第 1 题(本题2分)：以下哪个因素不会显著影响B树的访问性能?

- A：树的高度 ✗
- B：树的阶 ✗
- C：节点的空间大小（通常一个节点为存储空间中的一页，因此可理解为页的大小） ✗
- D：节点内部的数据充满度 ✓

第 2 题(本题2分)：B树的平衡性主要由哪条性质保证?

- A：每个节点的大小固定 ✗
- B：每个节点的充满度都超过1/2 ✓
- C：叶子节点上的数据是有序的 ✗
- D：以上性质都不能 ✗

第 3 题(本题2分)：如果我们在属性price上创建一个索引（比如使用指令db.myColl.createIndex({ price: 1 }) ），那么以下哪个查询可以无法从这个索引获益?

- A：db.myColl.findone({ category:"apple", price:20 }) ✗
- B：db.myColl.findone({ category:"apple" }) ✓
- C：db.myColl.findone({ price:{$gte:20, $lte:30} }) ✗
- D：db.myColl.findone({ category:"apple", price:{$gte:20, $lte:30} }) ✗

第 4 题(本题2分)：如果我在多个属性上创建一个复合索引，例如db.myColl.createIndex({ score: 1, price: 1, category: 1 })，那么以下哪个查询无法从索引获益?

- A：db.myColl.find({ category:"apple", price:20, score:5 }) ✗
- B：db.myColl.find({ score:{$gte:4} }) ✗
- C：db.myColl.find({ category:"apple", price:{$gte:20, $lte:30} }) ✓
- D：db.myColl.find({ category:"apple", score:{$gte:4} }) ✗

在数据库中创建复合索引时，索引的字段顺序是至关重要的。对于复合索引 { score: 1, price: 1, category: 1 }，它首先按 score 排序，然后是 price，最后是 category。

基于这个复合索引的结构，我们来看每个查询：

A: 查询使用了所有索引字段并且它们都是等值查询，因此可以完全利用索引。

B: 查询只使用了索引的第一个字段 score，它可以利用索引进行范围查询。

C: 这个查询在 category 和 price 上有条件，但由于 score 是索引中的第一个字段，而且这个查询没有使用 score 字段，所以它不能充分利用索引。这是因为索引的字段是有序的，**而且 MongoDB 在使用复合索引时，只有当查询条件使用了索引的最左边的字段时，索引才是最有效的。**

D: 查询使用了索引的第一个字段和最后一个字段，但是由于它没有使用 price 字段，它也

可以利用索引，但不如查询 A 那样有效。
因此，选项 C 是正确答案，因为它无法充分从索引获益，索引的第一个字段 score 在查询中没有被使用。在复合索引中，如果不按索引定义的顺序进行查询，即不遵循最左前缀原则，那么查询就无法充分利用索引。

第 5 题(本题2分)：请问以下哪种情况最适合使用索引?

- (●) A：属性a常用作查询条件，属性b频繁被修改。在a上创建索引。 ✓
- ( ) B：属性a常用作查询条件，属性b频繁被修改。在b上创建索引。 ✗
- ( ) C：属性a常用作查询条件，文档频繁被插入和删除。在a上创建索引。 ✗
- ( ) D：属性a常用作查询条件，属性a频繁被修改。在a上创建索引。 ✗