MergeSort

MergeSort

Strategy

将数组分为相等的两部分，递归地对其排序后，再将其合并（使用 merge）

Divide & Conquer (Easy divide, hard combination)

Merge

- 输入：有 k 个元素的数组 A 和有 m 个元素的数组 B，其中元素以非降序排列
- 输出：有 $k+m$ 个元素的数组 C，其中元素以非降序排列

基本思想：取两个数组的最小元素进行比较，两者中的较小者一定是全局最小，将其放在输出的首位，对剩下元素重复该过程

```
Merge(A, B, C)                          // recursive version
    if(A is empty)
        rest of C = rest of B
    else if (B is empty)
        rest of C = rest of A
    else
        if (first of a <= first of B)
            first of C = first of A
            Merge(rest of A, B, rest of C)
        else
            first of C = first of B
            Merge(A, rest of B, rest of C)
    return
```

在最坏情况下，最后一次比较发生在两个输入数组的最后一个元素，由于每次比较，至少有一个元素加入结果数组，而在最后一次比较只有两个元素未加入结果数组，故最多的比较次数为 $n-1$ 次（$n=m+k$）

而可得任意 merge 算法在最坏情况下至少要比较 $n-1$ 次（两个数组元素总数为 n）

只需构造如下的输入序列

$$b_0 < a_0 < b_1 < a_1 < \cdots < b_i < a_i < b_{i+1} < \cdots < b_{m-1} < a_{k-1}$$

Merge 的空间复杂度是 $O(n)$

Algorithm

MERGE-SORT (A, p, r):

```
if p < r then
    q := (p + q) / 2;
    MERGE-SORT(A, p, q);
    MERGE-SORT(A, q+1, r);
    MERGE(A, p, q, r);
```

Analysis

$$W(n) = W(\lfloor n/2 \rfloor) + W(\lceil n/2 \rceil) + n - 1$$

即

$$W(n) = 2W\left(\frac{n}{2}\right) + O(n)$$

根据 Master Theorem，$W(n) = \Theta(n \log n)$

> Number of Comparison of Merge Sort
>
> 递归树最大深度 $D = \lceil \log(n+1) \rceil$
>
> 设在深度 $D-1$ 有 B 个 base case 结点，在深度 D 有 $n-B$ 个 base case 结点
>
> 则在深度 $D-1$ 有 $\frac{n-B}{2}$ 个非 base case 结点，非递归代价为 1
>
> 故

$$W(n)=\sum_{d=0}^{D-2}(n-2^d)+\frac{n-B}{2}=n(D-1)(2^{D-1}-1)+\frac{n-B}{2}$$

$$\text{Since } (2^D-2B)+B=n, B=2^D-n$$

$$W(n)=nD-2^D+1$$

$$\text{Let } \frac{2^D}{n}=1+\frac{B}{n}=\alpha, 1\leqslant\alpha<2, D=\log n+\log\alpha$$

$$\text{So, } W(n)=n\log n-(\alpha-\log\alpha)n+1$$

$$\lceil n\log n-n+1\rceil\leqslant\text{number of comparison}\leqslant\lceil n\log n-0.914n\rceil$$

The Merge Sort D&C

思想：将问题均匀划分，然后以 $O(n)$ 的代价合并，时间复杂度为 $O(n\log n)$

计算逆序对

使用 Merge Sort 作为载体，在 Merge 时计算。

设 Merge 时的两个数组 A,B 分别有 k,m 个元素，在比较 $A[i],B[j]$ 时

- 若 $A[i]<B[j]$ ，没有发现逆序对
- 若 $A[i]>B[j]$ ，则说明 $A[i],A[i+1],A[i+2],\dots A[k-1]$ 均大于 $B[j]$，即有 $k-i$ 个逆序对

Nearest Pair

在平面上有 n 个点，求其中距离最近的一对点

思想：

Divide：将平面均匀划分，不失一般性，设按横坐标划分，即划分为 $[-\infty,L),(L,+\infty]$ 两部分

Conquer：递归求解两边的最近点对，设最近距离分别为 δ_l,δ_r

Combine：令 $\delta=\min\{\delta_l,\delta_r\}$ ，则最近点对要么是递归求解的两个结果之一，要么是跨越边界的点对，而跨越边界的最近点对横坐标只可能出现在 $[L-\delta,L+\delta]$ 之内，在其中的点最多只用和其中的 7 个点比较距离（划分出 $\delta\times2\delta$ 的矩形，若矩形中任意两点距离均不超过 δ ，则矩形中最多有 8 个点），故可在线性时间内合并（比较时可将其他点按照纵坐标升序排列，只比较前 7 个）

其余应用

Max-sum subsequence

Maxima on a plane

Finding the frequent element

Integer/matrix multiplication

详细内容 tutorial 2 中会讲解

Lower Bounds of Comparison-Based Sorting

Upper Bound and Lower Bound

> Upper bound:
>
> For **any** possible input, the cost of the **specific** algorithm A is no more than the *upper bound*
>
> $$\max\{Cost(i) \mid i \text{ is an input}\}$$
>
> Lower bound:
>
> For **any** possible (comparison-based) sorting algorithm A, the worst-case cost is no less than the *lower bound*
>
> $$\min\{Worst-case(a) \mid a \text{ is an algorithm}\}$$

关于上界，讨论的是某一特定算法，而关于下界，讨论的是解决一类问题的所有算法

> **定义 6.2** 算法问题的下界
>
> 给定算法问题 P，对于解决问题 P 的任意算法 A，若 A 的最坏情况时间复杂度总是满足 $W(n)=\Omega(l(n))$，则称 $l(n)$ 为该算法问题的最坏情况时间复杂度的下界

Decision Tree

决策树是一颗 2-tree

> **定义 B.2** 2-tree
>
> 称一颗二叉树为 2-tree，如果它的每个结点的子结点的个数均为 2 或 0。
>
> 子结点个数为 2 的结点称为内部结点（internal node），子结点个数为 0 的结点称为外部结点（external node）

决策树这一抽象数学工具可以完全刻画排序：决策树的内部结点表示两个元素 x_i, x_j 之间的比较，假设元素各不相同，则每次比较只有两种结果：$x_i < x_j$ 或 $x_i > x_j$，分别对应其左子结点和右子结点。内部结点的子结点可以是内部结点也可以是外部结点，内部结点表示下一次比较，外部节点表示的是排序结果。

任何基于比较的排序算法一次执行过程可以看作决策树上从根结点到子结点的一条路径，路径长度为算法代价。

- n 个不同元素的序列有 $n!$ 种排列，即决策树至少有 $n!$ 个叶结点
- 算法的最坏情况时间复杂度即为根到叶结点的最长路径长度，也即决策树的高度
- 算法的平均情况时间复杂度为根节点到不同叶结点路径长度的加权和，取决于输入的概率分布

Lower bounds for Worst-case

> **引理 6.1** 假设一颗二叉树的高度为 h，叶结点个数为 L，则有：$L \leqslant 2^h$

根据引理 6.1 和之前结论（决策树至少有 $n!$ 个叶结点），有

$$h \geqslant \log L = \log(n!)$$

$$\log(n!) \geqslant \log(n(n-1)\dots(\lceil n/2 \rceil)) \geqslant \log(\frac{n}{2})^{\frac{n}{2}} = \frac{n}{2}\log(\frac{n}{2}) = \Theta(n\log n)$$

即任何基于比较的排序算法对于 n 个元素的输入至少要比较 $\log(n!)$ 次，即大约 $\lceil n\log n - 1.443n \rceil$ 次

> **定理 6.1** 比较排序算法的最坏情况时间复杂度为 $\Omega(n\log n)$

Lower bounds for Average-case

> **定义 6.3** 外部路径长度（EPL）
>
> 定义一棵树 T 的外部路径长度 $EPL(T)$ 为根结点到所有叶结点的路径长度之和
>
> - 只含有唯一一个根结点/叶结点的树 $EPL = 0$
> - 一棵树的左子树，右子树分别为 T_L, T_R
>
> $$EPL(T) = EPL(T_L) + N_L + EPL(T_R) + N_R$$
>
> N_L, N_R 为左右子树的结点个数

设决策树叶结点个数为 L 基于外部路径长度，可得基于比较的排序算法的平均情况时间复杂度为 $\frac{EPL}{L}$，而所有比较排序的平均情况时间复杂度下界即为 $\frac{EPL}{L}$ 的下界

> **引理 6.2** 越平衡的 2-tree 有越少的 EPL
>
> 证明：若一个二叉树的高度为 h，且其中存在一个叶结点深度 $k \leqslant h-2$，则存在同样结点数和同样叶结点数的更平衡的一颗 2-tree 具有更小的 EPL
>
> 可直接构造，设 2-tree 中的深度为 h 的叶结点的父结点为 Y，且存在深度为 $h-2$ 的叶结点为 X，则将 Y 的两个子结点给 X，易得原先计算 EPL 时 X 和 Y 的两个子结点贡献 $h+h+k$，现在贡献 $(h-1)+2(k+1)$，EPL 增长 $k-h+1$，而根据假设 $k \leqslant h-2$，可得 $k-h-1<0$，EPL 减小

对于一颗平衡的 2-tree，其叶结点数为 L，高度为 $\Theta(\log L)$，$EPL=\Theta(L\log L)$

> $epl \geqslant m\log m$, where m is the number of external nodes in t
>
> - $epl = epl_L + epl_R + m \geqslant m_L \log m_L + m_R \log m_R + m$
>   - $f(x)+f(y) \geqslant 2f(\frac{x+y}{2})$ for $f(x)=x\log x$
> - $epl \geqslant 2(\frac{m_L+m_R}{2})\log\frac{m_L+m_R}{2} + m = m(\log m - 1) + m = m\log m$

故平均情况时间复杂度

$$\begin{aligned} A(n) &= \frac{EPL}{L} \\ &= \Omega(\frac{L\log L}{L}) \\ &= \Omega(\log n!) \\ &= \Omega(n\log n) \end{aligned}$$

基于比较的排序算法对 n 个元素的输入平均要比较 $\log n!$ 次，即大约 $n\log n - 1.443n$ 次

比较排序算法的平均情况时间复杂度为 $\Omega(n\log n)$

对于合并排序，由于平均情况时间复杂度一定不超过最坏情况时间复杂度，
$A(n)=O(W(n))=O(n\log n)$，而根据上文结论，对于任意排序算法有
$A(n)=\Omega(n\log n)$，故合并排序的平均情况时间复杂度 $A(n)=\Theta(n\log n)$，合并排序有较好的平均情况时间复杂度