排序 I

吉林大学计算机学院
谷方明
fmgu2002@sina.com

学习目标

- 掌握排序的概念和术语
- 掌握排序算法的度量指标
- 掌握基于插入的排序算法：插入排序和**Shell**排序
- 掌握基于交换的排序算法：冒泡排序和快速排序
- 掌握基于选择的排序算法：选择排序和堆排序
- 掌握归并排序算法

排序(Sorting)

- **Sorting**
  - ✓ 英文词典：按种类安排事物的过程；
  - ✓ 计算机：把事物排成递增或递减的次序；
- **排序是一个重要的实际问题**
  - ✓ 分班、评奖学金、保研等；
- **排序是一个经典的有趣课题**
  - ✓ 发现了很多巧妙的算法（冒泡排序1956，…， Library Sort 2004，…）；程序设计、算法分析的经典实例
  - ✓ 还有很多有魅力悬而未决的难题；

基本概念

- 文件：给定待排序的 n 个数据对象，R_1，R_2，…，R_n，称这些数据对象为记录，并称这 n 个记录的集合为一个文件；
- 关键词域：通常数据对象包括多个属性域，可将其中的一个属性域作为排序的依据，称其为关键词域（Key）。K_1，K_2，…，K_n；
- 排序:就是将一组杂乱的数据按关键词域排列起来的过程。（确定记录的一个排列p(1),p(2),…，p(n)，默认按递增序$K_{p(1)}$<= $K_{p(2)}$<= … <= $K_{p(n)}$）

排序算法的重要度量指标

- 时间复杂度
  - ✓ 关键词的比较次数 和 数据的移动次数
- 空间复杂度
  - ✓ 排序过程使用的辅助存储空间
- 稳定性
  - ✓ 关键词相同的任意两个记录，排序前后相对位置保持不变，则称排序算法是稳定的。
  - ✓ 即 若$K[i] = K[j]$ 且 $i < j$, 则 $p(i) < p(j)$.

插入排序

- 思想：将一个记录插入到已排好序的有序表中，从而得到一个新的有序表且记录个数增一。

- 例：

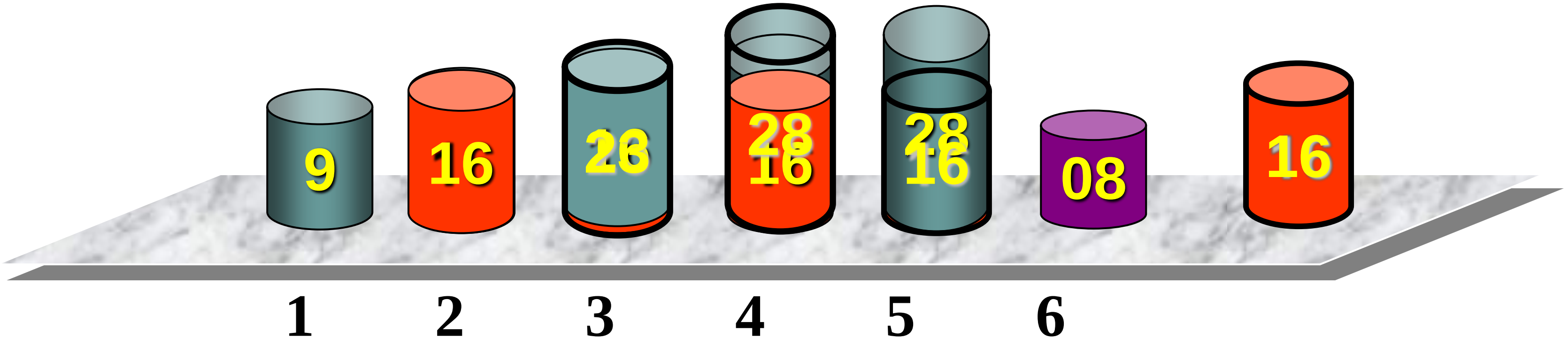
原有序表：　(9　15　　23　28　37) 20

找插入位置 : (9　15 ↑ 23　28　37) 20

新有序表:　　(9　15　20　23　28　37)

直接插入排序：一次插入过程

- 找插入位置，从后向前，边找边移动
- 例：j = 5



- 原有序表中关键词比 R_j 大的记录数：d_j
- 比较次数：d_j+1　　　　移动次数：d_j+2

时间复杂度分析

设 d_j 是 R_j 左边关键词大于K_j 的记录个数，则插入算法中关键词的比较次数为：

$$\Sigma_{j=2\ldots n}(1+d_j)=n-1+\Sigma_{j=2\ldots n}d_j$$

记录的移动次数为

$$\Sigma_{j=2\ldots n}(2+d_j)=2(n-1)+\Sigma_{j=2\ldots n}d_j$$

最好情况和最坏情况

- 最好时间复杂度

  排序前记录已按关键词从小到大排列，即 $d_j = 0$. 每趟只需与前面的有序序列的最后一个记录的关键词比较 1 次，记录移动 2 次，总的关键词比较次数为 $n-1$，记录移动次数为 $2(n-1)$.

- 最 坏时间复杂度

  排序前记录已按关键词逆序排列，即$d_j = j-1$. 总的比较次数为$n-1+n(n-1)/2$，总的移动次数为$2(n-1)+ n(n-1)/2$.

期望时间复杂度

考察分析 $\Sigma_{j=2 \ldots n} d_j$ 的期望值。

对于序列 K_1，K_2，…，K_n，如果 $1 \le i < j \le n$，且$K_i > K_j$，则称（K_i，K_j）为上述序列的一个反序对.实际上，$\Sigma_{j=2 \ldots n} d_j$ 正好是序列 K_1，K_2，…，K_n 的反序对个数。反序对的平均个数为

0+ (0+1)/2+ (0+1+2)/3+

(0+1+2+3)/4+(0+1+2+3+4)/5+

(0+1+2+3+4+5)/6+ … +(0+1+2+…+n–1)/n

= 0+1/2+2/2+ 3/2+4/2+5/2+…+(n–1)/2

=n(n–1)/4

直接插入排序小结

	比较次数	记录移动次数
最好	n–1	2n–2
平均	(n–1)(n+4)/4	(n–1)(n+8)/4
最坏	(n–1)(n+2)/2	(n–1)(n+4)/2

- 优点：算法简单.
- 缺点：期望和最坏复杂度为 $O(n^2)$.
- 稳定性：稳定
- 辅助空间： $O(1)$

希尔排序(shell)

- 将记录按下标的一定增量分组，对每组使用直接插入排序算法排序；
- 随着增量值逐渐减少，每组包含的关键词越来越多；当增量值减少到 1 时，整个文件恰被分成一组，算法终止。

[例] 将十个数进行希尔排序的示例。

下标	0	1	2	3	4	5	6	7	8	9
	36	25	48	12	65	25	43	58	76	32

$d_1=5$

	25					36				
		25					43			
			48					58		
				12					76	
					32					65

一趟排序结果	25	25	48	12	32	36	43	58	76	65

[例] 将十个数进行希尔排序的示例

下标	0	1	2	3	4	5	6	7	8	9
	25	25	48	12	32	36	43	58	76	65

$d_2=2$

25 32 43 48 76

12 25 36 58 65

二趟排序结果	25	12	32	25	43	36	48	58	76	65
三趟排序结果	12	25	25	32	36	43	48	58	65	76

$d_3=1$

希尔排序增量选取方法

- $d_1 = n / 2 = 10 / 2 = 5$
- $d_2 = d_1 / 2 = 5 / 2 = 2$
- $d_3 = d_2 / 2 = 2 / 2 = 1$

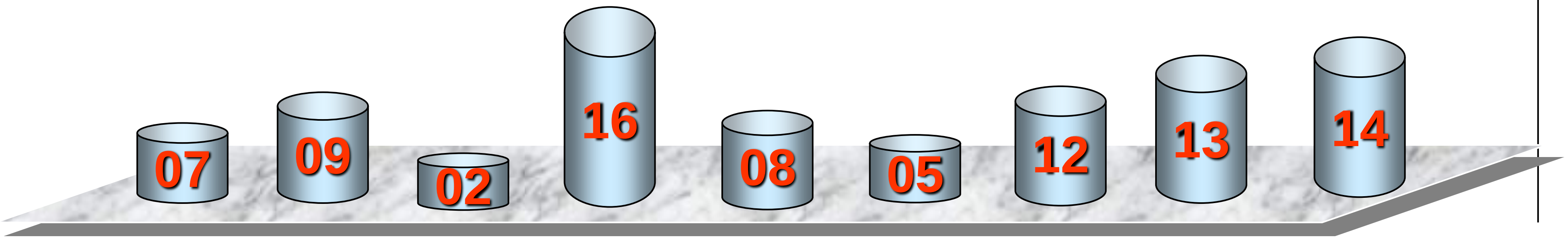
- $d_1 = n / 2$
- $d_{i+1} = d_i / 2$

Shell排序算法分析

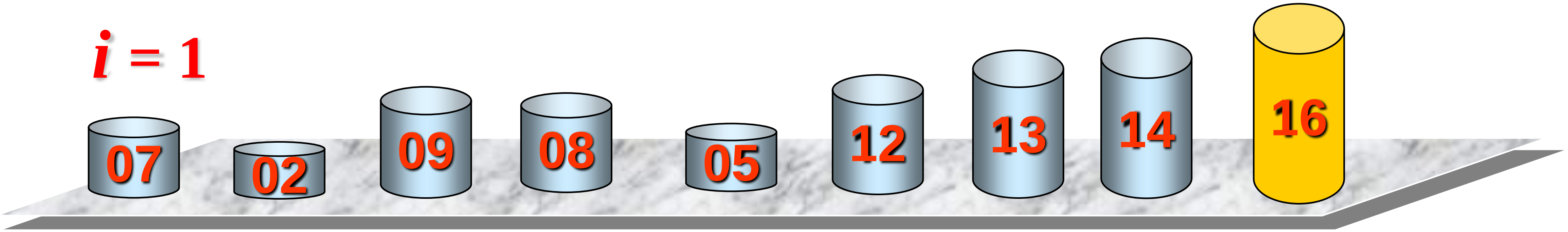
- Shell算法的性能与所选取的分组长度序列有很大关系。关键词比较次数（记录移动次数）与增量选择间的关系，至今仍然是数学难题。
- 1969年，普拉特（V. Pratt）证明了如下结论：如果渐减增量序列取值为形如 2^p3^q且小于n的所有自然数的集合, 即$\{2^p3^q|2^p3^q<n\}$, 则 Shell 算法的时间复杂度$O(n(\log_2 n)^2)$.
- Knuth从大量的实验统计资料中得出：当 n 很大时，关键词的平均比较次数大约是$n^{1.25}$，记录的平均移动次数大约是 $1.6n^{1.25}$.
- Shell算法不稳定

交换排序

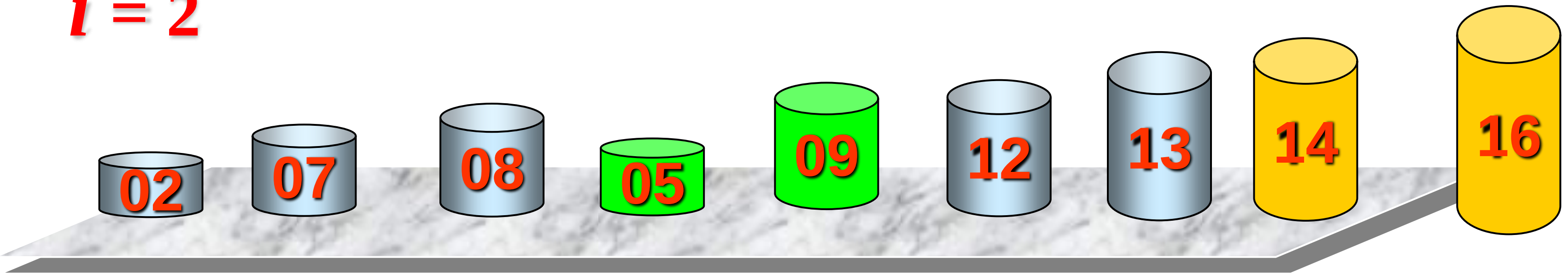
- 交换排序思想：交换文件中存在的反序对，直到不存在反序对为止.
- 冒泡排序：通过比较相邻记录的关键词，交换存在逆序的记录；使关键词较大的记录如气泡一般逐渐往上“飘移”直至浮出“水面”。


07
09
02
16
08
05
12
13
14

$i = 1$


07
02
09
08
05
12
13
14
16

$i = 2$


02
07
08
05
09
12
13
14
16

冒泡排序算法分析

算法BSort (*R*，*n*)

BS1 [冒泡过程]

for (*i* = *n* ; i >1 ; i --)

for (*j* = *1* ; j <i ; j ++)

if($K_j > K_{j+1}$) swap(R_j , R_{j+1});

- 关键词的比较次数:

  $(n-1)+(n-2)+\ldots+1= (n-1)n/2$

冒泡排序算法的改进

算法Bubble (*R*，*n*)

Bubble1. [初始化]

    BOUND = *n*; //每趟冒泡关键词比较的终止位置

Bubble2. [冒泡过程]

    while (BOUND > 0) { // BOUND=0, 终止算法

        t = 0; //每趟冒泡记录交换的最后位置

        for (*j* = *1* ; j <BOUND ; j ++)

            if ($K_j > K_{j+1}$) { swap(R_j , R_{j+1}); t = j; }

        BOUND = *t*;

    }

	第1趟	第2趟	第3趟	第4趟	第5趟	第6趟	第7趟	8趟	9趟
13	16	16	16	16	16	16	16	16	16
14	13	15	15	15	15	15	15	15	15
12	14	13	14	14	14	14	14	14	14
10	12	14	13	13	13	13	13	13	13
08	10	12	12	12	12	12	12	12	12
03	08	10	11	11	11	11	11	11	11
06	03	08	10	10	10	10	10	10	10
11	06	03	08	09	09	09	09	09	09
05	11	06	03	08	08	08	08	08	08
15	05	11	06	03	07	07	07	07	07
04	15	05	09	06	03	06	06	06	06
16	04	09	05	07	06	03	05	05	05
01	09	04	07	05	05	05	03	04	04
09	01	07	04	04	04	04	04	03	03
02	07	01	02	02	02	02	02	02	02
07	02	02	01	01	01	01	01	01	01

算法分析

- 最好情形：记录的初始排列按关键词从小到大排好序时，此算法只执行一趟起泡，做 **n-1** 次关键词比较，不发生记录交换；
- 最坏情形：算法执行了**n-1**趟起泡，第 **i** 趟 **(1≤ i< n)** 做了 **n- i** 次关键词比较，执行了**n-i** 次记录交换,此时，总的关键词比较次数和记录交换次数为**(n-1)n/2**

一般情形

- 假定记录序列 $R_1, R_2, \ldots, R_n$ 所对应的关键词序列为 $A = \{ K_1, K_2, \ldots, K_n\}$，令 A 中第 j 小的关键词 K' 对应的记录为 R'，在 R' 左边诸记录对应的关键词中，大于K'的关键词个数为 $b_j(1 \le j \le n)$，则文件$B=\{ b_1, b_2, \ldots, b_n \}$ 被称为 A 的反序表.

	1	2	3	4	5	6	7	8	9
A	07	09	02	16	08	05	12	14	13
b_j	0	0	2	0	2	4	1	1	2
j	3	5	1	9	4	2	6	8	7
B	2	4	0	2	0	1	2	1	0

定理7.1

设$\{K_1$，K_2，…，$K_n\}$是序列$\{1$，2，…，$n\}$的一个排列，$\{b_1$，b_2，…，$b_n\}$是对应的反序表．如果算法Bubble的一趟冒泡把$\{K_1, K_2,\ldots, K_n\}$改变为$\{K_1', K_2',\ldots, K_n'\}$, 那么将$\{b_1, b_2,\ldots, b_n\}$中每个非零元素减1, 就得到相应的反序表$\{b_1',b_2',\ldots,b_n'\}$.

- A ={07, 09, 02, 16, 08, 05, 12, 14, 13}
- B ={2, 4, 0, 2, 0, 1, 2, 1 , 0}
- A′={07, 02, 09, 08, 05, 12, 14, 13 , 16}
- B′={1, 3, 0, 1, 0, 0, 1, 0 , 0}

推论7.1

- 冒泡(Bubble)排序算法的
  冒泡趟数 $A=1+\max\{b_1, b_2, \ldots, b_n\}$;
  记录交换次数 $B=\sum_{i=1\ldots n} b_i$;
  关键词比较次数 $C=\sum_{i=1\ldots A} C_i$, 其中 C_i 等于第 i 趟冒泡时的BOUND值减 1.

	比较次数	交换次数	冒泡趟数
最好情况	$n-1$	0	1
平均情况	$n/2(n-\log_e n)+O(n)$	$n(n-1)/4$	$n-(\pi n/2)^{1/2}+O(1)$
最坏情况	$n(n-1)/2$	$n(n-1)/2$	n

冒泡排序算法小结

- 时间复杂度：$O(n^2)$（最坏和平均）．
- 辅助存储空间：$O(1)$.
- 稳定性：冒泡排序是稳定的排序方法。

鸡尾酒排序

- 交替进行气泡上浮和气泡下沉的排序方法

	第一趟	第二趟	第三趟	第四趟	第五趟	第六趟
79	90	90	90	90	90	90
56	79	88	88	88	88	88
90	56	79	79	79	79	79
4	88	56	56	56	56	56
32	4	35	35	35	35	35
27	32	4	32	32	32	32
16	27	32	4	27	27	27
88	16	27	27	4	16	16
35	35	16	16	16	4	4

图7.5 单纯上浮排序算法的运行过程

该例需要扫描 6 趟

	第一趟	第二趟	第三趟	第四趟
79	90	90	90	90
56	79	79	88	88
90	56	88	79	79
4	88	56	56	56
32	4	32	35	35
27	32	27	32	32
16	27	16	27	27
88	16	35	16	16
35	35	4	4	4

图7.6 交替上浮下沉排序算法，需四趟完成排序.

对相同的文件，上浮方法比交替方法需多用两趟才能完成排序

快速排序（分划交换排序）

- 任取待排序文件中的某个记录(如第一个记录)作**基准(pivot)**，按照该记录的关键词大小，将整个文件分划为左右两个子文件：
  - 左侧子文件中所有的关键词都 ≤ 基准记录的关键词
  - 右侧子文件中所有的关键词都 > 基准记录的关键词
  - 基准记录排在两个子文件中间。
- 分别对两个子文件重复上述方法，直到所有记录都排在相应位置上为止。

一趟分划交换

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
70	73	69	23	93	18	11	68	100
	i(第一个大于)						(第一个小于) j	
70	68	69	23	93	18	11	73	100
				i		j		
70	68	69	23	11	18	93	73	100
					j	i		
70	68	69	23	11	18	93	73	100
18	68	69	23	11	70	93	73	100

算法QSort(R, m, n)

/* R_m 为基准记录, $K_{n+1}= +\infty$; m、n分别为R的左右边界*/

QSort1.[递归出口]

  if (m >= n) return ;

QSort2.[交换 , 分划]

  i = m , j = n+1. $K=K_m$.

  while(i < j) {

    i ++ ; while ($K_i < K$)   i++ ;

    j-- ; while ($K_j > K$) j-- ;

    if (i < j) swap (R_i , R_j); }

  swap (R_m , R_j)

QSort3.[递归调用]

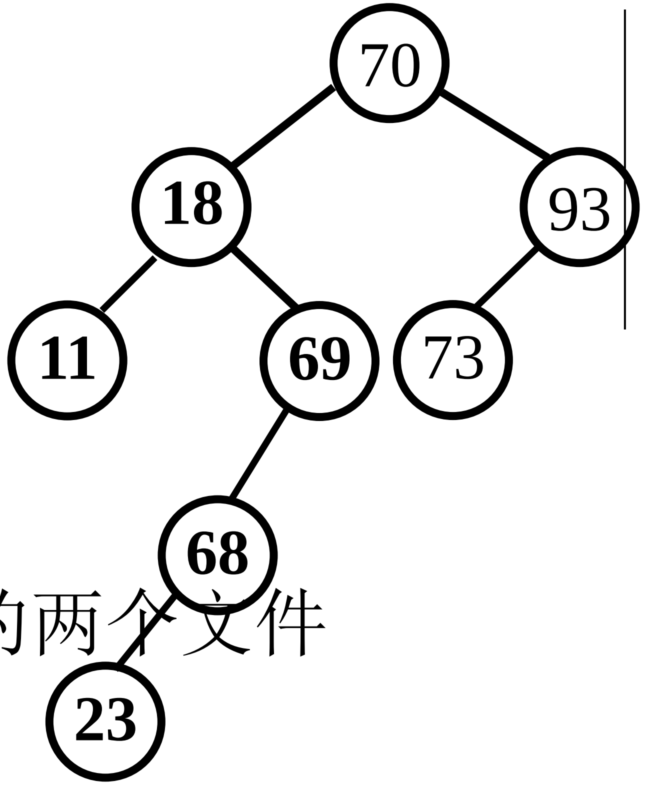
  QSort (R , m , j–1);

  QSort (R , j+1, n); ▌

分析

- 正确性
  - ✓ 数学归纳法
- 时间复杂度
  - ✓ 基本运算：关键字比较
  - ✓ 分析工具：递归调用树

递归调用树



- 定义
  - 根结点代表基准元素
  - 左右子树分别代表小于和大于基准元素的两个文件
- 一个结点代表一次递归调用
- 一次分划过程(R, m, n)，i > j 时的两个记录与基准记录比较两次，其余的记录和基准记录各比较一次，关键词的比较次数为 n – m + 2或(n+1).
- 快速排序的时间效率取决于递归树的深度.

最好时间复杂性

- 每次分划后，都得到长度相等的两个子序列。总的计算时间为：

$$T(n) = \begin{cases} a & n \le 1,\ a\text{为常数} \\ f(n) + 2 \times T(n/2) & n > 1 \end{cases}$$

- **T(*n*) ≤ *cn* + 2 T(*n*/2)　　// *c*是一个常数**

$$\le cn + 2\,(\,cn/2 + 2T(n/4)\,) = 2cn + 4T(n/4)$$

$$\le 2cn + 4\,(\,cn/4 + 2T(n/8)\,) = 3cn + 8T(n/8)$$

………

$$\le cn \log_2 n + nT(1) = O(\,n \log_2 n\,)$$

最坏时间复杂性

- 最坏情况：即待排序记录已按关键词由小到大(或由大到小)排好序，其递归树为单支树，每次分划只得到比上次少一个记录的子序列。此时, 排序共需 $n-1$ 趟, 且第 i 趟需要 $n-i+2$ 次关键词比较，总关键词比较次数为：

$$(n+1)+n+\ldots+3=(n-1)(n+4)/2$$

- 定理7.2　如果规定关键词比较为基本运算，则算法QSort(1, n) 的期望复杂度为$O(n\log_2 n)$，最坏复杂度 $W_n=(1/2)n^2+(3/2)n-2$.

期望时间复杂性

- 设 E_n 是算法Qsort的期望时间复杂度，n是元素数；假定关键词的分布是随机的；

  $E_0 = E_1 = 0$

  $E_n = \sum p_s(n + 1 + E_{s-1} + E_{n-s}) \quad (n \geqslant 2)$

- $E_n = (2\ln 2)\, n\log n + O(n) = 1.386\, n\log n$
  - ✓ $n E(n) = n(n+1) + 2\sum_1^n E(k-1)$
  - ✓ $\frac{E(n)}{n+1} = \frac{E(n-1)}{n} + \frac{2}{n+1}$

快速排序小结

- 时间复杂度
  - ✓ 最坏情况的时间复杂度：$O(n^2)$
  - ✓ 平均情况的时间复杂度：$O(n\log_2 n)$
- 辅助空间
  - ✓ 最好：$O(\log_2 n)$
  - ✓ 最坏：$O(n)$
- 稳定性：快速排序是不稳定的排序方法。
- 实现
  - ✓ 留空法
  - ✓ ……

快速排序改进

- 用一个随机函数选择用于控制分划的记录，但随机数的产生也很费时。
- 三者取中法，每个待排序记录序列的第一个记录、最后一个记录和位置接近正中的3个记录，取其关键词居中者为基准记录。即令K_m是K_m、$K_{\lfloor (m+n)/2 \rfloor}$和$K_n$的中间值。

算法Partition2(R，m，n．j) //三者取中法一次分划过程

Part1.[选中间值元素]

mid = (m+n)/2;

if $(K_{mid} > K_n)$ **swap(** R_{mid}, R_n **)**

if $(K_m > K_n)$ **swap(** R_m, R_n **)**

if $(K_{mid} > K_m)$ **swap(** R_{mid}, R_m **)**

Part2.[用 K_m **分划(** $R_m, \ldots, R_n$ **)]**

i = m． j = n+1． K = K_m **.**

while (i<j) {

i = i + 1; while($K_i < K$ **) i=i+1;**

j = j – 1; while($K_j > K$ **) j=j–1;**

if (i < j) swap(R_i, R_j **)**

swap(R_m, R_j **)** ▌ //三者取中依然可能退化，如34165

非递归快速排序算法

算法 HSort(n, R, M)

/* 变量M已给定，5≤M≤15．(R_1, R_2, …, R_n)为待排序文件 */

HSort1.[初始化]

CREATEStack(S) .

f =1; t=n; K_{n+1}=+∞ ; K_0 = –∞ ;

if (n>=M) S⇐(f, t) ;

HSort2.[对长度≥M的记录序列分划排序]

while (！ S.*empty*()) {

(f, t) ⇐ S.

Partition2(R, f, t . j)．// 三者取中分划文件

if (j-f >= M) S⇐(f, j-1).

if (t-j >= M) S⇐(j+1, t). }

Hsort5.[插入排序]

InsertSort(R, 1, n) .

思考

- 算法HSort中的栈S可能包含的最大元素个数

- Hsort中的栈可以用队列代替吗？

选择排序

思想：

对待排序的文件进行 n–1 次选择操作，其中第 i 次选择第 i 小(或大)的记录放在第 i 个(或第 n–i+1个)位置上。

直接选择排序算法

- 教材方法: 选择第i大，与 n-i+1位置交换
  总的关键词比较次数为
  (n–1)+(n–2) + … + 1= (n – 1)n/2
  记录交换次数等于 n-1
- 时间复杂度: $O(n^2)$(包括最好、最坏和平均).
- 稳定性：不稳定排序（位置交换时被替换元素导致不稳定.）
- 辅助空间： $O(1)$.

堆排序

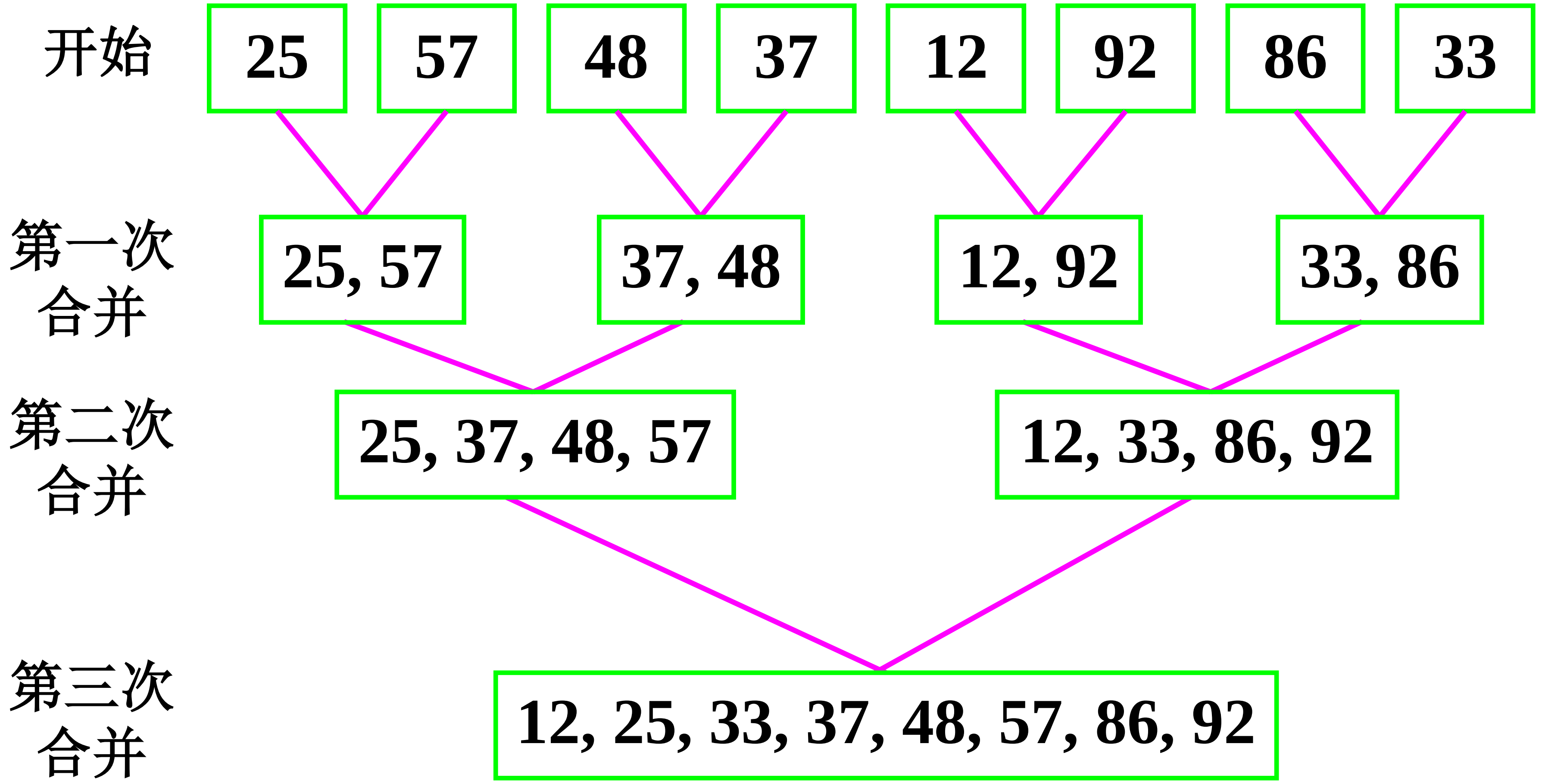
- 使用**down**操作即可
- 时间复杂度:
  - ✓ $T(n) = 2(\lfloor \log n \rfloor + \lfloor \log(n-1) \rfloor + \ldots + \lfloor \log 2 \rfloor) \leqslant 2n\log n$
  - ✓ $T(n) \geq 2(\lfloor \log n \rfloor + \ldots + \log n/2) \geq 2*n/2*\log n/2 \geq n\log n/2$
  - ✓ $O(n\log n)$（最好、最坏和平均）.
- 稳定性：不稳定排序
- 辅助空间： ***O*(1)** .

归并排序

- 合并(归并)：把两个或多个有序文件合并成一个有序文件。

- 例：文件　{ 503, 703, 765 } 和
  文件　{ 087, 512, 677 } 合并
  得到
  文件　{ 087, 503, 512, 677, 703, 765 }.

合并排序过程示例


开始
25
57
48
37
12
92
86
33
第一次合并
25, 57
37, 48
12, 92
33, 86
第二次合并
25, 37, 48, 57
12, 33, 86, 92
第三次合并
12, 25, 33, 37, 48, 57, 86, 92

合并 A 中已排序的两个文件le..mid和 mid+1..ri 到 X 中

```
void
merge( int A[], int le, int mid, int ri, int X[] ){
        int i = le, j = mid+1, k;
        for( k = le ; k <= ri ; k++ )
                if(i<=mid && j<=ri && A[i]<=A[j]
                        || j>ri)        X[k]=A[i++];
                else X[k] = A[ j++ ];
}
```

一趟合并排序

```
void MPass( int A[], int n, int len, int X[] ){
        int i, j;
        for(i=1;i + 2*len-1<=n ; i+=2*len)
                merge(A, i, i+len-1, i+2*len-1,X);
        if( i+len-1 < n) merge(A, i, i+len-1, n, X);
        else for( j = i; j<=n; j++) X[j] = A[j];
}
```

合并排序算法

```
void MSort( int A[], int n, int X[]){
        int len=1;
        while(len < n){
                MPass(A,n,len,X);
                len*=2;
                MPass(X,n,len,A);
                len*=2;
        }
}
```

归并排序算法分析

- **时间复杂度O(nlogn)**
  - ✓ Merge: 基本运算是元素移动，2*len次， $O(len)$.
  - ✓ MPass：要调用Merge函数$\lceil n/(2\times len)\rceil \approx O(n/len)$次，MPass的复杂度为$O(n)$
  - ✓ Msort：调用MPass正好$\lceil \log_2 n\rceil$次，所以算法总的时间复杂度为$O(n\log_2 n)$
- **辅助存储空间：$O(n)$**
- **归并排序是稳定的排序方法。**

归并排序拓展

- 元素移动代价可能很大
  - ✓ 改为指针（课后思考）

- n较小时，合并代价大
  InsertSort(类似HSort)
  前几次**len**小，也可**InsertSort(A,le,le+2len-1)**

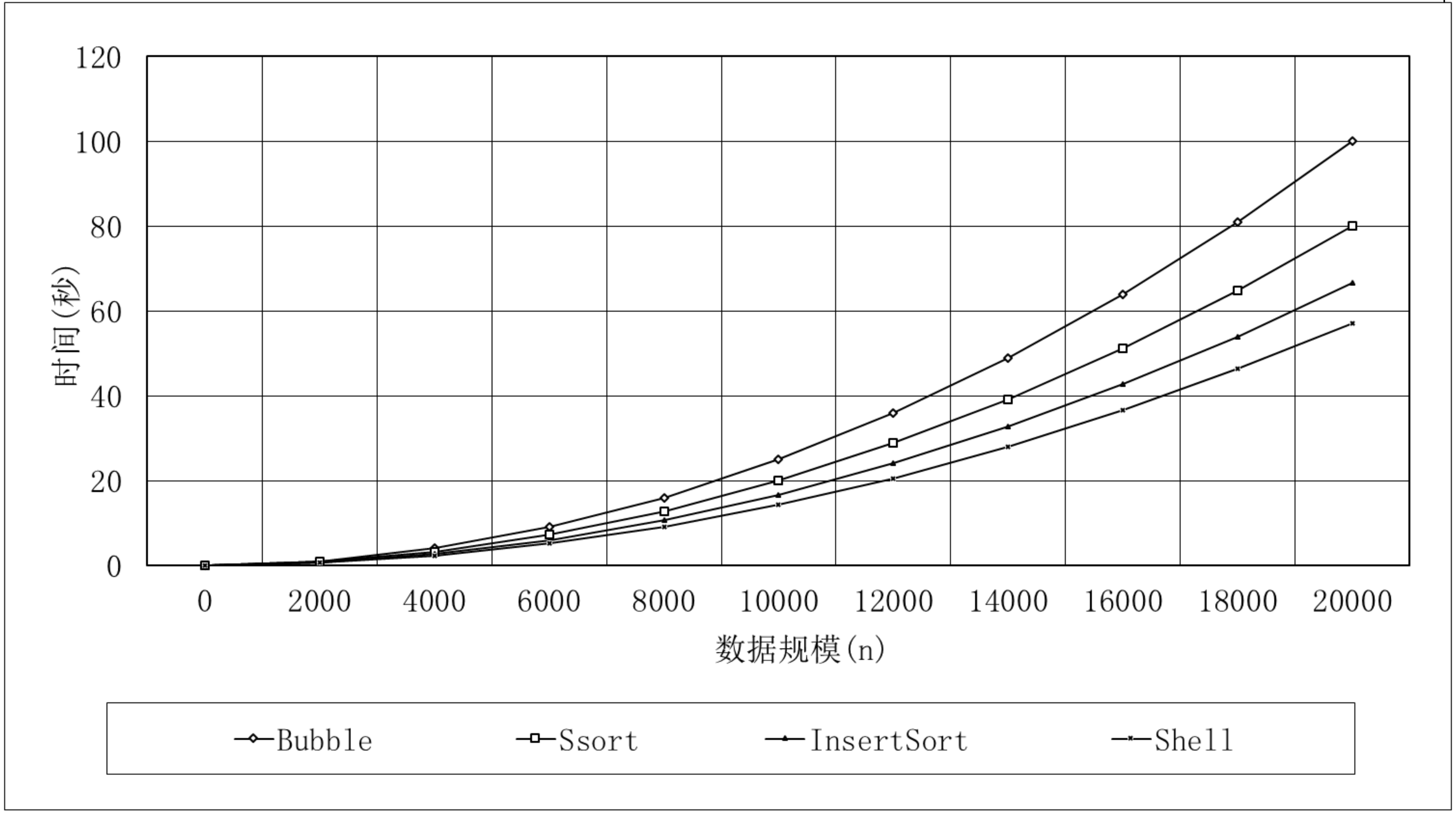
- 递归版
  实现简单，解题常用算法

```
//归并排序递归版，X全局数组
void msort( int A[], int le, int ri){
        int i = le, mid = (le + ri)/2 , j = mid+1, k;
        if(le >= ri) return;
        msort(A, le, mid);
        msort(A, mid+1,ri);
        for( k = le ; k <= ri ; k++ )
                if(i<=mid && j<=ri && A[i]<=A[j]
                        || j>ri)        X[k]=A[i++];
                else X[k] = A[ j++ ];
        for( k = le ; k <= ri ; k++ ) A[k] = X[k];
}
```

内排序方法的比较

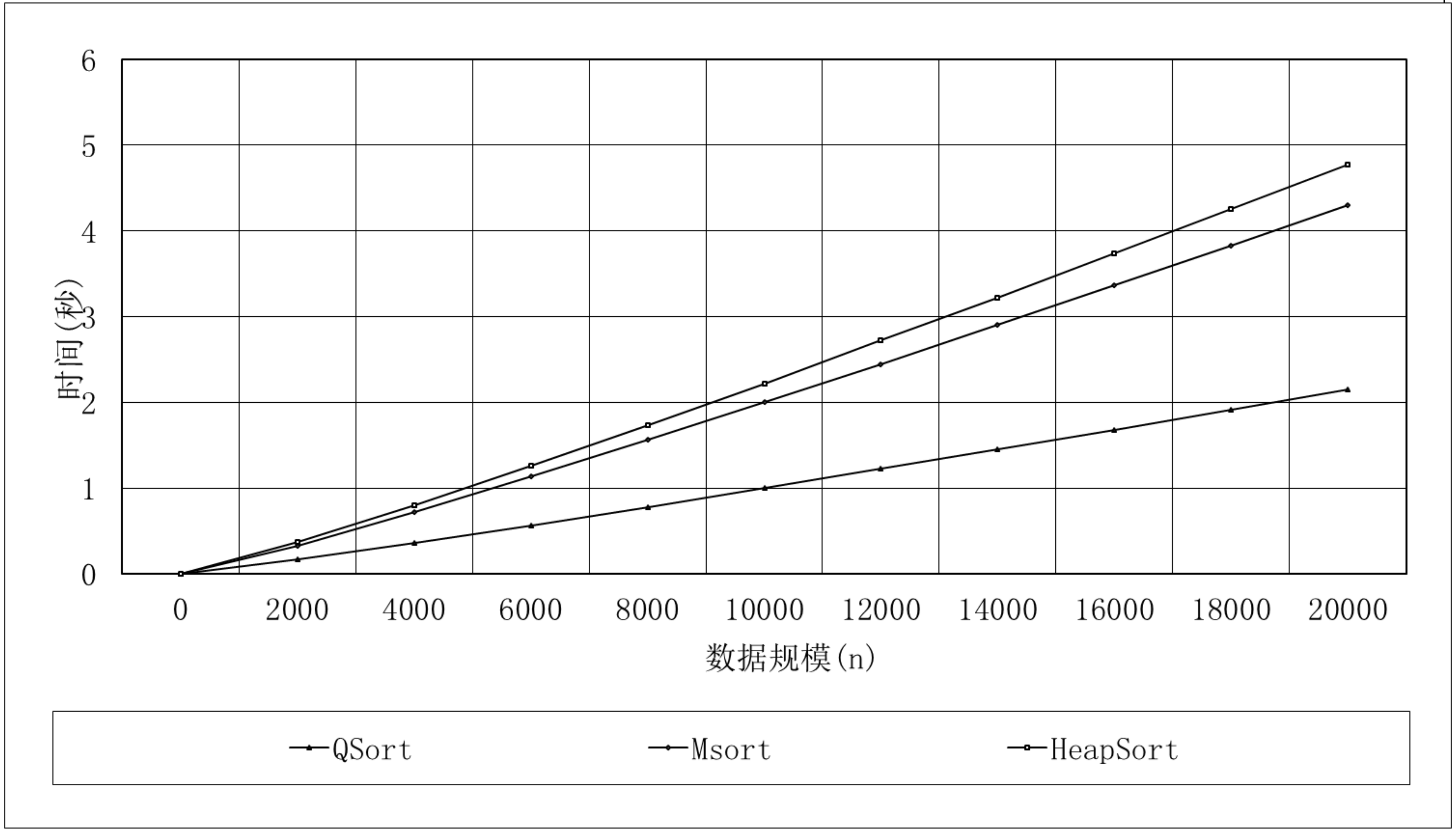
排序方法	最好	平均	最坏	辅助空间	稳定性
直接插入	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
冒泡	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
希尔		$O(n^{1.25})$		$O(1)$	不稳定
快速	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$	不稳定
堆	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$	不稳定
归并	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	稳定

平方阶排序算法示意图



120
100
80
60
40
20
0
时间(秒)
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
数据规模(n)
Bubble
Ssort
InsertSort
Shell

nlogn阶排序算法示意图



第7章 任务

- 慕课
  - 在线学习/预习 第 7 章 视频

- 作业
  - P266：7-3，7-5， 7-8，7-18，
    7-24，7-45，7-50
  - 在线提交