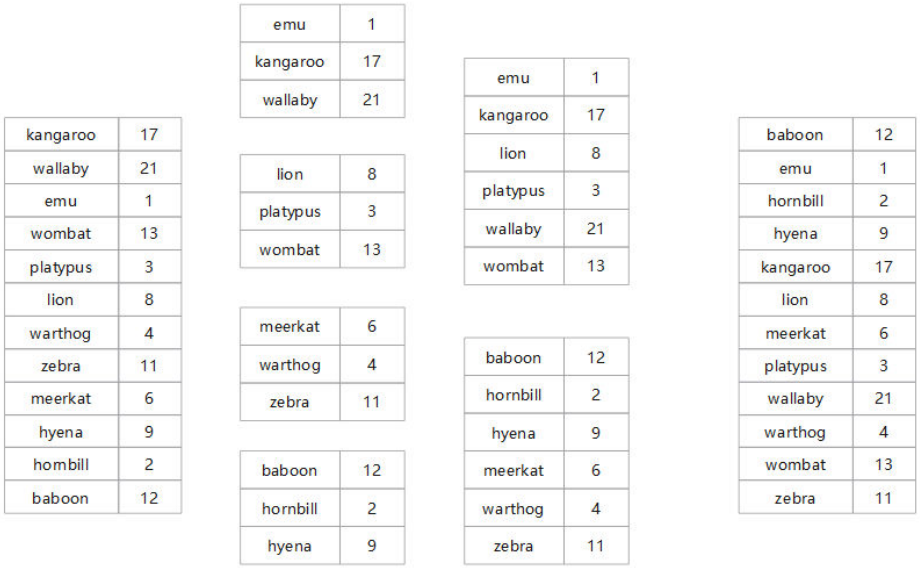
1. **假设一个块中只能容纳一个元组，内存最多可以容纳三个块。当应用于对第一个属性排序以下元组时，请展示排序合并算法的每个传递中创建的运行：** (kangaroo, 17), (wallaby, 21), (emu, 1), (wombat, 13), (platypus, 3), (lion, 8), (warthog, 4), (zebra, 11), (meerkat, 6), (hyena, 9), (hornbill, 2), (baboon, 12)。



kangaroo	17
wallaby	21
emu	1
wombat	13
platypus	3
lion	8
warthog	4
zebra	11
meerkat	6
hyena	9
hombill	2
baboon	12

emu	1
kangaroo	17
wallaby	21

lion	8
platypus	3
wombat	13

meerkat	6
warthog	4
zebra	11

baboon	12
hornbill	2
hyena	9

emu	1
kangaroo	17
lion	8
platypus	3
wallaby	21
wombat	13

baboon	12
hornbill	2
hyena	9
meerkat	6
warthog	4
zebra	11

baboon	12
emu	1
hornbill	2
hyena	9
kangaroo	17
lion	8
meerkat	6
platypus	3
wallaby	21
warthog	4
wombat	13
zebra	11

2. **设 r 和 s 是没有索引的关系，并假设这些关系没有排序。在假设有无限内存的情况下，计算 r ⋈ s 的最低成本方法（以 I/O 操作为代价）是什么？这个算法需要多少内存?**

在没有索引且关系没有排序的情况下，计算 $r \bowtie s$（即两个关系的自然连接）的最低成本方法是使用**嵌套循环连接（Nested-Loop Join）**算法。这种算法的基本思想是对于 r 中的每一行，都与 s 中的所有行进行比较，找到满足连接条件的行。这是一种简单但成本较高的连接算法，其计算成本是 $|r| \times |s|$。

具体步骤如下：

1. 对于关系 r 中的每一行 r_i，从关系 s 中开始顺序扫描每一行 s_j。
2. 对于每一对行 (r_i, s_j)，检查它们是否满足连接条件。
3. 如果满足条件，将它们加入结果中。

这个算法的主要特点是简单易懂，但由于每个 r 中的行都需要与 s 中的所有行比较，因此成本较高。在没有索引和排序的情况下，这是一个适用的方法。

至于内存需求，嵌套循环连接算法需要内存来存储两个关系中的行，因此内存需求是 $|r| + |s|$即单个的数据行相加。这是因为算法的基本操作是在内存中对两个关系进行顺序扫描，逐行比较。在没有排序的情况下，这是一种相对较低内存需求的算法。

其时间复杂度是 $O(|r| \times |s|)$，在数据规模较大时可能会导致性能问题。

3. **假设您需要对一个大小为 40 GB 的关系进行排序，每个块为 4 KB，使用 40 MB 的内存。假设寻道的成本为 5 毫秒，而磁盘传输速率为每秒 40 MB。**

    1. **计算在 b_b = 1 和 b_b = 100 的情况下，对关系进行排序的成本，以秒为单位。** 按照书本上的成本计算公式：

        - b_b = 1 $b_r = 1 \times 10^7$ $M = 1 \times 10^4$ 结果为：60012s
        - b_b = 100 $b_r = 1 \times 10^7$ $M = 1 \times 10^4$ 结果为：1812s

    2. **在每种情况下，需要多少次归并操作?**

        - b_b = 1 一共1次归并操作
        - b_b = 100 一共3次归并操作

    3. **假设使用闪存存储设备代替磁盘，其延迟为 20 微秒，传输速率为每秒 400 MB。在这种设置中，重新计算在 b_b = 1 和 b_b = 100 的情况下，对关系进行排序的成本，以秒为单位。**

        即每次传输磁盘都需要 $1 \times 10^{-5} + 2 \times 10^{-5} = 3 \times 10^{-5}s$

        通过公式计算得到

        - b_b = 1 结果为：300.06s
        - b_b = 100 结果为：18.12s

4. **设计一个混合 归并-连接 算法的变种，其中两个关系都没有物理排序，但它们在连接的属性上都有一个排序的次级索引。**

    设计一个混合归并-连接算法的变种，其中两个关系都没有物理排序，但在连接的属性上都有一个排序的次级索引，可以按照以下步骤进行：

    1. **输入参数：**

- 两个关系R和S，它们分别具有连接属性A和B。
- 次级索引：R上的次级索引为A，S上的次级索引为B。

2. **预处理：**

- 遍历关系R，构建一个以A为键的哈希表或树形结构，将记录按照A的值进行组织。
- 遍历关系S，构建一个以B为键的哈希表或树形结构，将记录按照B的值进行组织。

3. **合并阶段：**

- 对关系R和S的连接属性进行归并排序，得到两个有序的序列R_sorted和S_sorted。
- 初始化两个指针分别指向R_sorted和S_sorted的开头。

4. **连接操作：**

- 循环遍历排序后的序列R_sorted和S_sorted。
- 对于当前指针指向的元素，比较A和B的值。
- 如果A小于B，则移动R_sorted的指针，反之移动S_sorted的指针。
- 如果A等于B，说明找到了连接的匹配，将匹配的记录输出。

5. **性能优化：**

- 对于每个A值，在次级索引中可能有多个匹配的B值，因此可以采用一些策略来减少比较次数，比如跳跃指针等。

6. **复杂性分析：**

- 归并排序的时间复杂度为O(n log n)，连接操作的复杂度取决于连接的记录数量。
- 空间复杂度主要取决于哈希表或树形结构的大小，一般为O(n)。

5. **如果每个运行时缓冲块数量都会增加，而用于缓冲运行的整体内存保持不变，对于合并操作的运行的成本会有什么影响?**

如果每个运行时缓冲块的数量增加，而用于缓冲运行的整体内存保持不变，对合并操作的运行成本会有一些影响。让我们考虑一下可能的影

响：

1. **更大的内存缓冲：**

- 增加每个运行时缓冲块的数量可能意味着每个缓冲块可以容纳更多的元素，因此每次从磁盘读取的次数可能减少。这可以提高I/O效率，减少了磁盘访问的开销。

2. **更高的内存开销：**

- 随着每个缓冲块的数量增加，整体内存中存储缓冲块所需的内存开销可能会增加。这可能导致更频繁的内存管理操作，例如分配和释放内存，这可能会对性能产生一定的影响。

3. **更大的排序开销：**

- 如果合并操作涉及对缓冲块进行排序，增加每个缓冲块的数量可能导致更大的排序开销。排序本身的复杂度可能取决于缓冲块的大小，因此更多的元素可能意味着更多的比较和交换操作。

4. **更高的并发性：**

- 增加缓冲块的数量可能使得在内存中并行处理更多的数据块成为可能。这可以提高合并操作的并发性，特别是在多核系统中，从而提高整体性能。

5. **更少的外部合并次数：**

- 由于每个缓冲块可以容纳更多元素，外部合并的次数可能减少。这有助于减少磁盘访问的开销，因为外部合并通常需要将数据写回到磁盘。