A1 Question 1

To implement pattern matching with a pattern containing special characters, I took an approach using the Z-Algorithm. The idea here is to break up the pattern into blocks of literals (substrings with no wild cards) and also store their offsets in the original string. Then, we can run the Z-algorithm to search for those blocks in the text. Once we find the occurrences of each of the blocks in the text, we can use the positions and offset to map each occurrence back to the index where the pattern would begin, and check which positions have every block correctly match back, which means a full match.

An example of the blocks for the pattern “#ab#c#” would be:

Blocks: [“ab”, “c”]

Offsets: [1, 4]

Each offset corresponds to the index in which a block begins on the original pattern.

The time complexity of my solution is O(N*B + M) where N is the size of the text, B is the number of blocks of literals, and M is the size of the pattern.

Preprocessing the string to split the pattern up requires an iteration through the pattern, O(M). Then we have B, the number of blocks, and we have to pattern match each block to the text. Pattern matching is linear time because it’s done with the Z algorithm, which requires O(N+|block|) for each block. So doing that B times would be O(B(N+|block|)). However, since we know that the sum of the lengths of the blocks are bounded by M, we can instead write this as O(B*N + M), since each block needs to have the Z algorithm run on it, and the sum of the |block|s (block lengths) is bounded by M.

Then iterating through the block occurrence indices to find the full matches takes O(B*N) since there are B blocks and the number of occurrences is bounded by B.

In total, O(M) + O(B*N + M) + O(B*N) which evens out to O(N*B + M) time complexity.

Note that the efficiency of this approach depends heavily on the number of blocks. In the best case, If the number of blocks is low, then the time complexity can approach O(N+M), but in the worst case, if the number of blocks scales linearly with M, then it’s more like O(N*M+M) time which is O(N*M).

The space complexity of my solution is

Splitting the pattern up initially needs two arrays of size B, the number of blocks. One of those arrays, the “blocks” array, stores the substrings themselves, so that one’s got a bound of M, so O(B+M). B cannot be lower than M, so this is actually O(M)

Then for searching for each block in the text, for each block, we need to create a combined string of size b + N where b is the size of each block. The Z algorithm creates a Z array of the same size. This auxiliary space can be reused for each block. The size of a block b is bounded by the size of M, so the space complexity is O(M+N) for this step.

Then, to map it back to get full matches, we need an array of size B. B is bounded by M so this is O(M)

In total, O(M) + O(N+M) + O(M) = O(N+M) space complexity.