If one block can store 6 records, and each record is 40 bytes, a single block used for storing records would be:

6 records/block * 40 bytes/record = 240 bytes

Considering the block size is 256 bytes, you'd have 16 bytes of unused space per block for record storage. This space might be used for record metadata or remain unused due to alignment.

Now, let's consider the B+ tree index. The B+ tree will have internal nodes and leaf nodes:

- **Leaf Nodes**: These nodes contain pointers to the actual records. If a block can store 6 records, each leaf node in the B+ tree will point to 6 records.
- **Internal Nodes**: These nodes contain keys and pointers to other nodes (either leaf nodes or other internal nodes).

The size of the keys and pointers within the B+ tree nodes will define how much space each node occupies. In a typical filesystem or database implementation:

- **Keys**: Might be the same as the primary key of the records.
- **Pointers**: Could be the size of the disk block reference (which may be 4 bytes, 8 bytes, or more, depending on the system and implementation).

For simplicity, let's assume:
- Each key is an integer (4 bytes).
- Each pointer is 8 bytes (typical on 64-bit systems).

A B+ tree node will contain `d` keys and `d+1` pointers, where `d` is the order of the B+ tree. If we can store 6 records per block, the order of the B+ tree (`d`) might be considered to be 6 for leaf nodes, as each block corresponds to a leaf node. For internal nodes, you might have a higher order depending on how many child pointers you can fit into a block.

Assuming that an internal node just contains pointers (and minimal metadata), and each block can contain a complete node, the number of child pointers an internal node can hold is given by:

256 bytes/block / 8 bytes/pointer = 32 pointers/block

So, each internal node can have up to 31 keys (since with `d` keys, there are `d+1` pointers), and we'll use this as the internal node order for our B+ tree.

With these assumptions, you'd estimate the space needed for the B+ tree by considering how many blocks are needed to index all the records. You have 10,000 records, so you'll need:

10,000 records / 6 records/block = 1,667 leaf blocks (approximately)

For the internal nodes, it's more complex because you'd need to calculate the depth of the tree. For a B+ tree, all leaf nodes will be at the same depth. The number of internal nodes you need will depend on the branching factor (the order) of the tree, but you can estimate it using a binary tree assumption to give a rough idea:

If every internal node points to 32 child nodes (31 keys and 32 pointers), then the total number of internal nodes can be roughly estimated by the number of leaf node levels:

- Level 1 (closest to leaf nodes): 1,667 leaf blocks / 32 = 53 nodes (rounded up)
- Level 2: 53 nodes / 32 = 2 nodes (rounded up)
- Level 3: 2 nodes / 32 = 1 node (root)

This gives us a very rough estimate of 53 + 2 + 1 = 56 internal nodes (this is highly simplified and the actual number might be different because B+ trees spread out more than binary trees).

Total blocks needed for the B+ tree index would then be approximately:

1,667 leaf blocks + 56 internal blocks = 1,723 blocks

To get the total bytes for the B+ tree:

1,723 blocks * 256 bytes/block = 441,088 bytes

Please note this is a simplified calculation and assumes perfect filling of the B+ tree, which in practice is not the case. The actual implementation may vary, and overhead for node metadata, as well as partially filled nodes, can change the calculation significantly. It's also possible to have more than two levels of internal nodes depending on the distribution of keys and other implementation details.