2 = 0, k = 0;

while ( i1 < n1 && i2 < n2 ) {
    if ( array1[i1] < array2[i2] ) {
        arrayout[k] = array1[i1];
        ++i1;
    } else {
        assert( array1[i1] >= array2[i2] );
        arrayout[k] = array2[i2];
        ++i2;
    }
    ++k;
}
```

We're not finished yet, we have to empty out the remaining array

```
for ( ; i1 < n1; ++i1, ++k ) {
    arrayout[k] = array1[i1];
}

for ( ; i2 < n2; ++i2, ++k ) {
    arrayout[k] = array2[i2];
}
```

Case	Run Time	Comments
Worst	$\Theta(n \ln(n))$	No worst case
Average	$\Theta(n \ln(n))$	
Best	$\Theta(n \ln(n))$	No best case

Merge Sort

The time required to sort an array of size $n > 1$ is:

- the time required to sort the first half,
- the time required to sort the second half, and
- the time required to merge the two lists

That is: $$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & n > 1 \end{cases}$$

Solution: $T(n) = \Theta(n \ln(n))$

Merge Sort

Problem 4(4pts): Prove that: The time complexity for mergesort is $O(nlog(n))$.

Solution:

(1) $$T(n) = 2T(n/2) + O(n)$$

(2) $$= 2(2T(n/4) + O(n/2)) + O(n)$$

(3) $$= 4T(n/4) + 2O(n/2) + O(n)$$

(4) $$= 4T(n/4) + 2O(n)$$

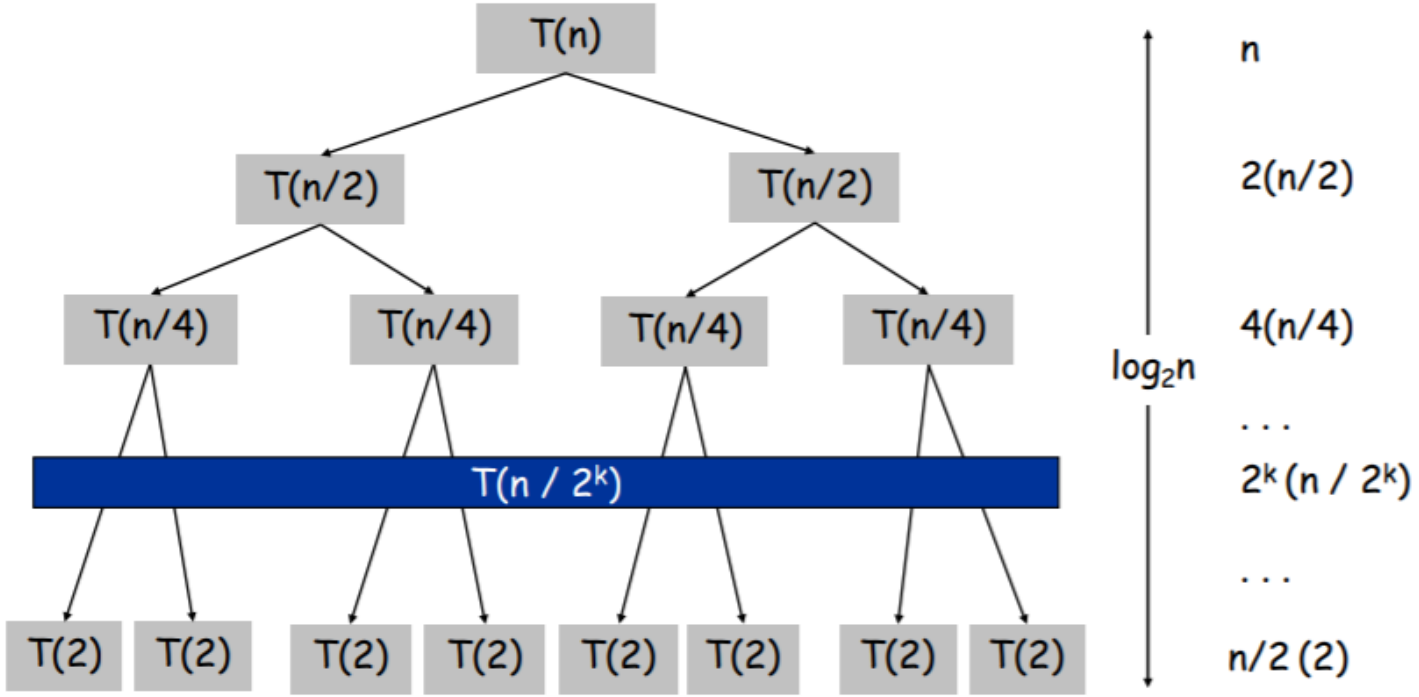
(5) $$= ...$$

(6) $$= O(nlog(n))$$

Merge Sort

Proof by Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



基数排序

基数排序是一种很特别的排序方法，它不基于比较和移动进行排序，而基于关键字各位的大小进行排序。基数排序是一种借助多关键字排序的思想对单逻辑关键字进行排序的方法。

假设长度为 n 的线性表中每个结点 a_j 的关键字由 d 元组（$k_j^{d-1}, k_j^{d-2}, \cdots, k_j^1, k_j^0$）组成，满足 $0 \le k_j^i \le r-1$（$0 \le j < n, 0 \le i \le d-1$）。其中 k_j^{d-1} 为最主位关键字，k_j^0 为最次位关键字。

为实现多关键字排序，通常有两种方法：第一种是**最高位优先**（MSD）法，按关键字位权重递减依次逐层划分成若干更小的子序列，最后将所有子序列依次连接成一个有序序列。第二种是**最低位优先**（LSD）法，按关键字权重递增依次进行排序，最后形成一个有序序列。

基数排序

下面描述以 r 为基数的最低位优先基数排序的过程，在排序过程中，使用 r 个队列 $Q_0, Q_1, \cdots, Q_{r-1}$。基数排序的过程如下：

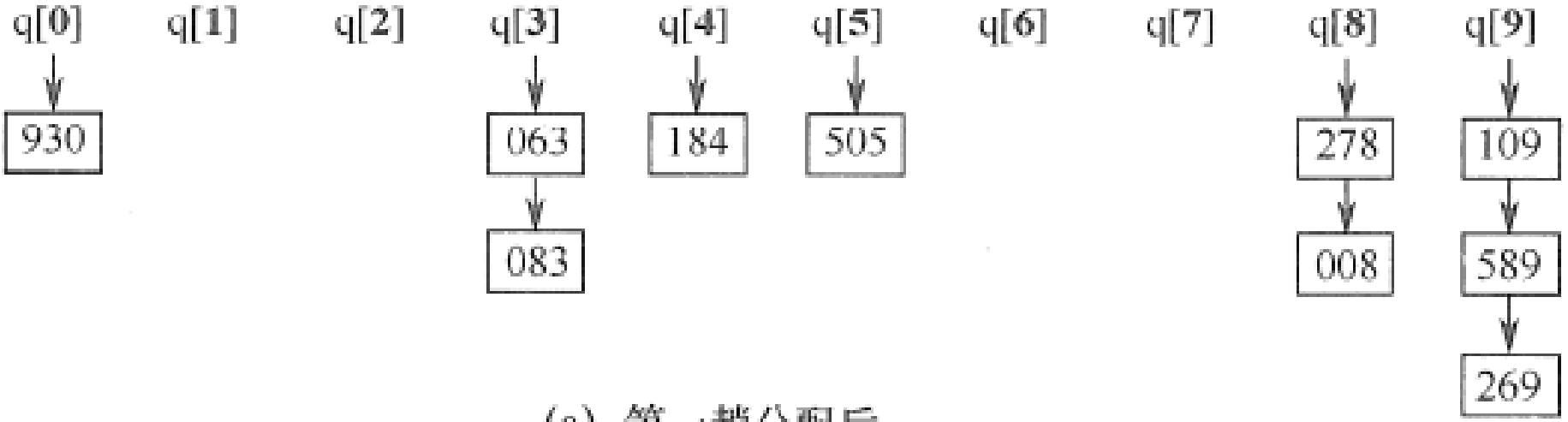
对 $i = 0, 1, \cdots, d-1$，依次做一次“分配”和“收集”（其实是一次稳定的排序过程）。

分配：开始时，把 $Q_0, Q_1, \cdots, Q_{r-1}$ 各个队列置成空队列，然后依次考察线性表中的每个结点 a_j（$j = 0, 1, \cdots, n-1$），若 a_j 的关键字 $k_j^i = k$，就把 a_j 放进 Q_k 队列中。

收集：把 $Q_0, Q_1, \cdots, Q_{r-1}$ 各个队列中的结点依次首尾相接，得到新的结点序列，从而组成新的线性表。

通常采用链式基数排序，假设对如下 10 个记录进行排序：

基数排序

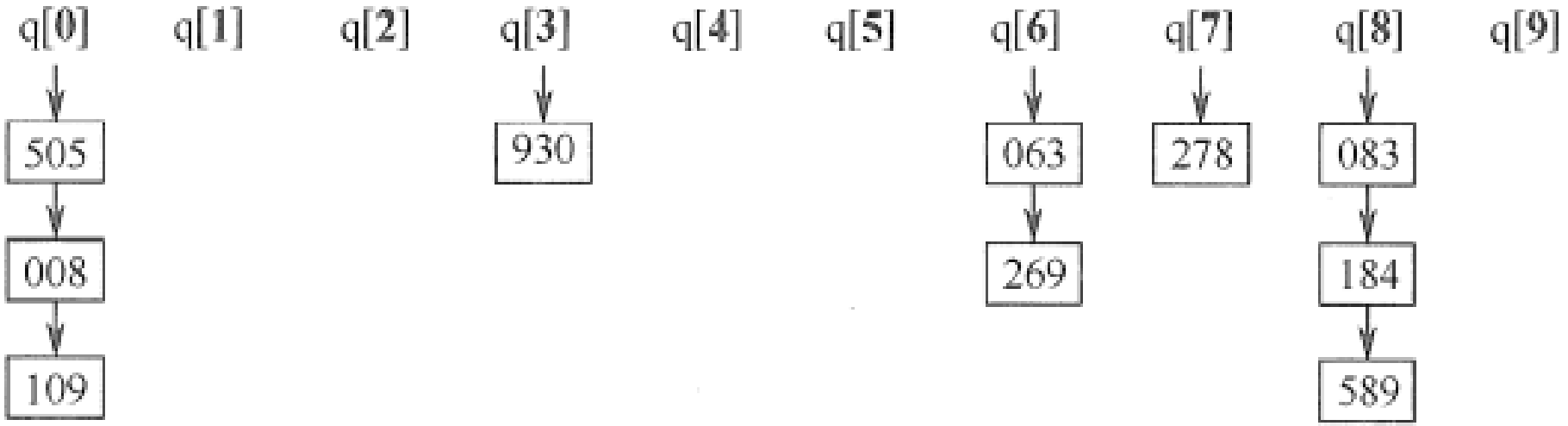


(a) 第一趟分配后

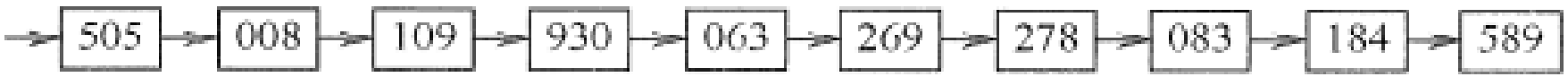
930 → 063 → 083 → 184 → 505 → 278 → 008 → 109 → 589 → 269

(b) 第一趟收集后

基数排序

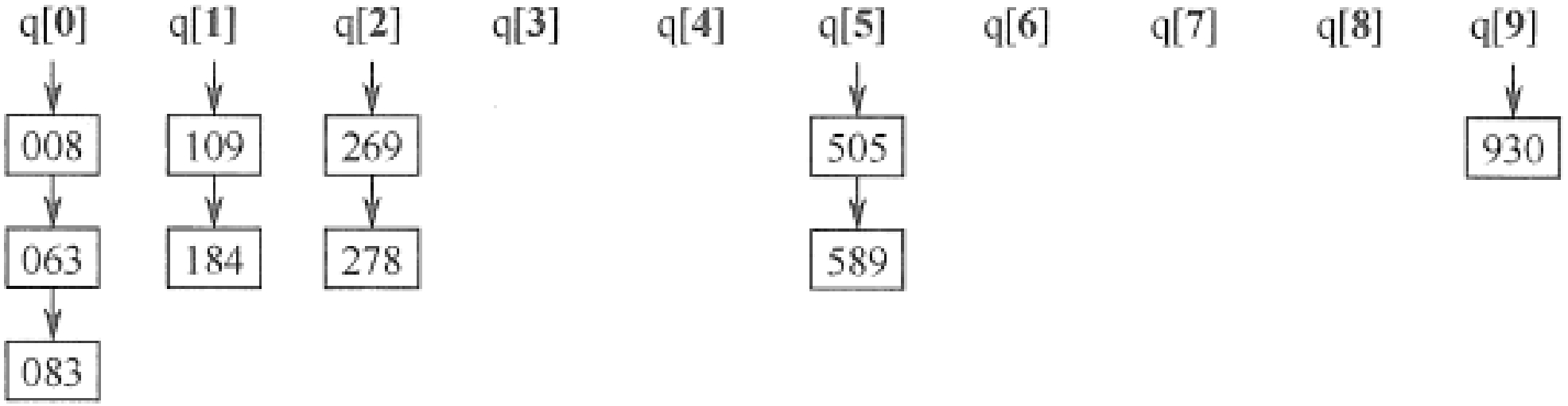


(a) 第二趟分配后

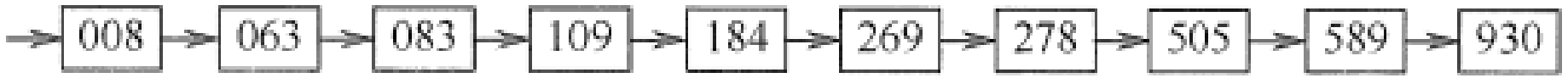


(b) 第二趟收集后

基数排序



(a) 第三趟分配后



(b) 第三趟收集后

基数排序

空间效率：一趟排序需要的辅助存储空间为 r（r 个队列：r 个队头指针和 r 个队尾指针），但以后的排序中会重复使用这些队列，所以基数排序的空间复杂度为 $O(r)$。

时间效率：基数排序需要进行 d 趟分配和收集，一趟分配需要 $O(n)$，一趟收集需要 $O(r)$，所以基数排序的时间复杂度为 $O(d(n+r))$，它与序列的初始状态无关。

稳定性：对于基数排序算法而言，很重要一点就是按位排序时必须是稳定的。因此，这也保证了基数排序的稳定性。

Quick Sort

	Average Run Time	Worst-case Run Time	Average Memory	Worst-case Memory
Merge Sort	$\Theta(n \ln(n))$		$\Theta(n)$	
Quicksort	$\Theta(n \ln(n))$	$\Theta(n^2)$	$\Theta(\ln(n))$	$\Theta(n)$

Merge sort splits the array into two sub-lists and sorts them

- It splits the larger problem into two sub-problems based on *location* in the array

Consider the following alternative:

- Chose an object in the array and partition the remaining objects into two groups relative to the chosen entry

Quick Sort

Median-of-three

Consider another strategy:

- Choose the median of the first, middle, and last entries in the list

If we choose a random pivot, this will, on average, divide a set of n items into two sets of size $1/4\ n$ and $3/4\ n$, *why?*

Choosing the median-of-three, this will, on average, divide the n items into two sets of size $5/16\ n$ and $11/16\ n$

- Median-of-three helps speed the algorithm
- This requires order statistics:

$$2\int_0^{\frac{1}{2}} x \cdot \left(6x(1-x)\right)dx = \frac{5}{16} = 0.3125$$

Quick Sort

Problem 4(4pts): Prove that: When performing quicksort, if the array is **equally** divided into two parts, the time complexity for quicksort would be $O(nlog(n))$.

Solution:

$$
\begin{aligned}
(1) \quad & T(n) = 2T(n/2) + O(n) \\
(2) \quad & = 2(2T(n/4) + O(n/2)) + O(n) \\
(3) \quad & = 4T(n/4) + 2O(n/2) + O(n) \\
(4) \quad & = 4T(n/4) + 2O(n) \\
(5) \quad & = ... \\
(6) \quad & = O(nlog(n))
\end{aligned}
$$

Heap Sort

- A kind of selection sort
  - Place the objects into a heap
    - $O(n)$ time
  - Repeatedly popping the top object until the heap is empty
    - $O(n \ln(n))$ time
  - Time complexity: $O(n \ln(n))$

总结

算法种类	时间复杂度			空间复杂度	是否稳定
	最好情况	平均情况	最坏情况		
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	是
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	是
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	否
希尔排序				$O(1)$	否
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(\log_2 n)$	否
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	否
2 路归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	是
基数排序	$O(d(n+r))$	$O(d(n+r))$	$O(d(n+r))$	$O(r)$	是

历年真题

4) 冒泡排序（Bubble sort）算法中的比较次数与初始元素序列的排列无关。（　　）

5) 在排序算法中，快速排序（Quick sort）的执行时间一定最短。（　　）

历年真题

Given an array that has been almost sorted in ascending order, to sort this array in ascending order, which of the following sorting algorithm would be the most efficient in running time?

A. Insertion sort
B. Merge sort
C. Quicksort
D. None of the above

如果给定的一个数组已经基本上按升序排列，要把这个数组排为升序，下面哪一个排序算法在运行时间上最高效？

A. 插入排序
B. 归并排序
C. 快速排序
D. 以上都不正确

历年真题

3. If array P = [5, 8, 3, 1, 10, 14, 12, 11] is the output of running the partition function in Quicksort, which of the following elements is the pivot at its final position?

   如果数组 P = [5, 8, 3, 1, 10, 14, 12, 11] 是快速排序算法里的分区函数的输出，下面哪个元素是已经被放在其最终位置的枢轴？

   A. 1
   B. 8
   C. 10
   D. 11

历年真题

9. Which of the following is true of merge sort and quicksort? （ ）

   A. Both are divide and conquer algorithms

   B. Both take Θ(n) time in each divide step, given an input array of size n

   C. Both have the same worst-case time complexity

   D. All of the above are true

   下列关于归并排序和快速排序的陈述哪一项是对的？（ ）

   A. 都是用分治法

   B. 给定一个长度为 n 的输入数组，都是需要Θ(n)的时间进行每一次切分

   C. 都有相同的最坏时间复杂度

   D. 以上都对

10. Which of the following is the closest to the minimum number of comparisons between array elements that any comparison-based algorithm needs to perform to find both the minimum and maximum values in an input array of size n? （ ）

    任何一个基于比较的算法在一个长度为 n 的数组中同时找出最大和最小值，至少需要数组元素之间的比较的次数与下列哪个最接近？（ ）

    A. n

    B. 1.5n

    C. 2n

    D. n^2

历年真题

1) Write the pseudocode of the merge function of the merge sort algorithm, to merge two lists sorted in ascending order into one list sorted in ascending order, of totally n elements. Analyze the worst-case time complexity of the merge function. (5 points)
写出归并排序算法里的归并函数的伪代码，把两个总共包含 n 个元素的已经排为升序的列表合并成一个升序的列表。分析这个归并函数在最坏情况下的时间复杂度。（5 分）

2) Suppose you are given k lists with a total of n distinct elements, where k > 2. Each list has been sorted in descending order. Design an algorithm using the strategy of Merge sort to merge the k lists into one list sorted in descending order, in running time $O(n\log_2 k)$. Write the pseudocode of your algorithm and analyze its worst-case time complexity. (7 points)
假如给定 k 个总共包含 n 个不同元素的列表，这里 k>2。每个列表已经排为降序。用归并排序算法的策略设计一个算法，将这 k 个列表在 $O(n\log_2 k)$ 时间内归并到一个降序排列的列表。写出你设计的算法的伪代码并分析它的最坏时间复杂度。（7 分）

3) Design an algorithm using a maximizing heap to solve the problem of merging k lists into one list in 2), in running time $O(n\log_2 k)$. Assume that the subroutines of Heapsort algorithm using a maximizing heap are already given. Write the pseudocode of your algorithm and analyze its worst-case time complexity. (8 points)
设计一个算法，用一个最大化堆来解决题 2）提到的归并 k 个列表的问题，使得时间复杂度是 $O(n\log_2 k)$。假设使用最大化堆的堆排序算法的子程序都已经给定。写出你的算法的伪代码，并分析它的最坏时间复杂度。（8 分）

历年真题

2) 下列 5 段代码（(a)，(b)，(c)，(d)，(e)）描述了一个排序算法，但 5 段代码的排序被打乱了，请仔细阅读代码，并写出算法实现的正确顺序。（5 分）

(a)
```
List sort(List input){
    if(input.length <= 1) {return input.copy();}
```

(b)
```
List sorted_left = left.sort();
List sorted_right = right.sort();
sorted_left.append(p);
sorted_left.append(sorted_right);
```

(c)
```
for (int i=0; i<input.length; i++){
    if (input[i] < p) {left.append(input[i]);}
    else {right.append(input[i]);}
}
```

(d)
```
int p = input.poplast(); // Now, 'p' no longer exists in 'input'
List left = new List();
List right = new List(); // 'left' and 'right' are empty
```

(e)
```
cout << sorted_left << endl;
// print all elements in the array from left to right
delete sorted_right;
return sorted_left;}
```

3) 问题 2）中所描述的算法与以下哪种排序算法最接近？（3 分）

A. 冒泡排序
B. 插入排序
C. 归并排序
D. 快速排序

4) 请使用问题 2）中所描述的排序算法对数字序列 9,7,3,5,1,8,4,2,6 进行排序，并写出算法完成后所有显示在电脑显示屏上的结果。（8 分）

历年真题

7. Quicksort (25 points) 快速排序（25 分）

Suppose we apply the quicksort algorithm to an array E of n integers, to sort the n elements in ascending order. The time complexity is measured in the number of comparisons between array elements, and each comparison takes constant time.

假设我们把快速排序算法用在一个包含 n 个整数的数组 E，将这 n 个元素排为升序。我们用元素之间的比较次数来衡量时间复杂度，而每一次元素间的比较需要常数时间。

1) Suppose a version of quicksort always chooses the left-most element in an array to be the pivot. Give a worst-case input array of size n for this quicksort algorithm, represented as a permutation of all integers in set {1, 2, …, n}. Derive the asymptotic time complexity of the algorithm on this input in terms of Θ. （8 points）

   假设一个快速排序算法的版本总是选数组里最左边的元素作为基准。对这样一个快速排序算法，找到一个最坏情况的包含 n 个元素的输入数组，用集合{1, 2, …, n}中的所有整数的一个排列来表示。对这样的输入，推导这个算法的渐近时间复杂度，用Θ来表示。（8 分）

2) As a recursive algorithm, quicksort tends to call itself for many small subarrays, which would compromise the time efficiency. It is known that the Insertion sort algorithm is fast on small input arrays. Describe a method to modify the quicksort algorithm by using Insertion sort, such that the above issue can be addressed. Suppose each subarray on which Insertion sort is applied has no more than k elements. Show that the modified algorithm has average-case time complexity of O(nk + nlg(n / k)). （9 points）

   作为一个递归算法，快速排序算法一般要对很多小的子数组进行递归调用，从而影响了时间效率。 而插入排序算法在小数组上运行效率很高。描述一个修改快速排序算法的方法，利用插入排序算法来解决上述的问题。假设插入排序算法处理的每一个子数组的长度不超过 k 个元素。推导证明这个修改过的快速排序算法的平均时间复杂度是 O(nk + n lg(n / k))。（9 分）

3) Suppose that quicksort uses the median of E[first], E[middle] and E[last] as the pivot. Show that the number of comparisons between array elements performed by this quicksort algorithm is no more than $\frac{n^2}{4} + o(n^2)$ in the worst case. （8 points）

   假设快速排序算法总是从输入数组的第一个、最后一个、和中间位置这三个元素中选择中位数作为基准。证明该快速排序算法最坏情况下完成排序需作不超过 $\frac{n^2}{4} + o(n^2)$ 次元素间的比较。（8 分）

Q&A