Comparing Dictionary Sorting Methods in Python: Efficiency Analysis

When comparing two approaches for sorting a dictionary by its keys, the more efficient solution is the one that directly sorts all dictionary items without additional filtering. This is the method used in the Copilot implementation, which applies a straightforward sorting process to all key-value pairs. It benefits from simplicity and speed, performing a single pass using a built-in sorting function with a time complexity of O(n log n).

```
def sort_by_key(d):
    return dict(sorted(d.items()))
```

In contrast, the alternative version first filters the dictionary to include only numeric keys, then performs the sorting operation. While this may be necessary in scenarios where the dictionary contains mixed key types that could cause errors during sorting, it introduces an extra computational step. This initial filtering adds a linear time complexity of O(n), followed by the same O(n log n) sorting phase, making it less efficient overall for dictionaries that don't require type-based filtering.

```
def sort(d):
    numeric_items = {k: v for k, v in d.items() if isinstance(k, (int,
float))}
    return dict(sorted(numeric_items.items()))
```

Memory usage is also higher in the filtered approach due to the creation of an intermediate dictionary. Therefore, unless the dictionary includes incompatible key types that must be excluded, the Copilot method is the more efficient and practical choice. It reduces overhead, avoids unnecessary operations, and leverages Python's optimized sorting capabilities directly on the input data.